A Search Engine for memphis.edu domain

Muktadir Chowdhury
mrchwdhr@memphis.edu
Department of Computer Science
University of Memphis

December 4, 2016

1 Introduction

Modern world-wide-web contains too much information that it is almost impossible for a human being to search it manually for relevant information. According to W.H. Auden, The greatest problem of today is how to teach people to ignore the irrelevant, how to refuse to know things, before they are suffocated. For too many facts are as bad as none at all". We need an automated information retrieval system to and the relevant pages on internet based on some query written in natural language. This searching should be computationally inexpensive because we have too much diverse things know. To make this problem more challenging, worldwide web is growing at an exponential rate [1] and its contents are not static. We are in need of a search engine that can address three things. Firstly, it is capable to return relevant pages in web corpus. Secondly, searching should be fast. Thirdly, it is capable to order the search results based on the similarity with search query and most relevant pages is at the beginning of the results.

2 Approach

Two commonly used retrieval models are Boolean Model (also Extended Boolean model), and Vector Space Model. Both have some strong sides as well as some weak sides. We need to pick the most appropriate one.

2.1 Boolean Model

Boolean model [1] depends on set operation. Documents are represented as a set of keywords. User queries are expressed as Boolean expression of keywords connected by AND, OR, and NOT while brackets are used to indicate scope. For example a query is like below:

*(Text **OR** information) **AND** Retrieval **AND** (**NOT**) Theory*

A document containing `text information retrieval` or `text retrieval` will be a match. On the other hand texttinformation retrieval theory will not be present in search result because of having textttheory in document. Boolean Model's positive sides are, it is easy to understand for simple queries and for normal user queries it is easy to implement. On the other hand, it has lots of limitations:

- **Firstly**, it is very rigid where AND means all, OR means any. So it returns either too many search results or too few.
- **Secondly**, it is difficult to express complex user queries for peoples with limited knowledge about set operation.

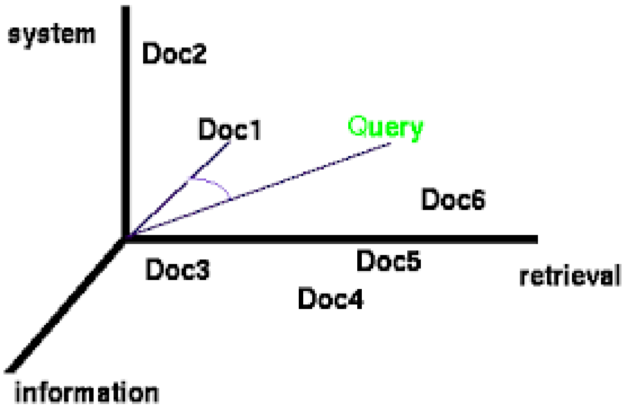

Figure 1: Vector Space Model

- **Thridly**, it is difficult to incorporate relevance feedback via query modification
- **Fourthly**, Basic Boolean Model does not support to order the search result according to the relevance score with search query. However, Boolean Model can be extended to support this ranking [3]. Given the limitation of Boolean Model we need to pick some model which can distinguish important words in a document for weighted comparison. We need a similarity measure between document and user query
- **Lastly**, we need a way to incorporate relevance feedback of user via query modification. This leads to Vector Space Model.

2.2 Vector Space Model

Vector Space Model [4] is also known as Term Vector Model. It is an algebraic model for representing text documents as vectors of terms where a weight is associated with each term dimension. User query on the other hand is also represented as a vector of terms. A similarity score like, Euclidian Distance, Manhattan Distance, Inner Product, or Cosine can be used to measure the similarity score between document vector and query vector. Figure 1 depicts an example of Vector Space Model. Strength of Vector Space Model are:

- It is simple and mathematically based model.
- It takes into account both local importance in a document and global importance in the whole corpus.
- It provides partial matching with query and ranked result.
- It works quite well practically.
- It is possible to have an efficient implementation of this algorithm to provide a fast ranking of relevant documents.

Despite all the strong factors it has some limitations too.

- It does to take into account the semantic meaning of words, or their occurrence orders.
- We lack the control of rigidly returning search results.

Despite these weaknesses there many strong factors of Vector Space Model and thats why we have selected it as our retrieval model.

3 Design

3.1 Weight of Vector

Weight of document vector is defined as a product of term frequency, tf, and inverted document frequency, idf. Term frequency is defined as frequency of term in a document. However, term frequency is normalized by dividing it with maximum term frequency in the same document. On the other hand, document frequency, df, is defined as number of documents a term is found. If df increases means that importance of the term is reduced. So it is logical to used N/df as idf, where N is the total number of documents in the corpus. However N/df is a big number as compared to tf. So, $log_{10}(N/df)$ is used as idf to dampen the effect relative to tf. So weight for a term i in document j is Equation (1).

$$\begin{aligned} w_{ij} &= tf_{ij} * idf_{ij} \\ &= tf_{ij} * log_{10}(N/df_i) \end{aligned} \tag{1}$$

However, for query we are calculating weight using Equation (2)

$$\begin{aligned} w_{iq} &= (0.5 + (0.5 * tf_{iq})) * log_{10}(N/df_i) \\ &= (0.5 + (0.5 * freq(i,q)/max_l(freq(l,q)))) * log_{10}(N/df_i) \end{aligned} \tag{2}$$

3.2 Similarity Measure

VSM utilizes similarity measures in order to rank the retrieved documents.

Euclidean Distance: Euclidean distance between document vector $d1$ and query vector $d2$ is the length of vector, $|d1 - d2|$ Equation (3).

$$\begin{aligned} EuclideanDistance(X,Y) &= |X - Y| \\ &= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \end{aligned} \tag{3}$$

It has the same limitation of Euclidean Distance, as it has lower limit of 0, but unlimited upper limit. So we are in need of a normalized version of score.

Inner Product: Inner Product similarity between vectors for the document d_i and query q can be computed as the vector inner product. For binary vectors, the inner product is the number of matched query terms in the document (size of intersection). For weighted term vectors, it is the sum of the products of the weights of the matched terms.

$$\begin{aligned} InnerProduct(d_j, q) &= d_j \cdot q \\ &= \sum_{i=1}^{n} |w_{ij} * w_{iq}| \end{aligned} \tag{4}$$

Cosine Similarity: Distance between document vector $d1$ and query vector $d2$ captured by the cosine of the angle x between them. So it measures the similarity, not the distance. It is bounded between 0 and 1. So this is a normalized version of similarity score. So, longer documents do not get more weight. From Figure 1 we get the similarity between document vector $D1$ and query vector Q is cosine of .

$$\begin{aligned} CosineSimilarity(d_j, d_k) &= \frac{d_j \cdot d_k}{|d_j||d_k|} \\ &= \frac{\sum_{i=1}^{n} w_{ij} * w_{ik}}{\sqrt{\sum_{i=1}^{n} w_{ij}^2}\sqrt{\sum_{i=1}^{n} w_{ik}^2}} \end{aligned} \tag{5}$$

3.3 Morphological Variation

Words in the query may be a morphological variant of words in documents. Stemming [2] is a method to normalize all morphological variations to a canonical form, namely the base form of the word, aka stem. For example, **index** is the base form for **indexed** and **indexing**. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. It reduces the number of distinct index terms and thus of the index. However, stemming is controversial from a performance point of view.

4 Implementation

4.1 Crawler

Crawler maintains two queues. Firstly, **queueUrls** contains all the URLs that need to be downloaded and parsed. Secondly, **visitedUrls** contains all the URLs that are visited and processed and has more than 50 tokens. If this array contains more than 10K unique URLs then crawlers job is done. **queueUrls** is initially seeded with two URL *http://www.cs.memphis.edu/ vrus/teaching/ir-websearch/* and *http://www.memphis.edu/*. At first crawler pop an URL from **queueUrls**. This is a priority pop, giving priority to memphis.edu the most. Every time crawler visits a page, it finds all the URLs that are available in anchors. If an URL is in relative path format crawler converts it to absolute URL. If this absolute URL is already available in visitedUrls means that specific URL is already visited it should be discarded. Otherwise the URL controller path ending with .php, .htm, .html, or does not contain any dot, crawler consider it as html content type and push the URL in queueUrls to process in next iteration. Http content type is not used because it can be misleading due to poor implementation in server side code. If the URL controller ends with .pdf it is considered as pdf file, and ending with .txt it is considered as text file.

4.2 Preprocessing

4.2.1 HTML File Processing

- Remove html comments.
- Remove JavaScript snippet.
- Strip html tags.
- Remove URLs from html file.
- Replace html special characters like, &nbsp;, &amp; etc., special characters (#, $), and punctuations with spaces

4.2.2 Text Processing

- Each text file is preporcessed following some steps mentioned below in order.
- Remove URLs
- Remove numbers
- Remove punctuations, e.g., comma (,), semicolon (;), period (.), question (?), exclamatory sign (!)
- Remove stop words: Stop word list is downloaded from http://www.cs.memphis.edu/ vrus/teaching/ir-websearch/papers/english.stopwords.txt and saved as stopwords.txt in the same directory of the script. Script needs this file to exclude all the words which is so frequent in the corpus that they do not contain significant information
- Convert each character to its lower case form.
- Remove special characters
- Apply stemming to find the morphological root form of the word. The Porter Stemming Algorithm is used.

4.3 Inverted Index

Index is stored as a hash of hash where *first level hash* is of **word** and *second level* is **document id**. Value of the hash is term frequency (tf) of corresponding document. Term frequency is normalized by dividing tf by maximum tf of the document. Hash is stored in a file and everytime query program is run, this index file is loaded as hash of hash. The hash table is stored as a binary hash file, this reduces the running time of the search engine, since access time of binary hash file is very less.

4.4 Query Pre-processing

Query is treated same as a document. Each query token is preprocessed as described for the documents. However, weight calculation is little bit different which is discussed in the Design section. A perl script is written to show the relevant information. CGI-API is used to interface between the webserver and perl. An interface takes search query user and submit the form using HTTP GET method. The API calls the perl script, captures the output, and display results (as shown in Figure 3) in the page in descending order according to its relevance, that is in the descending order of the cosine score. Figure 2 depicts the view of the search engine. Document id should be linked to its URL. We also need maximum term frequency of document to normalize term frequency.

5 Result

Precision, recall, and F-measure are used as evaluation matrics.

Precision, P (also called positive predictive value), is the fraction of retrieved instances that are relevant, while recall, R (also known as sensitivity), is the fraction of relevant instances that are retrieved. Both precision and recall are therefore based on an understanding and measure of relevance. F-Measure (F) is the harlmonic mean of recall and precision.

$$P = \frac{Number\ of\ Relevant\ document\ retrieved}{Total\ number\ of\ documents\ retrieved} \tag{6}$$

$$R = \frac{Number\ of\ Relevant\ document\ retrieved}{Total\ number\ of\ Relevant\ documents} \tag{7}$$

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}} \tag{8}$$

For obtaining a dependable recall, we have considered the google search for Memphis.edu as the gold standard for the total relevant documents retrieved. The table below describes the results obtained for each query. We have considered the top 50 (in descending order of cosine score) documents for evaluation. Relevant count column shows the total number of relevant documents retrieved by the search engine. The three metrics, precision, recall, f-measure are also shown for each of the 10 queries.

	Query	**Search Result**	**Relevant Count**	**Ground Truth (Google)**	**Precision**	**Recall**	**F-Measure**
1	Career Service Memphis	50	34	7500	0.68	0.005	0.01
2	Software Engineering Research	50	12	722	0.24	0.02	0.04
3	Cookie	16	15	65	0.94	0.23	0.37
4	President of the University	50	21	13100	0.42	0.002	0.004
5	Computer Science Research Awards	50	43	617	0.86	0.07	0.13
6	Semantic Similarity	50	25	92	0.5	0.27	0.35
7	Tiger Bike's Current Offer	50	15	71	0.3	0.21	0.25
8	Tiger Bioinformatics Internship	50	48	80	0.96	0.6	0.74
9	How to graduate with honors?	50	48	9130	0.96	0.005	0.01
10	Who teaches web search?	50	20	350	0.4	0.06	0.1

6 Future Work

- We need to increase the corpus size.
- Since the content of the webpages keep on changing, we need to update the corpus periodically.
- Inverted index file need to be built on real-time, in order to further reduce the running time.
- Feedback on query needs to be implemented
- We need to implement a better stemmer algorithm in order to improve the morphological variations of the vocabularies
- Semantic analysis of query needs to be developed.

References

[1] Frederick Wilfrid Lancaster and Emily Gallup. Information retrieval on-line. Technical report, 1973.

[2] Julie B Lovins. *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory Cambridge, 1968.

[3] Gerard Salton, Edward A Fox, and Harry Wu. Extended boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[4] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.