Information Retrieval Project Report

Ruoting Zheng
Rui Cao

Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, 66045, KS

1. Introduction

Information retrieval (IR) is a process of locating and returning the relevant material to a user's query. In classical SQL queries of database, the structured data are exactly matched to the query based on the ranking defined in query. However, in IR system, a bunch of unstructured objects (i.e. text document, image, audio) that satisfy the information need are searched and ranked from within large collections [1, 2].

The computer-based IR systems were started in 1950s [4]. This traditional text retrieval system is the version of web search 1.0 [3]. Two important tasks for an IR system were developed at that time: index documents and retrieve them [4]. In 1952, M. Taube et. al proposed to index items using a list of keywords [4] which is still used today. The so-called Boolean model was used to search and retrieve documents. In the Boolean model, each query was a logical expression of terms, and the returned documents were those which exactly matched the query [1, 4]. In 1960s, Prof. Salton at Cornell University, who is one of the pioneer researchers in the field of IR, suggested to rank the similarity between a document and query vector using cosine coefficient [1, 4]. The idea of relevance feedback was also introduced at this time. This process iteratively re-ranks the documents based on the feedback from users. One of the examples of relevance feedback is the "related articles" link on Google Scholar [4]. In 1970s, the theories of term frequency (tf), inverse document frequency (idf) and the weight of tf*idf were developed to rank the relevant documents and queries [1, 4]. Since mid-1990s to 2000s, machine learning has been used to learn the ranking, i. e. Rocchio's relevance feedback algorithm [1, 4]. Since the year of 2012, the search engine giants, i. e. Google and Baidu have been incorporating deep learning to their search engines [5].

In this final project, two versions of search engine have been implemented and tested using different queries. The niche crawler with multi-threaded spiders has been developed to crawl web page sources from KU domain. Processing of documents includes removing html tags, tokenizing, removing stop words and stemming. The queries are processed using stop words list and stemmer. Boolean model and Vector Space model have been constructed in the search engine 1. Search engine 2 has been built by adding Term proximity to search engine3. user interface of the information retrieval system has been developed to demo the performance of the two versions of search engine.

2. Abstract

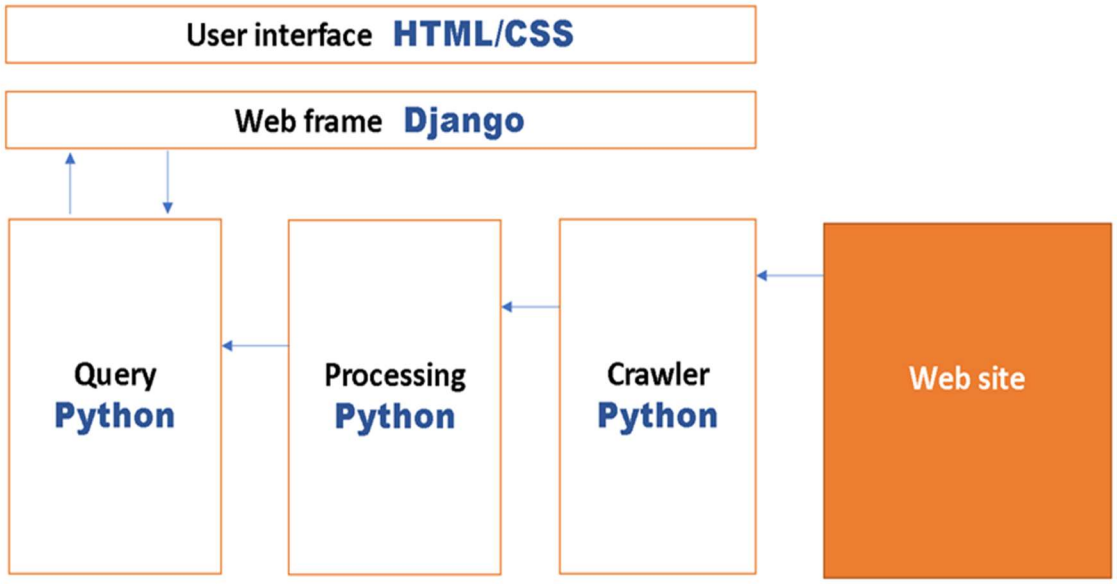
In this project, we aimed to implement an information retrieval system that contains the following components:

(1) Document processing and indexing: in this part, we need to remove HTML tags, convert all letters into lower case, implement a stop list and a stemmer for pre-processing document and query. Then, it should build an inverted index (including dictionary and posting lists) for documents.
(2) Vector space model: The goal is to provide a TF-IDF-based ranking for the documents.
(3) Niche crawler: The crawler can identify a domain of interest and should contain at least three components: (1) a multi-threaded spider that fetches and parses webpages, (2) the URL frontier which stores to-be-crawled URLs; and (3) the URL repository that stores crawled URLs.
(4) Feed the collected documents to the search engine; implement a Web-based interface to take user queries and return answers.
(5) Add term proximity into scoring mechanism
(6) evaluate and compare the performance of the original search engine (step 4), and the new versions (5) and (6)

3. Programming Platform:

The program is based on Python 3.5.2 and related package and module.
The web interface is implemented with Django frame (and Bootstrap 3.3.7 for CSS).

The architecture of the system is showed as follow:

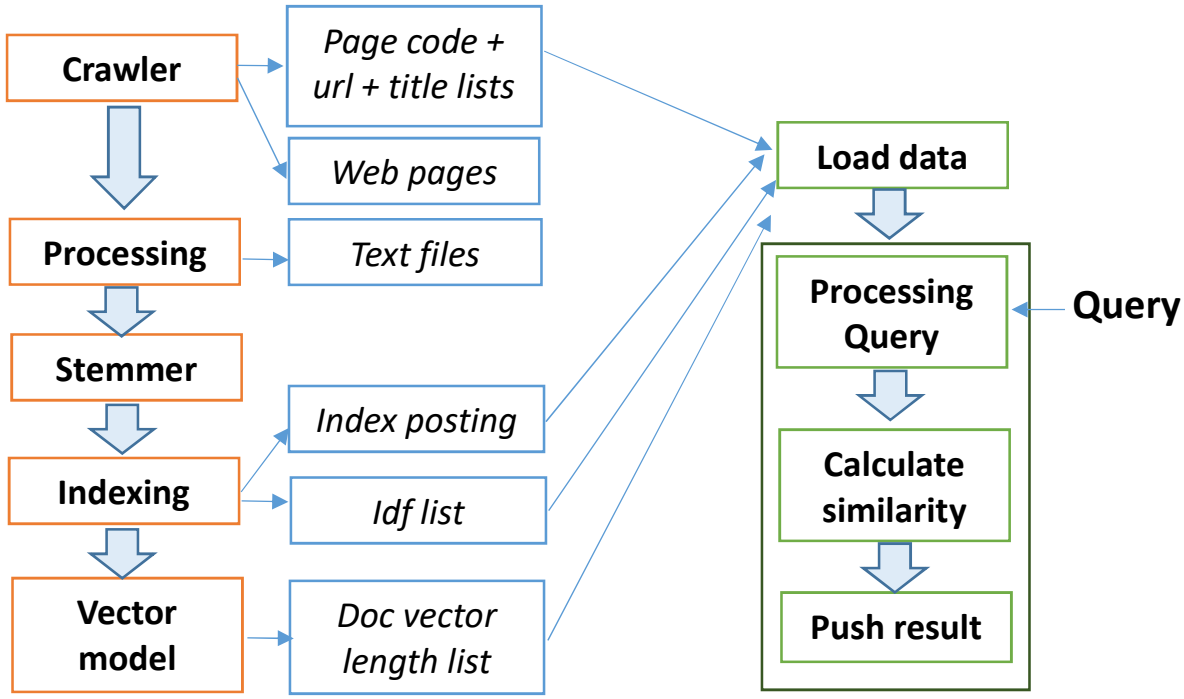


4. Design of our search engine

Pre-processing and process

In this final project, two versions of search engine have been implemented. The topology of our search engine is shown in figure 2. The web crawler with multi-threaded spiders has been developed to feed crawled web pages for the three search engines. As shown in figure 2, the documents were preprocessed with the following four steps.
a). Remove the HTML tags
b). Excluding punctuations and special characters
c). Convert into lower case
d). Using NLTK package to remove words in a stop list and do Porter stemming

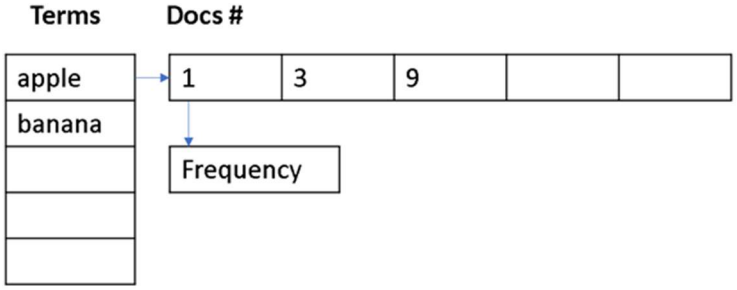
After processing each document into an independent list of tokens (without removing duplicates) we decided to hash these tokens into a dictionary like structure using the standard library hash table. The idea was to use the terms as keys for the dictionary and their respective definitions (i.e. values of the keys) would be a data structure representing the statistics of the term in the entire collection.



5. Data Structures and methods

In pre-processing, we first removed all html tags from all html pages. Then we used NLTK package to remove stop list and do Porter stemming. Next step is to build inverted index dictionary with posting list, which contains related TF and DF information. The data structures we used is dictionary. Based on the inverted index, we built the TF-IDF table for all the documents. Boolean model and Vector Space model have been built based on the TF-IDF table. The relevance between documents and query is calculated using cosine similarity.

First, build the index posting. We use hash table for all the elementary structures to speed up the query. The structure of index posting is as following:



For building vector space model, we follow these steps:

(1) Calculate the inverse document frequency(idf) for each term in dictionary.
(2) Calculate the tf-idf weight for each term in different documents by using the tf and idf information we got from previous steps. The formula is :

$$w_{t,d} = tf_{t,d} * idf_t = tf_{t,d} * \log_{10} \frac{N}{df_t}$$

At the same time, calculating the length of each vector. Also calculating the length of query vector when the users input the query. Then, use Boolean model to select related documents as candidate documents (at least one same term). Finally, calculating the similarity between query and each candidate document. We use cos similarity formula:

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \bullet \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

6. Web crawler

Web crawler is used to traverse and gather web pages in specific domain on the internet. The topology of web crawler is shown in following figure:

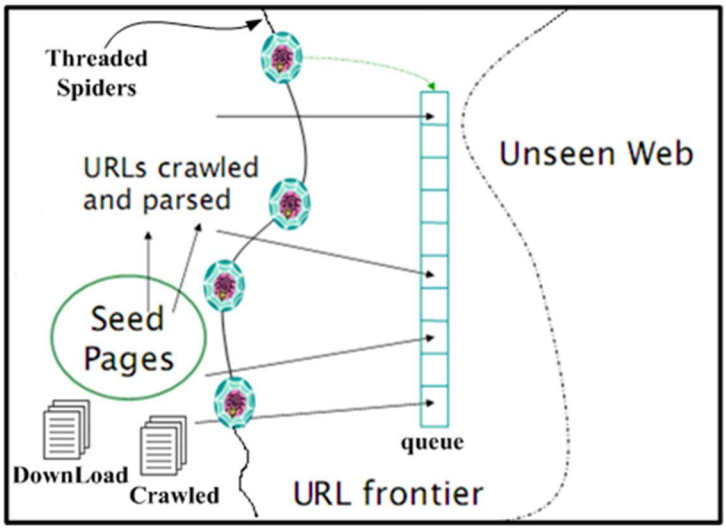


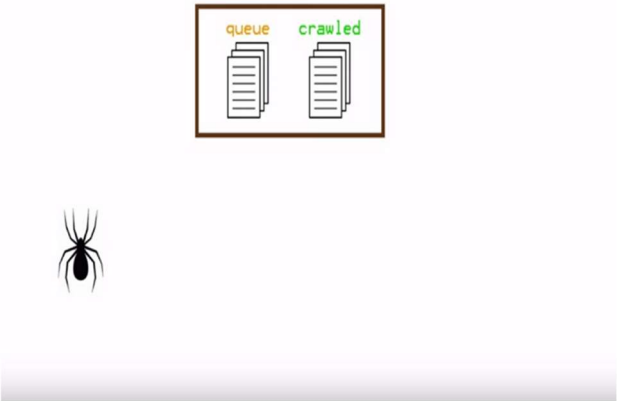
Fig. The topology of web crawler

The web crawler contains the following three components.

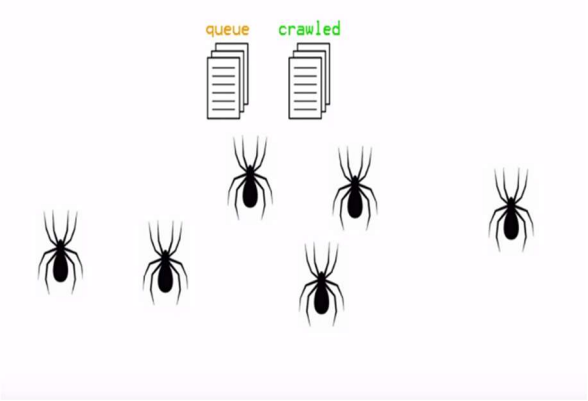
a). HTMLParser, which inherits from the class of HTMLParser in the module of HTMLParser in python. In this component, the web page is fetched and parsed, and then the relative urls are normalized to absolute urls using urljoin().

b). Spider.
Data structure： set.
Queue： URL to crawl. Crawled: URL crawled. All Spider share Queue set and Crawled set. Creates directory and files for project on first run and starts the spider. Updates user display, fills queue and updates files, converts raw response data into readable information and checks for proper html formatting, Saves queue data to project files.

c). Crawler. Given a seed page, the urls in this seed page is parsed and stored in queue using one spider. Then, multi-threaded spiders are used to crawl web pages. Each spider gets one task from the queue, and the spiders won't stop crawling until the queue is empty. The crawled web pages are then feed into the two version of search engine



First Spider：
Base url: https://www.ku.edu/
Domain name: ku.edu



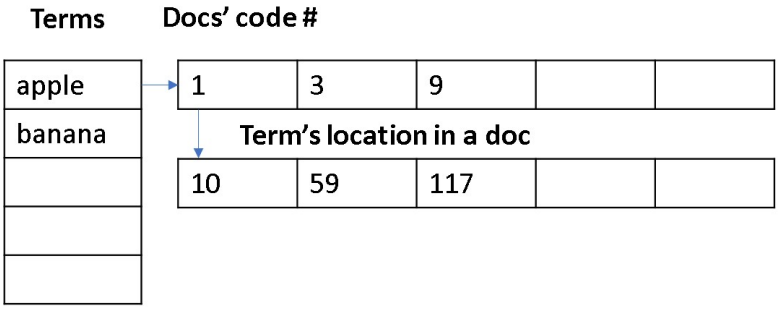
Multiple threads to crawl the pages.

7. Performance

Structure	Space consuming
Idf_list	1.2 MB
Posting list	3.4 MB
Others (url dict, etc)	Within 100KB
Process	**Time consuming**
Query	0.02 sec on average
Document processing	Within 1 min
Indexing	1.2 sec
VS-model preparation	0.4 sec

8. Term proximity

We made modification on the index posting to contain the location info into structure.



To calculate the proximity, the idea is to find the shortest distance between the different query terms in a web page.

For example, if we have two query terms, music and exam. And we have two lists of numbers showing their locations in a page.

Music [1,3,8,16]

Exam [10,21,57]

The shortest distance between 'music' and 'exam' in this file is 2 (10-8 =2).

In another word, we are finding the smallest window to cover two terms.

The algorithm for finding the shortest distance:

For finding the shortest distance between the two lists of locations, instead of calculating all the distance and get the minimum one (for which the time complexity is O(M*N)), we developed a method with only O(M+N) complexity. The fake code is shown as following:

```
Input: two lists of locations, a and b
V = a large number
For (i=0, j=0; i<len(a) && j<len(b);  ){
    X = a[i] – b[j]
    If (x>0):
        j++;
        if (x<v) v=x; continue;
    If (x<0):
        i++; x*=-1;
        if (x<v) v=x; continue;
    If (x==0):
        V=0; break;
```

}

Output: V

And for the query which include three or more terms, we calculate the average shortest distance. For all distinct query terms, we first get the distance in each pair of terms, and then calculate the average.

N = number of terms Number of pairs = N*(N-1)/2

Terms	Pairs
3	3
4	6
5	10

Average shortest distance

And for adding the proximity into the score, we are using the following formula:

Score = Cosine similarity + (1/average shortest distance of terms) * m

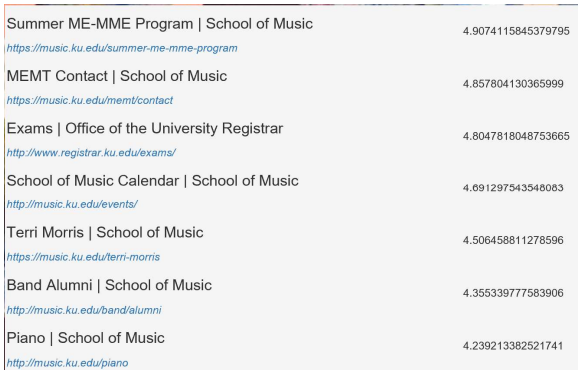
The m is a constant.

9. Comparation between two version of system

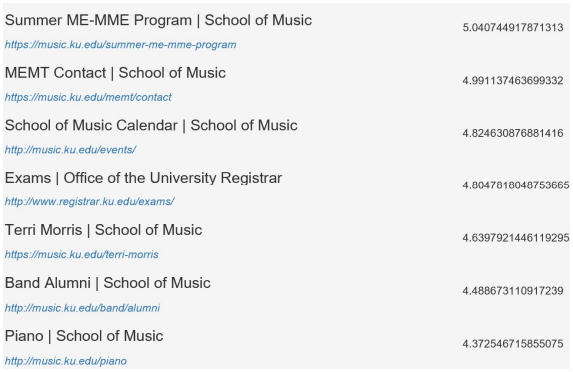
Query: Music && exam

Origin search

Proximity const = 0.1



Proximity const = 0.5



We made a comparation between the system without and with proximity approach.

It can be seen from the snipping that with proximity const = 0.5, the rank changed a little. The calendar of the music school now ranking higher than the web page from exam office since it shows higher relation between term music with term exam. The rank of result improved.

10. How do I get set up?

* Summary of set up the crawler

1. cd into the “crawler” directory (please put all the code files in the same folder)
2. Run main.py "python main.py"

The path has already been set in the program and the folders needed will be created.

* Summary of set up the pre-processing program

1. cd into the “info_retrieve” directory (please put all the code files in the same folder)
2. Run main.py "python main.py"

The path has already been set in the program and the folders needed will be created.

* Server Deployment instructions (How to start up the server)

1. cd into the “web_server” directory (prototype search)
2. Run “python manage.py runserver”
3. The web set is deployed on 127.0.0.1:8000. Please open the browser, and open the http://127.0.0.1:8000/index/.

Start your journey of exploring websites.

Reference

[1] Introduction to Information Retrieval, by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Cambridge University Press. 2008.
[2] https://en.wikipedia.org/wiki/Information_retrieval
[3] http://research.microsoft.com/en-us/collaboration/global/asia-pacific/talent/webirhistoryfuturetrends.pdf
[4] Sanderson, Mark, and W. Bruce Croft. "The history of information retrieval research." Proceedings of the IEEE 100.Special Centennial Issue (2012): 1444-1451.

] http://www.wired.com/2016/02/ai-is-changing-the-technology-behind-google-searches/