STEP BY STEP GUIDE TO

Building A Mini Search Engine

Rajendra Banjade

12/4/2012

1 Introduction

Search engines are programs that search documents for specified query and return a list of the relevant documents. Most of the today's search engines order the retrieved documents (usually huge in number) based on some relevancy criterion and return them sorted. Depending on the factors considered for setting the premise for the relevancy, various retrieval models exist. A retrieval model specifies the details of document representation, query representation and retrieval function. It also determines a notion of relevance. Some of the general retrieval models are Boolean model, Vector space model, Probabilistic model etc. In Boolean model, a document is represented as a set of keywords. Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate scope. Though Boolean model is very simple, the model outputs all the documents that satisfy the query and it doesn't have partial matching and ranking capacity. In the vector space model, each document is represented as vector of terms with certain weight. When user provides a query, that query is also expressed as a vector having same dimensions as that of the document vectors. The similarity of query vector is calculated with documents having at least one word from the query and ranked based on that score.

Vector space model has been implemented in the mini search engine. It is fairly effective and simple to implement. This retrieval engine has Ad hoc retrieval function (i.e. it has fixed document corpus but is able to serve various queries related to the document collection).

The operations of retrieval engine are:

- Web crawling
- Preprocessing
- Indexing
- Retrieving
- Ranking
- Presentation

Retrieval engine works by storing information about many web pages from the various websites. Given the seed URL, the crawler program (sometimes also known as a spider) gets the pages by automatically following the connected web links. Exclusions can be made by the use of robots.txt. If the content owners do not want their information be crawled by the web crawlers, they specify that resources in the robot.txt file. Search engine use spiders as a means of providing up-to-date data and to create a copy of all the visited pages for later processing. The number of documents has been limited to 1500 and they are downloaded using breadth-first approach.

The contents of each page are preprocessed to enhance the retrieval performance of the engine. They are analyzed to determine how it should be indexed. The text is lowercased and punctuations, stop words, special characters are removed. To effectively find the morphological variations of words, the words are stemmed.

Data about web pages are stored in an index database for use in later queries. A query can be a single word. The purpose of an index is to allow information to be found as quickly as possible.

When a user enters a query into a search engine (typically by using keywords), the engine examines its index and provides a listing of best-matching web pages according to its criteria. The engine looks for the words or phrases exactly as entered. Some search engines provide an advanced feature called proximity search which allows users to define the distance between keywords. There is also concept-based searching where the research involves using statistical analysis on pages containing the words or phrases being searched for. As well, natural language queries allow the user to type a question in the same form one would ask it to a human. This retrieval engine, however, applies the keyword searching. Document is retrieved if at least one of the query terms is present in that document.

The usefulness of a search engine depends on the relevance of the result set it gives back. While there may be large number of web pages that include a particular word or phrase, some pages may be more relevant, popular, or authoritative than others. Most search engines employ methods to rank the results to provide the "best" results first. How a search engine decides which pages are the best matches, and what order the results should be shown in, varies widely from one engine to another. In this retrieval engine, the terms in the document are weighted by looking the local and global importance. Only the highly ranked documents should be presented to the user by ranking in such a way that most relevant document comes first.

The nature of data published on the internet is mostly unformatted and it is very difficult for the search engine to select the desired contents. That's why even some prominent search engines such as Google, Yahoo, Bing still striving for the better performance of their engines. Before dealing with the shortcomings of the existing engines, it is very worthwhile to understand the basic working principles of the search engines and for this purpose retrieval engine. This prototype engine implements the vector space model that performs fairly well.

2 Approach

2.1 Breadth first crawling

Breadth-first crawling, wherein pages are crawled in the order they are discovered. A crawler that downloads pages in breadth-first search order discovers the highest quality pages during the early stages of the crawl [1]. As the crawling progresses, the quality of the downloaded page deteriorates. The breadth-first search is a good crawling strategy because the most important pages have many links to them from numerous hosts, and those links will be found early, regardless of on which host or page the crawl originates. Discovering high-quality pages early on in a crawl is desirable for search engines as none of the search engines is able to crawl and index more than a fraction of the web. Although breadth-first search crawling seems to be a very natural crawling strategy, not all of the crawlers we are familiar with employ it. Though this crawling strategy has bias towards high-quality pages, it is beneficial for the search engine.

2.2 Stemming

Stemming is a heuristic process that chops off the ends of words and often includes the removal of derivational affixes in the hope of achieving more similarity with the texts that contain the morphological variations of certain words. For example, *computer* and *computation* are different if exact matching is required but they are similar in meaning. Stemming helps catch morphologically varied words by changing to root word (here, *computer* and *computation* both turn to *comput*). Stemming also helps reduce the index size. Porter stemmer is widely used and implemented in different programming languages.

2.3 Inverted document indexing

An inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from terms (words or phrases), to its locations in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the collection. It is the most popular data structure used in document retrieval systems [2], used on a large scale for example in search engines. The typical inverted index contains the list of terms each pointing to the list of documents (or their ids) and the frequency of that term in that document. Indexing is central to the search engine as it allows fast retrieval of documents. A list of documents relevant to the query can be retrieved without scanning all the documents. Though there are optimized versions of inverted indexing, the type of inverted index works well for relatively small set of document collections.

```
{"important”,2} ->  {2,5} {8,100}

{"word",4} -> {15,1} {123, 3} {500, 2} {502,7}

{“information”,2} -> {777, 3} {1000,1}
```

Fig. 2: Inverted document index sample

2.4 The Vector space model

The vector space model treats the documents as sets of terms that can be individually weighted and manipulated, perform queries by comparing the representation of the query to the representation of each document in the space. Unlike Boolean model, it allows the retrieval of documents that partially match the query. If the vocabulary has *t* words, then the document and query are also expressed as vectors with *t* dimensions.

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$$

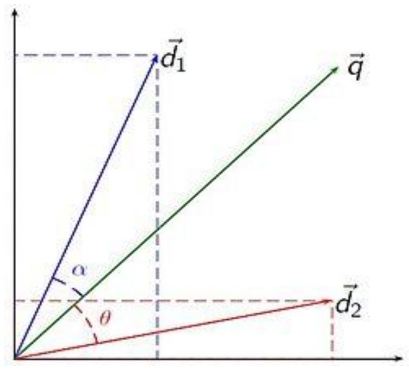

Fig. document vectors and query vector

2.5 Tf-idf term weighting

In statistical method, how to determine important words in a document and rank that document high when the query contains that words? How to determine the degree of importance of a term within a document and within the entire collection? These are the fundamental questions that need to be addressed by the retrieval engine. One widely used approach for term weighting in the document vectors is products of local and global parameters called tf-idf weighting. Tf – is the frequency (normalized) of word in the document and idf attenuates the effect of terms that occur too often in the collection to be meaningful for relevance determination.

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: total number of documents)

$w_{ij} = tf_{ij}\ idf_i = tf_{ij} \log_2 (N/ df_i)$

tf_{ij} - is the frequency of term *i* in the document *j*.

For query, the weight of the term is expressed as,

$w_{iq} = (0.5 + [0.5 * freq(i,q) / maxl(freq(l,q)]) * \log(N/n_i)$

2.6 Cosine similarity

To identify the similarity of the given query and document containing one or more query terms can be calculated using cosine. It is a length normalized method. The cosine similarity can be used to rank the documents.

$$\mathrm{sim}(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

3 Design

The functions of retrieval engine can be categorized as – Crawling, Preprocessing, Indexing, Retrieving, ranking, presenting the result.

3.1 Crawler

```
Q = {http://www.<a website>} (Queue having list of unique URLs to be fetched from web).
U = {} (list containing unique urls to html, pdf or text file found, that are in <user specified> domain).
C = 0 (crawled page count)
While C < 1500 do
    l = Q.dequeue()
    D = Get page from l
    Preprocess (D)
    If (D is pdf)
        ConvertToText (D)
    Save (D)
    L =  URLS linked to the current page
    Foreach l in L do
        If l is not in U AND l contains <user specified domain> AND l links to {.pdf|.txt|.html}
            Q.enque(l)
            U.add(l)
    end
    C = C + 1
```

3.2 Indexing and searching

For indexing, create the inverted document index. For searching, do not calculate cosine for all the documents, just enough if it is done for the documents having at least one query term.

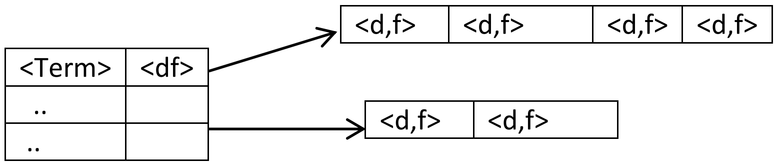

Fig 3: Inverted document index

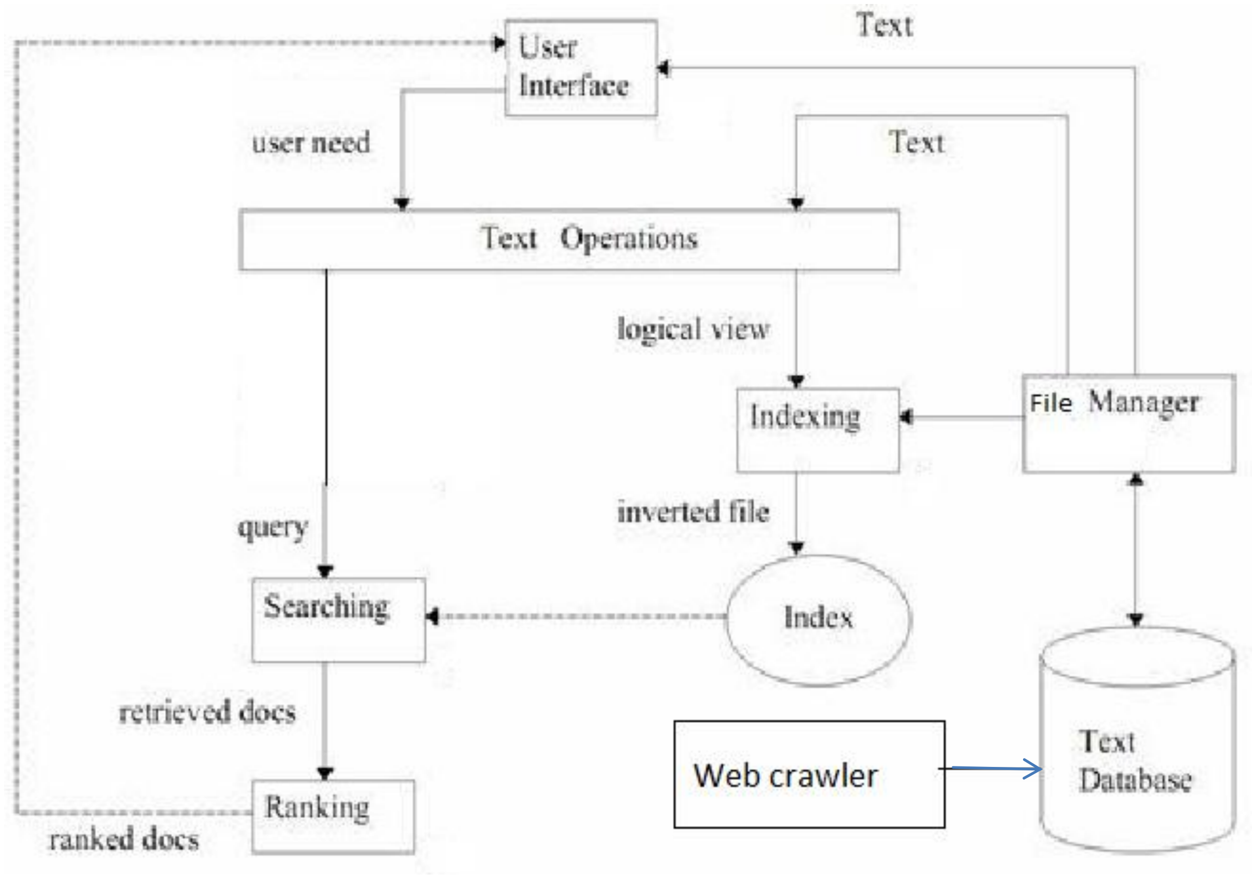

Fig 4: system architecture

3.3 User interface

Create a page that has at least two fields – query input, and search button. Set the CGI script as form handler.

In the form handler program, get the query input, do searching and ranking. Print the result in HTML format.

4 Implementation

The logic has been implemented in Perl program. Perl program is very compact and provides very elegant features for text processing.

The implementation of this retrieval engine can be viewed as following components,

4.1 Crawling

Breadth first crawler has been implemented. It downloads the first 1500 pages from the <user specified > domain, removes the html tags and saves them to disk. It only saves the .html, .txt, and .pdf files only.

In the case of pdf document, it first converts to text document. To convert pdf document to text file, the library from CPAN (CAM::PDF and CAM::PDF::PageText) has been used.

4.2 Preprocessing

In this step, for each downloaded page, removes digits, special characters, punctuations, stop words. And it perms stemming using porter stemmer. The preprocessed text looks like,

```
futur student current student alumni friend faculti staff parent visitor
academ admiss research librari athlet give resourc media room az list quick
link email bursar offic event calendar job ecoursewar um elearn rodp bookstor
tom employe servic campu map librari fedex institut transcript umdriv iam
polic servic contact current student featur link academ academ advis academ
```

4.3 Indexing

Inverted document index has been created using hash of hash. As shown in the following code snippet, for each token in the document, it checks if that word has seen earlier. If it is not there, it adds an entry in the main hash and then points to another document frequency table (hash).

Since the document collection is static, the indexed data is saved when it is generated for the first time. In the subsequent request, the index data is loaded from the file and it is comparatively faster.

4.4 Query processing

The query is preprocessed as it is done for the web pages and then query vector is created. Stop words and punctuations are removed, changed to lower cased and performed stemming.

4.5 Searching

In this phase, the candidate set is generated using vector space model. Cosine similarity is calculated for the documents containing at least one query term and they are retrieved efficiently using inverted document index.

4.6 Filtering and Ranking

From the candidate set of documents, the documents are ranked based on their cosine similarity with the query vector. Only limited number of documents (it can be configured) are displayed to the user.

Since calculating the recall is difficult as it is hard to find all the relevant documents in the collection, the number of relevant documents in the corpus has been predicted using some function.

Assuming that the documents having more query terms are more relevant, an exponentially decaying function has been used to predict the number of relevant documents in the corpus.

$P_r = \sum d_k x (1/(n^{(n/k - 1) *2}))$ where $1<=k<=n$, n = number of query terms, d_k – number of documents containing k query terms. The documents containing the all the query terms (i.e. k = n) has the decaying factor 1.

To improve the precision, only the 70% of the predicted number of relevant documents are retrieved (i.e. shown to the user).

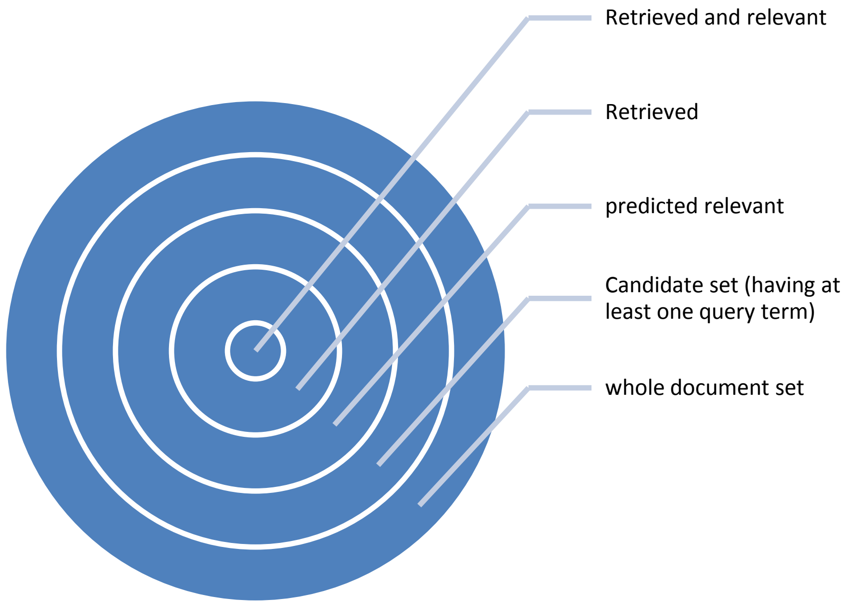

Fig 5: volume of documents in different level

4.7 User interface

The interface is a simple HTML page having query input field and search button. When Search button is clicked, the form is handled by the Perl script and prints the ranked documents title along with the ranking score (i.e. cosine similarity value), link to the web, and link to the local copy. The Perl script is deployed in the Apache server.

5 References:

1. Information retrieval course pages
   http://www.cs.memphis.edu/~vrus/teaching/ir-websearch/
2. Breadth first search: Breadth-First Search Crawling Yields High-Quality Pages
   http://www10.org/cdrom/papers/pdf/p208.pdf
3. Inverted indexing
   http://www.nist.gov/dads/HTML/invertedIndex.html
4. Introduction to Information Retrieval (Christopher Manning, http://nlp.stanford.edu/IR-book/)
5. Vector space model
   http://en.wikipedia.org/wiki/Vector_space_model