

**Retrieval Engine for The University of Memphis**

Information Retrieval Web Search

COMP- 7130

Author: Advisor:

Sree Lasya Vallabhaneni Dr. Vasile Rus

**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. The 2004 Pew Internet Survey Fallows 2004 found that “92% of Internet users say the Internet is a good place to

go for getting everyday information.” Information retrieval did not begin with the Web. In response to various challenges of providing information access, the field of information retrieval evolved to give principled approaches to searching various

forms of content. Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

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". 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 need 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.

**Approach**

For implementing the Search engine there are many kinds of models that can be chosen.

There is always a fight between choosing model. Similarly, there are two commonly used retrieval models Boolean Model and Vector Space Model. To choose one first a developer should be aware of advantages and disadvantages of both the models.

**Boolean Model:**

In Boolean Model documents are represented as a set of keywords. This model is based on set operation. When user enters a query, they are considered as keywords connected by AND, OR, NOT and to indicate the scope brackets are used. Set of words (indexing terms) present in

a document each term is either present (1) or absent (0)

*(Text* ***V*** *information)* ∧ *Retrieval* ∧*(****!****) Theory*

A document containing text information retrieval" or text retrieval" will be a match. On the other hand, information retrieval theory" will not be present in search result because of having theory" in document. One of the Advantages of Boolean Models are, for simple queries it is easy to understand and implement.

Disadvantages of Boolean Model is:

* 1. Firstly, it is very rigid where AND means all, OR means any. So, it returns either too many search results or too few.
  2. Secondly, it is difficult to express complex user queries for peoples with limited knowledge about set operation.
  3. Thirdly, it is difficult to incorporate relevance feedback via query modification.
  4. Fourthly, Basic Boolean Model does not support to order the search result per the relevance score with search query. However, Boolean Model can be extended to support this ranking [3].

Because of these limitations of Boolean Model other retrieval model needs to be considered for knowing important words and weights of the document for comparison. We need a similarity measure between document and user query. Also we need a way to incorporate relevance feedback of user via query modification.



**Vector Space Model (VSM)**

Vector Space Model 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. For similarity score we can use, Euclidian Distance, Manhattan Distance, Inner Product, or Cosine 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 [4] are

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

However, It has some limitations too.

* 1. It does to consider the semantic meaning of words, or their occurrence orders.
  2. We lack the control of rigidly returning search results.

Because of many advantages of the Vector space model Vector Space Model is selected for this project.



**Design**

**Raw File Processing**

Raw files like js, html, pdf, and txt files are processed and converted to a text file.

*HTML File Processing*

* Remove JavaScript snippet starting with “<script>” and ending with “</script>”
* Replace html special characters like, “&nbsp;”, “&amp;” etc., special characters (#, $), and punctuations with spaces
* Strip html tags enclosed with “<” and “>”
* Remove html comments starting with “<!-” and ending with “->”.
* Remove URLs from html file.

*Text Processing*

Each text file is preprocessed following some steps mentioned below in order.

* 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 numbers
* Remove special characters
* Apply stemming to find the morphological root form of the word. The Porter Stemming Algorithm is used

**Morphological Variation**

Words in the query may be a morphological variant of words in documents. Stemming [5] is a method to normalize all morphological variations to a canonical form, namely the base form of the word, aka stem. For example, **join** is the base form for **joining** and **joined**. 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.

**Inverted Index**

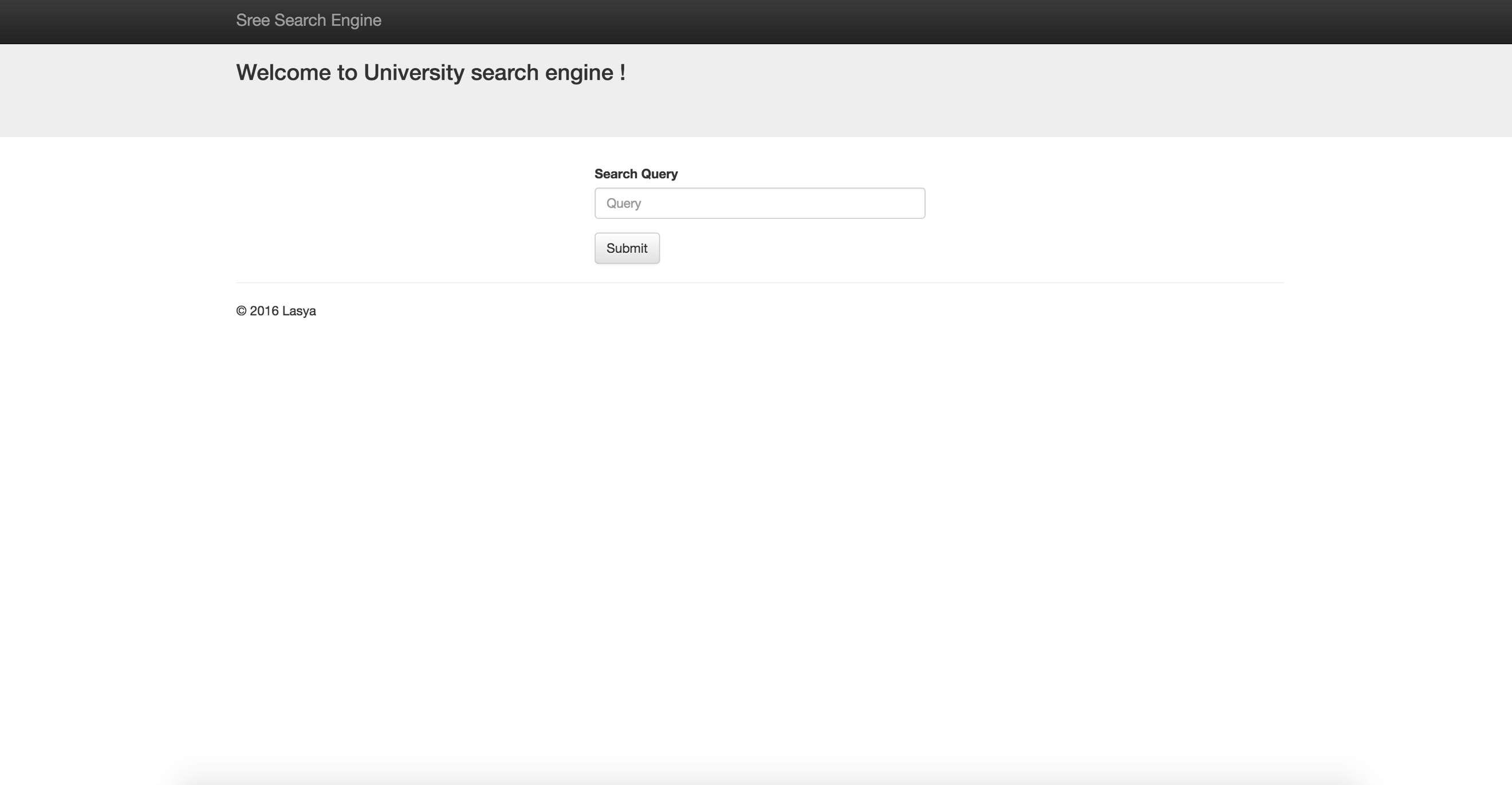
Index is stored as a dictionary of dictionary where *first level* dictionary is of **word** and *second level* is **document id**. *Value* of the dictionary key is **term frequency (tf)** of corresponding document. Term frequency can be normalized by dividing tf by maximum tf of the document. Dictionary is processed every-time query program is run, this index file is loaded as dictionary of dictionary. The dictionary is stored as a binary dictionary file, this reduces the running time of the search engine, since access time of binary dictionary file is very less.

**Crawler**

Crawler maintains two lists. **UrlCollection** 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. Crawler list size is kept only 10,000 not more than the limit with unique links. **UrlCollection** 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 **UrlCollection**. Every time crawler visits a page; it finds all the URLs that are available. 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 **UrlCollection** to process in next iteration. If the URL controller ends with *.pdf* it is considered as pdf file, and ending with *.txt* it is considered as text file. Http content type is not used because it can be misleading due to poor implementation in server side code.

**Query Pre-processing: Flask Framework**

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 python script is written to show the relevant information. Flask Framework is used to interface between the webserver and python. An interface takes search query user and submit the form using HTTP GET method. The API calls the python script and display results in other page where it routes to “search” page in descending order per its relevance, that is in the descending order of the cosine score. **Image** below 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.

****

**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.

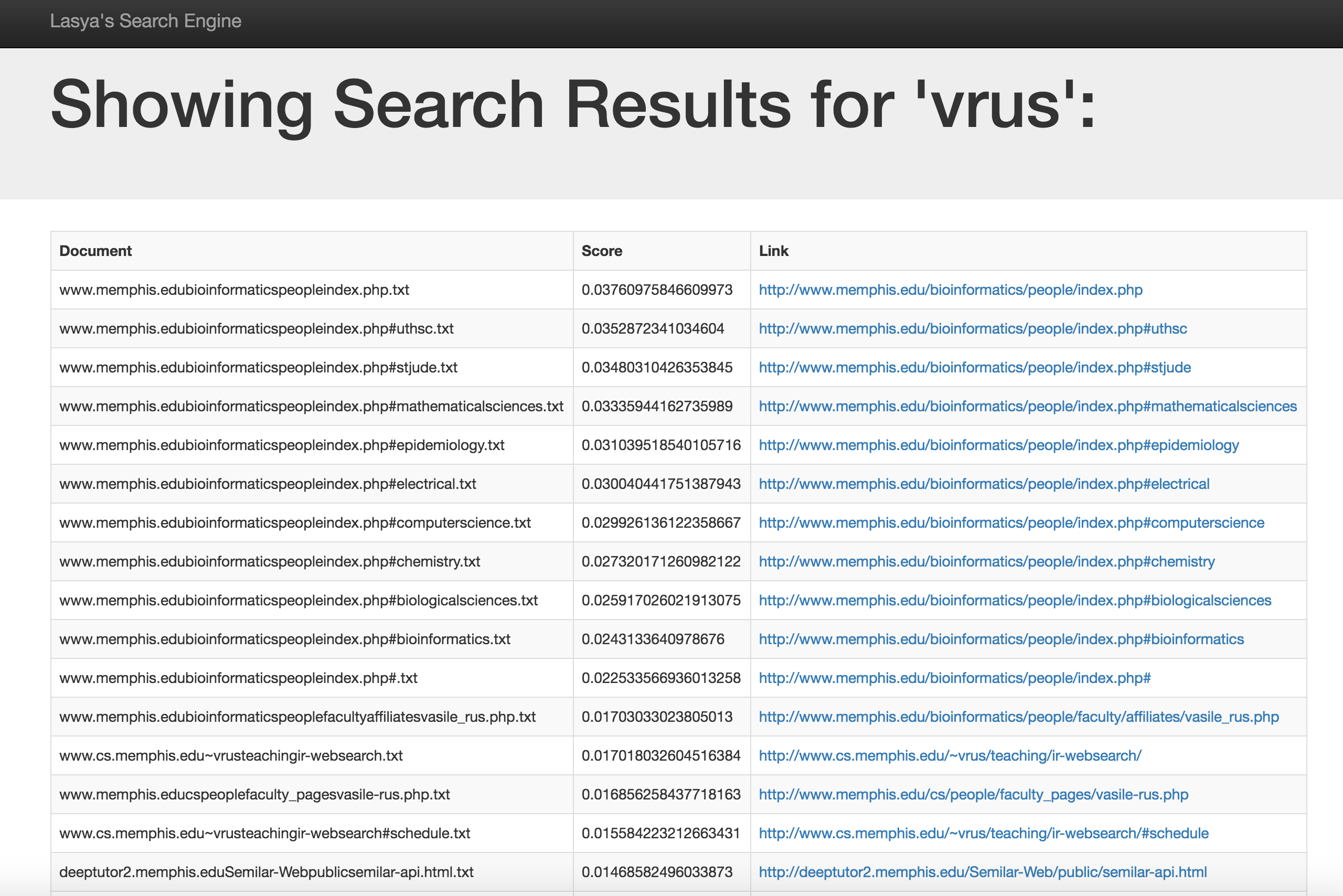
Document frequency (**df**) is number of documents a term is found. If **df** increases means that importance of the term is reduced. So, it is logical to used **N/df** as **idf**, where N is the total number of documents in the corpus. However, **N/df** is a big number as compared to **tf**. We can say that **df** is global variable. So, **log10(N/df)** is used as **idf** to dampen the effect relative to **tf**. So weight for a term **i** in document **j** is

*wij = tfij \* idfij = tfij \* log10(N/dfi)*

However, for query we are calculating weight as

*wiq = (0.5 + (0.5 \* tfiq)) \* log10(N/dfi)*

*= (0.5 + (0.5 \* freq(i, q)/maxl(freq(l, q)))) \* log10(N/dfi)*



**Similarity Measure**

Vector Space Model 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|,

*EuclideanDistance(X, Y) = | X – Y | =*

It has some limitations as it has lower limit of 0, but unlimited upper limit. So we are in need of a normalized version of score.

**Manhattan Distance:**

Manhattan Distance is also knows as *city block* measure. Idea behind it is that we cannot go in an angular way, we can only move in one director of an axis at a time.

*ManhattanDistance(X, Y) =*

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 **di** 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.

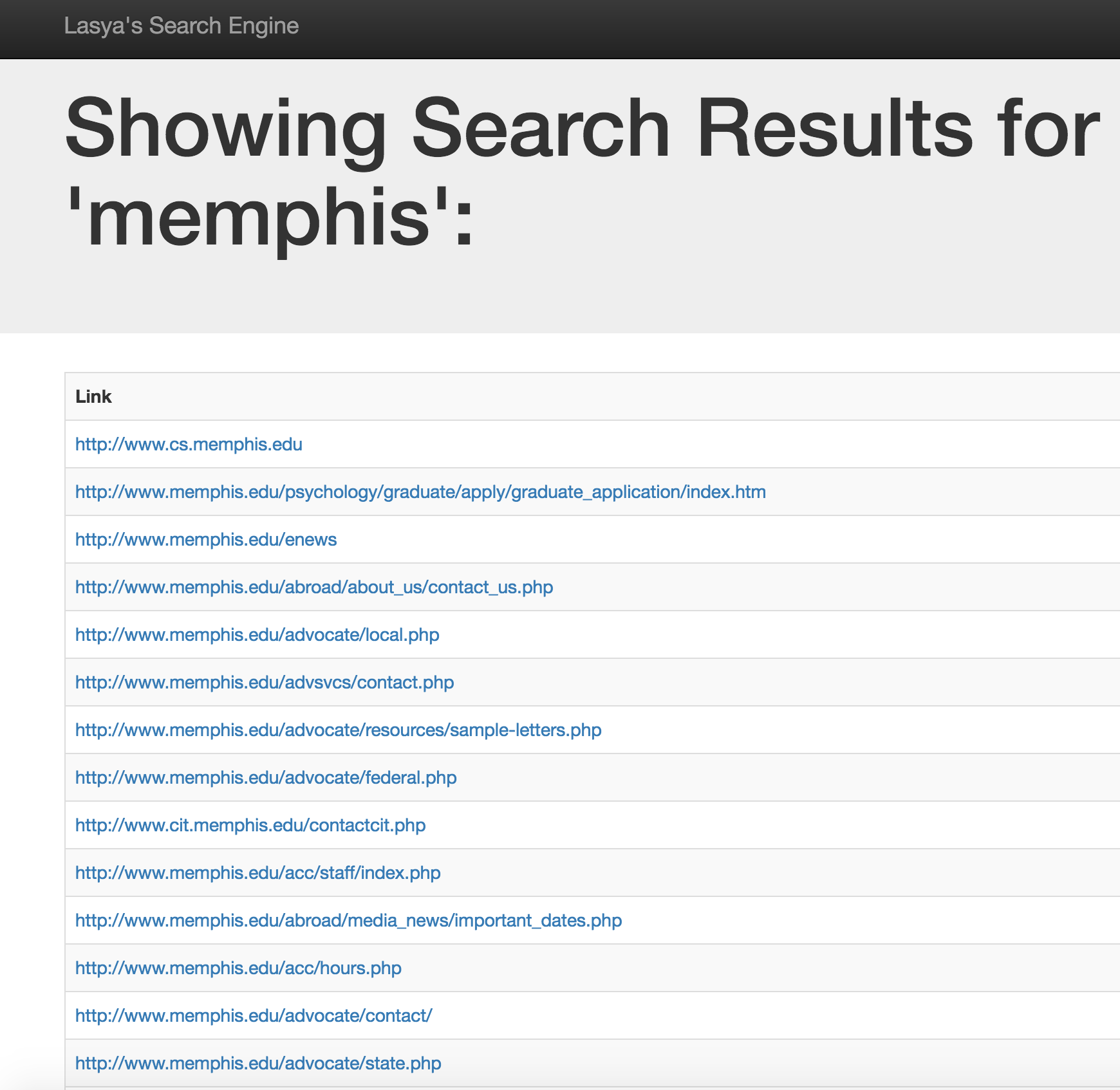
*InnerProduct(dj , q) = dj* ***.*** *q =*

It has the same limitation of Euclidean Distance, as it has lower limit of 0, but unbounded upper limit. It favors long documents with many unique terms. It measures how many terms matched but not how many terms are not matched. So, we are in need of a normalized version of score.

**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 *θ*.

*CosineSimilarity(dj , dk) = =*

****

**Results:**

For calculating Recall and Precision, as input file few sample files are considered and researched which documents are relevant to the query and how many are retrieved from the input files.

The result is for the sample queries specified in the Assignment.

**Evaluation Metric**

Three evaluation metrics have been used.

*Precision*

*P =*

*Recall*

*R =*

*F-Measure (Harmonic mean of recall and precision)*

*F =*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Query | Documents  Retrieved | Relevant Count | Precision | Recall | F-Measure |
| 1 | Career Service Memphis | *45* | *30* | *0.66* | *0.05* | *0.09* |
| 2 | software engineering research | *39* | *31* | *0.79* | *0.02* | *0.039* |
| 3 | Cookie | *32* | *12* | *0.53* | *0.23* | *0.32* |
| 4 | president of the university | *45* | *28* | *0.62* | *0.2* | *0.30* |
| 5 | computer science research awards | *50* | *41* | *0.82* | *0.2* | *0.32* |
| 6 | semantic similarity | *36* | *27* | *0.75* | *0.27* | *0.4* |
| 7 | tiger bike's current offer | *29* | *13* | *0.43* | *0.21* | *0.28* |
| 8 | tiger bioinformatics internship | *45* | *38* | *0.84* | *0.6* | *0.70* |
| 9 | How to graduate with honors? | *38* | *23* | *0.68* | *0.05* | *0.09* |
| 10 | Who teaches web search? | *44* | *31* | *0.70* | *0.06* | *0.11* |

**Future Work:**

* Semantic Analysis of the Query words can be implemented.
* Corpus can be increased to other websites not specific to domain.
* Better Ranking algorithm can be implemented for better top results.

**References:**

* Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze. Cambridge University Press © 2008 Cambridge University Press
* Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. Communications of the ACM, 18(11):613{620, 1975.