Digital Library Curriculum Development

Module 7-a: Indexing and Searching

(Draft, Last Updated, 10/16/09)

1. Module Name: Indexing and Searching

2. Scope

This module covers the general principles of information retrieval (IR) models, indexing methods, and the working of search engines applicable to digital libraries.

3. Learning Objectives

By the end of this lesson, students will be able to:

a. Describe the basic working of search engines, and explain the Indexing and Retrieval stages.
b. Explain how a crawler visits sites and collects documents for indexing.
c. Compare different indexing methods.
d. Compare the vector space model and other information retrieval models.

4. 5S Characteristics of the module

- Space: The concept of *space* for this module is physical space and virtual web space. The web pages, indexes and posting lists are stored in a physical space on servers, and the crawler crawls the virtual web space to find new content. Many IR models use spaces, e.g., vector or probability.

- Stream: The web crawler finds URL's in a webpage, which can be considered a stream of data, which it follows to find more streams recursively.

- Structure: The web crawler, inverted index and postings list all make use of particular data structures.

5. Level of effort required

a. Class time: 3 hours
b. Student time outside class: 6 hours
  - Reading before each class starts 4 hours
  - Homework assignment: 2 hours

6. **Relationships with other modules**

   3-c Harvesting: Harvesting applications in digital libraries crawl a set of archives/ document collections to find information specific to a particular topic. This is similar to search engines, discussed in this module; they use a crawler to collect information on the web/document collection.

   6-b Online information seeking behavior and search strategy: The 6-b module takes a user oriented view of the IR system and search strategies, while the *Indexing and Searching* module takes a more systems oriented view of the IR system, its structure and its functions.

   6-a Information needs, relevance: The *Indexing and Searching* module addresses the user's information needs by collecting and presenting hopefully relevant data based on the user's query (needs).

7. **Prerequisite knowledge required:** None

8. **Introductory remedial instruction:** None

9. **Body of Knowledge**

Topic 1: Search Engines

a. Overview of search engines in digital libraries

- The need for search engines in digital libraries was motivated and contextualized by reference to search engines that crawl the Web content, which includes incredible volume of academic content.
- It is estimated that about one billion individual documents are in the "visible" and nearly 550 billion documents on 200,000 web sites in the "deep" web, which cannot be crawled with standard methodology due to its dynamically generated content.
- For the research and teaching community the "invisible" web is of specific interest as it includes a major proportion of (high) quality content in free or licensed databases.

b. Shortcomings of commercial search engines for digital libraries.

- Commercial search engines cannot search the "invisible" part of the digital library.
- A search engine's primary business is to obtain revenue through advertising; therefore the contents of the digital library could be used to boost sales of certain products, which is profitable for the search engines.
- There is no guarantee of the long-term sustainability of an index; everything is left to the discretion of the search engine.

- Search engines aim at content that can be automatically indexed. It cannot consider manual conversion of data formats, which doesn't work well with digital libraries.

c. The way search engines work in digital libraries (DL) is illustrated in the following figure.

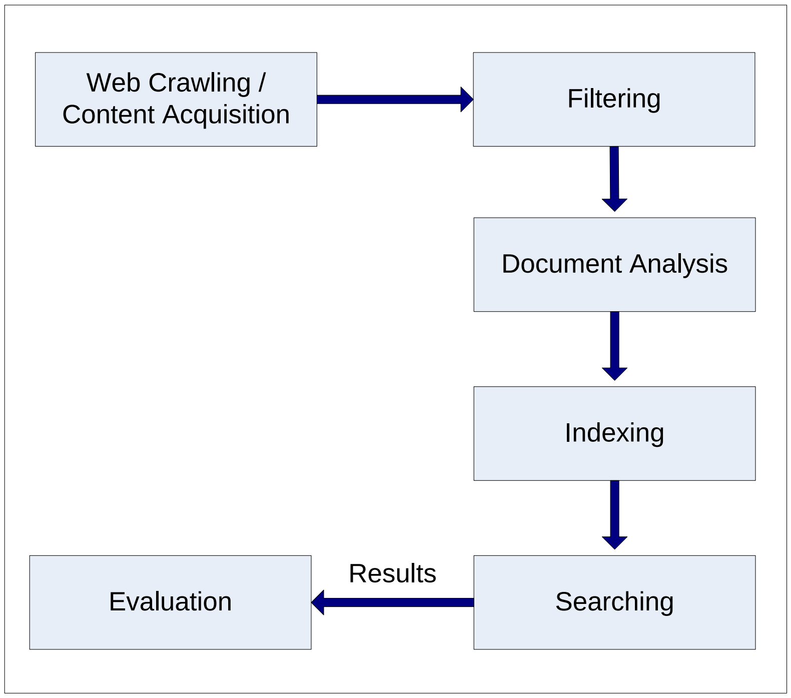


Figure 1. Content acquisition (Web crawling) in DL context

Topic 2: Web Crawling

a. A **Web crawler** (also known as a **Web spider)** is a program or automated script, which browses the World Wide Web or a specific part of the Web or private databases in a methodical and automated manner.
b. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches.
c. A crawler generally starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

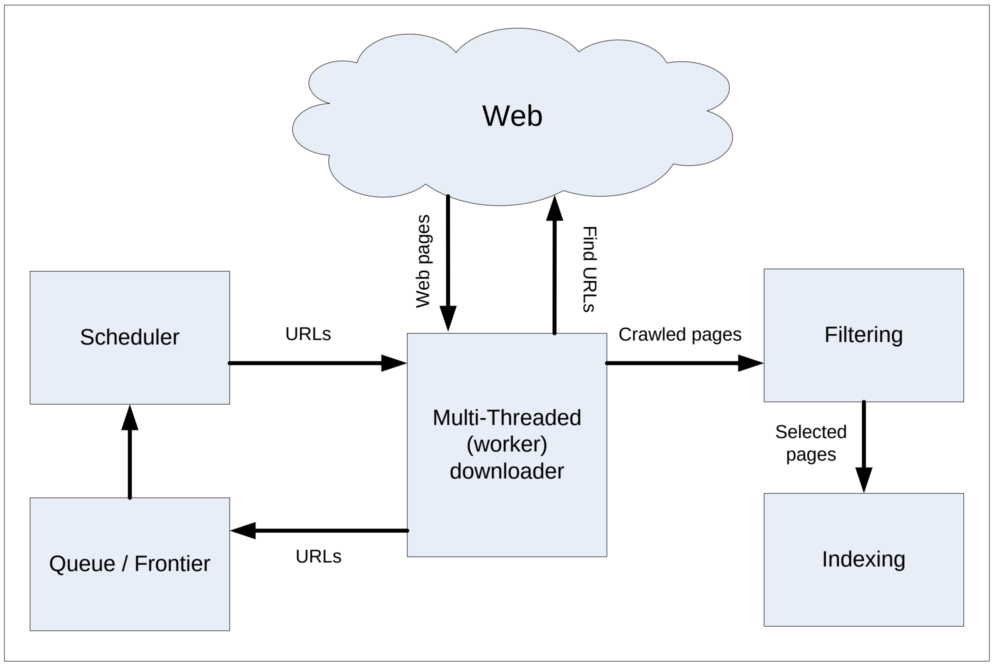


Figure 2. The working of Web crawler

d. Features a web crawler should provide include:

- **Distributed**: The crawler should have the ability to execute in a distributed fashion across multiple machines.

- **Scalable**: The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.

- **Performance and efficiency**: The crawl system should make efficient use of various system resources including processor, storage, and network bandwidth.

- **Quality**: Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biased toward fetching “useful” pages first.

- **Freshness**: In many applications, the crawler should operate in continuous mode: It should obtain fresh copies of previously fetched pages. A search engine crawler, for instance, can thus ensure that the search engine’s index contains a fairly current representation of each indexed web page. For such continuous crawling, a crawler should be able to crawl a page with a frequency that approximates the rate of change of that page.

- **Extensible:** Crawlers should be designed to be extensible in many ways to cope with new data formats, new fetch protocols, and so on. This demands that the crawler architecture be modular.

Topic 2: Document Analysis

Document Analysis consists of some of the following processes

a. Tokenization

- Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens.*
- For example “We shall rejoice” is broken down into 3 tokens ‘We’, ‘shall’ and ‘rejoice’.
- Tokenization is language and domain specific. For example, “San Francisco” could be tokenized into 2 tokens ‘San’ and ‘Francisco’, which should have been a single token ‘San Francisco’.
- The process of tokenization should not chop up words just by detecting white spaces. Words or phrases that could pose a problem could be phone numbers (800-444-5555), proper nouns, punctuation characters, apostrophes, etc.

b. Stop Words

- Some extremely common words which appear to be of little value in helping select documents matching a user’s need often are excluded from the vocabulary entirely. Such words are called ‘stop words’.
- Examples include a, an, and, are, be, it, by, has, from, for, were, will.
- This helps in dropping some tokens, and thus helps in reducing the size of the posting list (described in *Topic 3. Indexing*)
- Some search engines do not use stop word filtering or just limit the size of stop word lists (stop lists). This is because stop lists could be unfair to certain queries, e.g., ‘To be or not to be’

c. Stemming

- Documents tend to use different forms of the same word, e.g., differ, differed, differentiate, different.
- These could result in separate tokens, even though they stem from the same word ‘differ’.
- When a user queries for ‘different’ he might be interested in getting results that have other forms of the word ‘different’.
- Stemming reduces the size of posting lists and has been shown to improve document rankings. It enhances recall, but may reduce precision.

- More linguistically accurate processing, lemmatization, typically has similar effect, but usually is more expensive.

d. Normalization

- Normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens.
- The above mentioned processing should be done in such a way that consistent text handling occurs for queries as well as documents. This is especially important if documents can be queries.

Topic 3: Indexing

a. An index is used to quickly find terms in a document collection. As a Digital Library grows, an efficient method to do a full-text search is required. An inverted index is typically used to achieve this objective.
b. Text is preprocessed and all unique terms are identified. For each term, a list of document IDs containing the term also is stored. This list is referred to as a posting list.
c. Indexing requires additional overhead of I/O and space for the index to be stored on a secondary storage.
d. Indexing dramatically reduces the amount of I/O to satisfy an ad hoc query. Upon receiving a query, the index is consulted for each querying term, the corresponding posting list is referenced, and documents are ranked based on estimated query relevance.
e. Below describes building a simple inverted index as well as index compression and pruning methodologies:

- Building an inverted index (adapted from Grossman, D. A. and Frieder, O. (2004), Chapter 2.1 and 5.1.1)

  Create a simple inverted index with a posting list of the following documents D_1, D_2 and D_3:

  D_1: John sells oriental pots for a dollar.
  D_2: Oriental pots are made of clay.
  D_3: Kate buys cheaper and cheaper clay pots.

  Step1: Tokenization. Separate each word in the documents.

Term	DocId
john	1
sells	1
oriental	1
pots	1
for	1

a	1
dollar	1
oriental	2
pots	2
are	2
made	2
of	2
clay	2
kate	3
buys	3
cheaper	3
and	3
cheaper	3
clay	3
pots	3

Step 2: Remove stop words.

Stop words are removed by comparing the tokens against a list of predefined stop words. In this case, four stop words – ‘for’, ‘a’, ‘are’, ‘of’ and ‘and’ – are removed from the table above.

Term	**DocId**
john	1
sells	1
oriental	1
pots	1
dollar	1
oriental	2
pots	2
made	2
clay	2
kate	3
buys	3
cheaper	3
cheaper	3
clay	3
pots	3

Step 3: Stemming. Stem the tokens to their roots. In this case, ‘sells’ became ‘sell’, ‘buys’ became ‘buy’, ‘pots’ became ‘pot’, ‘made’ became ‘make’ and ‘cheaper’ became ‘cheap’.

Term	DocId
john	1
sell	1
oriental	1
pot	1
dollar	1
oriental	2
pot	2
make	2
clay	2
kate	3
buy	3
cheap	3
cheap	3
clay	3
pot	3

Step 4: Remove duplicate terms, keep the document IDs of the removed. Now an alphabetically-ordered simple inverted index with a posting list has been created.

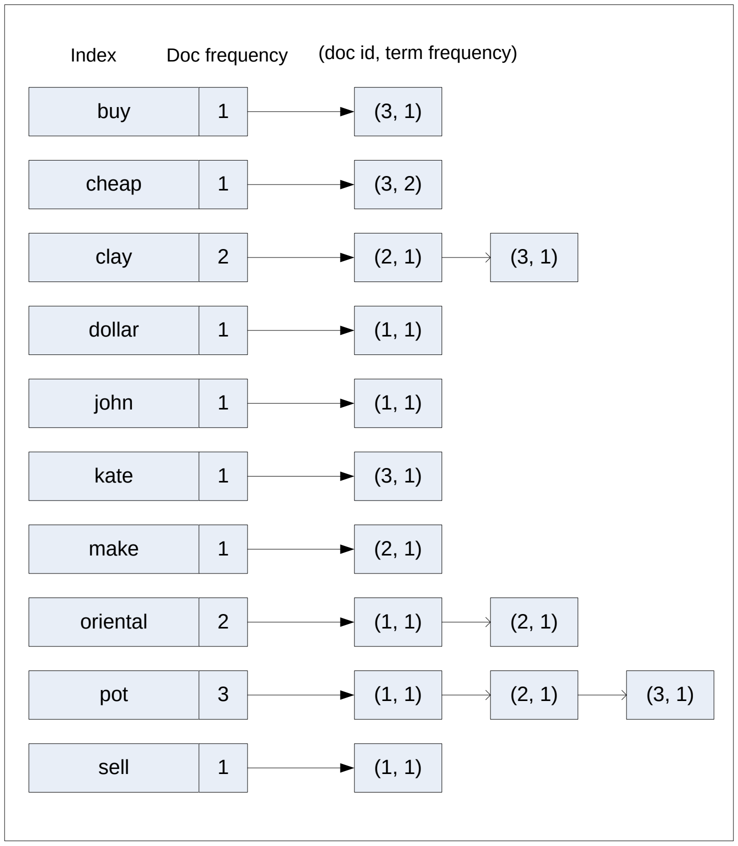


Figure 3. An example of an inverted index (boxes on the left) with a posting list (boxes on the right)

*** Once an inverted index (with a posting list) has been built, relevance ranking of documents can be efficiently computed given a query. This example continues in the Vector Space Model section of *Topic 4.***

- Compressing an inverted index
    - Large indexes and posting lists can pose I/O bandwidth and storage problems.
    - Compression can help significantly, e.g., see the 'Fixed Length Index Compression' algorithm.

- Variable Length Index Compression
    - Differences between consecutive entries in a posting list are used to compress the index.
    - Frequency distribution and offsets are used for compression.
    - See: 'Gamma Codes' and 'Varying Compression Based on Posting List size'

- Index Pruning
    - A 'lossy' approach in which certain posting list entries could be removed or pruned without significantly degrading performance.

- Reordering documents prior to indexing
    - Index compression efficiency can be improved by algorithms that reorder documents before inverted index compression.
    - Two types of algorithms can be used: Top-Down and Bottom-Up.

Topic 4: Searching

a. Information retrieval includes searching for information within documents. It typically uses an index, and retrieval strategies to achieve this objective.
b. A digital library can employ different retrieval strategies depending upon the type of documents stored in it.
c. For a given retrieval strategy, many different utilities can be employed to improve the results of the retrieval strategy. Retrieval utilities should be viewed as Plug-n-Play utilities which can be coupled with any retrieval strategy.
d. Below are a list of (selected) retrieval strategies:

Vector Space Model:

- It is based on the idea that a meaning of a document is conveyed by the words used in it. If one can represent the words in a document or a query by a vector, it is possible to compare documents with queries to determine how similar their content is.
- The model involves constructing a vector that represents the terms in the document and another vector that represents the terms in the query.

- The angle between the two vectors is computed by using the inner product of the two vectors. Each dimension of a vector has a weighting factor. The angle calculated is also referred to as a Similarity Coefficient (SC).
- Some of the terminologies used in the Vector Space Model are:

    - d = total number of documents in the collection
    - t = number of distinct terms in the document collection
    - tf_{ij} = number of occurrences of term t_j in the document D_i. This is referred to as *term frequency*.
    - df_j = number of documents that contain t_j. This is the *document frequency*
    - $idf_j = \log(d/df_j)$ where d is the total number of documents. This is the *inverse document frequency*.
    - $D_{ij} = tf_{ij} * idf_j$:weighting factor for a term j in a document i
    - Similarity Coefficient between query Q and document i:

      SC (Q, D_i) = $\sum_{j=1}^{t} w_{qj} \times d_{ij}$, where (w_{q1}, w_{q2}, …, w_{qt}) is a query vector and (d_{i1},d_{i2}, …, d_{it}) is a document vector of document i.

*** Continued from an example in *Topic 3. e.*** (adapted from Grossman, D. A. and Frieder, O. (2004), Chapter 2.1)

In the previous example of building a simple inverted index with a posting list in Topic 3. e, we had the following case insensitive documents D_1, D_2 and D_3:

D_1: John sells oriental pots for a dollar.
D_2: Oriental pots are made of clay.
D_3: Kate buys cheaper and cheaper clay pots.

Now a query Q is entered by a user:
Q: Cheap oriental clay pot.

There are three documents in the collection, so d = 3. If a term appears only in one of the documents, idf = log (d / df_j) = log (3/1) = 0.477. If a term appears in two of three documents, idf = log (3/2) = 0.176. For the terms that appear in all three documents, idf = log (3/3) = 0. In this way, low-frequency terms in the collection will have higher weight. This value is multiplied by the term frequency for overall weight. Therefore, documents that have high frequency of unique terms will be ranked higher.

Since there are 11 index terms in Figure 3, document vectors as well as a query vector of 11-dimension can be constructed by filling in the weighting factor, $D_{ij} = tf_{ij} * idf_j$ (Use Figure 3). Make sure that the query also goes through the steps of tokenizing, stop word removal and stemming.

Doc id	**buy**	**cheap**	**clay**	**dollar**	**john**	**kate**	**make**	**oriental**	**pot**	**sell**
D1	0	0	0	0.477	0.477	0	0	0.176	0	0.477
D2	0	0	0.176	0	0	0	0.477	0.176	0	0
D3	0.477	2*0.477	0.176	0	00	0.477	0	0	0	0
Q	0	0.477	0.176	0	0	0	0	0.176	0	0

$$SC(Q, D_1) = \sum_{j=1}^{t} w_{qj} \times d_{ij} = 0*0 + 0*0 + 0.176*0 + 0*0.477 + 0*0.477$$
$$+ 0*0 + 0*0 + 0.176*0.176 + 0*0 + 0*0.477$$
$$= (0.176)^2$$
$$\approx 0.031$$

$$SC(Q, D_2) = 2 * (0.176)^2$$
$$\approx 0.062$$

$$SC(Q, D_3) = 2 * (0.477)^2 + (0.176)^2$$
$$\approx 2 * 0.228 + 0.031$$
$$\approx 0.487$$

Therefore, the document ranking would be $D_{3,}$ $D_{2,}$ $D_{1.}$

Probabilistic Retrieval

- A probability based on the likelihood that a term will appear in a relevant document is computed for each term in the collection.
- For terms that match between a query and a document, the similarity measure is computed as the combination of the probabilities of each of the matching terms. Sometimes other probabilities are considered, too.

Neural Networks

- A sequence of ‘neurons’ or nodes in a network, that fire when activated by a query triggering links to documents.
- The strength of each link in the network is transmitted to the document and collected to form a similarity coefficient between the query and the document.
- Networks are trained by adjusting the weights on links in response to predetermined relevant and non-relevant documents.

Genetic Algorithms

- An optimal query to find relevant documents can be generated by evolution.
- An initial query is used with either random or estimated term weights. New queries are generated by modifying the weights in queries from the prior generation.

- A new query survives by being close to known relevant documents, while queries with less ‘fitness’ are removed from subsequent generations.

Boolean Algorithm

- A classical retrieval model, which is mostly adapted and used commercially.
- It is based on Boolean logic and classical set theory. Documents and query are represented as sets of terms. Retrieval is based on whether the documents contain the query terms or not.
- In mathematical point of view, the algorithm is straightforward, but there are problems to be solved in practical point of view, e.g., stemming, choice of terms, etc.

Extended Boolean Algorithm

- In conventional Boolean retrieval, relevance ranking to the query could not be estimated because the documents either satisfied or not satisfied the Boolean request.
- To incorporate relevance ranking, Extended Boolean was proposed.
- The idea is to assign term weights (ranges from 0 to 1) to the terms both in the query and the documents and incorporate those weights in the relevance ranking.
- The similarity coefficient (SC) is calculated using the Euclidean distance from the origin to the term weights and normalized.

Fuzzy Set Retrieval

- A document is mapped to a fuzzy set (a set that contains not only the elements but a number associated with each element that indicates the strength of membership).
- Boolean queries are mapped into fuzzy set intersection, union and complement operations that result in a strength of membership associated with each document that is estimated to be relevant to the query. This strength is used as a Similarity Coefficient.

e. Below are a list of (selected) performance metrics

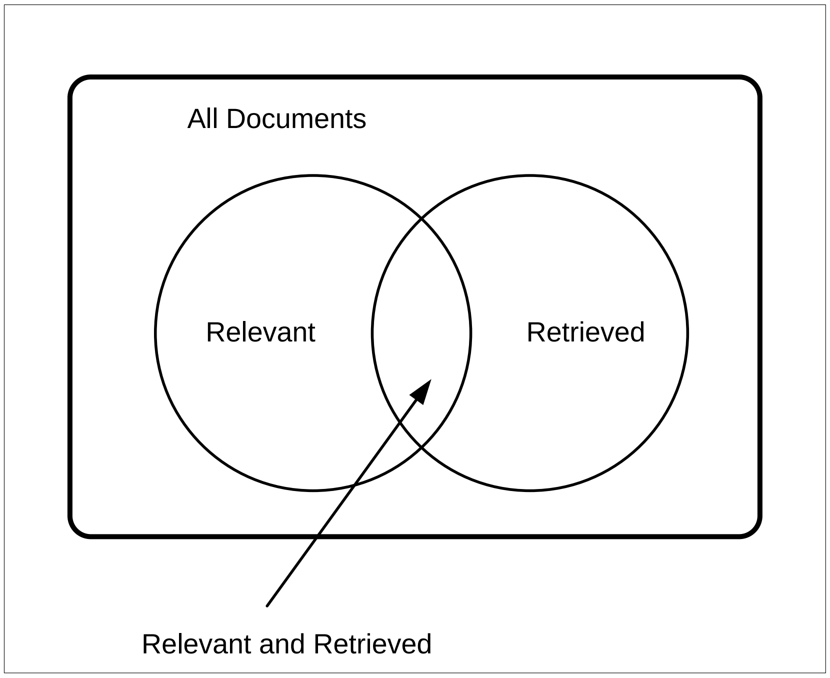


Figure 4. Venn diagram of relevant documents, retrieved documents and relevant and retrieved documents in the document collection

Precision: the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Precision provides one indication of the quality of the answer set, reflecting ability to avoid noise.

$$\textbf{Precision} = \frac{|\ \textbf{Relevant and Retrieved}\ |}{|\ \textbf{Retrieved}\ |}$$

Recall: the ratio of the number of relevant documents retrieved to the total number of documents in the collection that are believed to be relevant. This indication of quality reflects how comprehensive the query is.

$$\textbf{Recall} = \frac{|\ \textbf{Relevant and Retrieved}\ |}{|\ \textbf{Relevant}\ |}$$

***Fall-out:** the ratio of non-relevant documents that are retrieved to the all non-relevant documents in the collection. Fall-out of 0% can be achieved trivially by returning zero documents to any query.

$$\text{FallOut} = \frac{|\text{ Non-Relevant Retrieved }|}{|\text{ Non-Relevant }|}$$

***F-measure:** recall and precision are combined into a single measure. It was derived by van Rijsbergen in 1979. Depending on the attachment of importance to recall as precision, different β is assigned to F_β. F_1 is called a balanced F-score.

$$F_\beta = \frac{(1 + \beta^2)\ (\text{Precision x Recall})}{(\text{Precision + Recall})}$$

$$F_1 = \frac{2\ (\text{Precision x Recall})}{(\text{Precision + Recall})}$$

***Average precision of precision and recall:** it is to measure the relevancy of documents returned earlier in the result set.

$$\text{AveP} = \frac{\sum_{r=1}^{N} (P(r) \text{ x } rel(r))}{\text{Relevant}}$$

Where r is the rank, N is the number of documents retrieved, P(r) is the precision on the given cut-off rank, rel (r) is a binary function of relevance of a given rank

* From http://en.wikipedia.org/wiki/Information_retrieval

10. Resources

Required reading for students

Search Engine Technology and Digital Libraries, *"Libraries need to discover the academic content"*, Norbert Lossau, D-Lib Magazine, June 2004, Volume 10 Number 6 http://www.dlib.org/dlib/june04/lossau/06lossau.html

Grossman, D. A. and Frieder, O. (2004). Chapter 2: Retrieval Strategies. *In Information Retrieval: Algorithms and Heuristics*, Second Edition, Springer. pg 9-20

Grossman, D. A. and Frieder, O. (2004). Chapter 5: Efficiency, Inverted Index. In *Information Retrieval: Algorithms and Heuristics*, Second Edition, Springer. pg 182-195

Recommended reading for students

Manning C. D. et al. (2008). *Chapter 2. The term vocabulary and posting list*. In Introduction to Information Retrieval. Cambridge University Press. Online version retrieved in Sep. 14, 2009 from http://nlp.stanford.edu/IR-book/pdf/02voc.pdf

Grossman, D. A. and Frieder, O. (2004). Chapter 1. Introduction. In *Information Retrieval: Algorithms and Heuristics*, Second Edition, Springer. pg 3-5.

Index Construction, Chapter 4, Introduction to Information Retrieval, Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze Cambridge University Press. 2008 http://nlp.stanford.edu/IR-book/pdf/chapter04-construction.pdf

Manning C. D. et al. (2008). *Chapter 4.Index Construction*. In Introduction to Information Retrieval. Cambridge University Press. Online version retrieved in Sep. 14, 2009 from http://nlp.stanford.edu/IR-book/pdf/04const.pdf

Reading for instructors:

Lawrence, S. et al. (1999). *Indexing and Retrieval of Scientific Literature.* Proceedings of the eighth international conference on Information and knowledge management, Kansas City, Missouri, United States, Pages: 139 – 146. http://portal.acm.org/citation.cfm?id=319970&coll=GUIDE&dl=GUIDE&CFID=20419353&CFTOKEN=20765028&ret=1#Fulltext

Summann, F. (2004). Search Engine Technology and Digital Libraries, *"Moving from Theory to Practice"*. D-Lib Magazine, September 2004, Volume 10 Number 9. Article at http://www.dlib.org/dlib/september04/lossau/09lossau.html

DeRidder, J. (2008) *Googlizing a Digital Library*, The Code4Lib Journal, Issue 2. http://journal.code4lib.org/articles/43

Pant, G., Tsioutsiouliklis, K., Johnson, J., and Giles, C. L. 2004. Panorama: extending digital libraries with topical crawlers. In *Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries* (Tuscon, AZ, USA, June 07 - 11, 2004). JCDL '04. ACM, New York, NY, 142-150. DOI= http://doi.acm.org/10.1145/996350.996384

Web Crawler, Wikipedia. http://en.wikipedia.org/wiki/Web_crawler

IR-related resources from Wikipedia:

- Annotation http://en.wikipedia.org/wiki/Annotation
- Bayesian network http://en.wikipedia.org/wiki/Bayesian_network
- Binary classification http://en.wikipedia.org/wiki/Binary_classification
- Digital library http://en.wikipedia.org/wiki/Digital_library
- Content-based image retrieval http://en.wikipedia.org/wiki/Content-based_image_retrieval
- Edit distance http://en.wikipedia.org/wiki/Edit_distance
- Federated search http://en.wikipedia.org/wiki/Federated_search
- Hubs and authorities http://en.wikipedia.org/wiki/Hubs_and_authorities
- Human–computer information retrieval http://en.wikipedia.org/wiki/Human%E2%80%93computer_information_retrieval
- Hypertext http://en.wikipedia.org/wiki/Hypertext
- Image retrieval http://en.wikipedia.org/wiki/Image_retrieval
- Information http://en.wikipedia.org/wiki/Information
- Information filtering system http://en.wikipedia.org/wiki/Information_filtering
- Information retrieval http://en.wikipedia.org/wiki/Information_retrieval
- Language model http://en.wikipedia.org/wiki/Language_model
- Latent semantic analysis http://en.wikipedia.org/wiki/Latent_semantic_analysis
- Latent semantic indexing http://en.wikipedia.org/wiki/Latent_semantic_indexing
- Machine learning http://en.wikipedia.org/wiki/Machine_learning
- Metadata http://en.wikipedia.org/wiki/Metadata_%28computing%29
- Metasearch engine http://en.wikipedia.org/wiki/Metasearch
- N-gram http://en.wikipedia.org/wiki/N-gram
- Naive Bayes classifier http://en.wikipedia.org/wiki/Naive_bayes
- Okapi BM25 http://en.wikipedia.org/wiki/Probabilistic_relevance_model_%28BM25%29
- Open Archives Initiative http://en.wikipedia.org/wiki/Open_Archives_Initiative
- Precision and recall http://en.wikipedia.org/wiki/Precision_%28information_retrieval%29
- Query expansion http://en.wikipedia.org/wiki/Query_expansion
- Relevance feedback http://en.wikipedia.org/wiki/Relevance_feedback
- Skip list http://en.wikipedia.org/wiki/Skip_list
- Standard Boolean model http://en.wikipedia.org/wiki/Standard_Boolean_model
- Stemming http://en.wikipedia.org/wiki/Stemming
- Term Discrimination http://en.wikipedia.org/wiki/Term_Discrimination
- Tf–idf http://en.wikipedia.org/wiki/Tf-idf
- Vector space model http://en.wikipedia.org/wiki/Vector_space_model
- Web crawler http://en.wikipedia.org/wiki/Web_crawler
- Zipf's law http://en.wikipedia.org/wiki/Zipf%27s_law

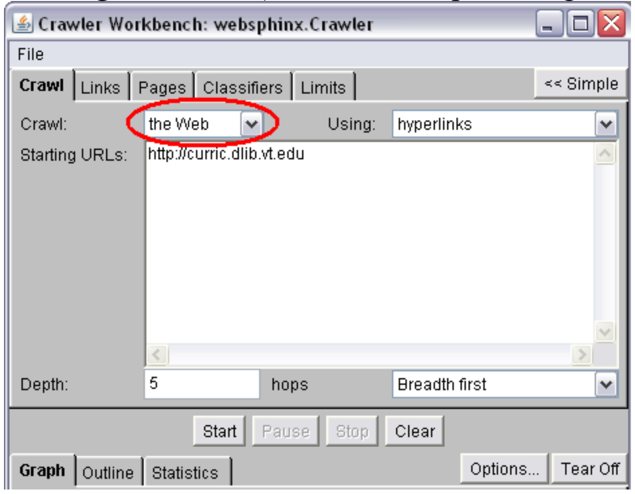
11. Concept map (Created by students)

12. Exercises/ Learning activity

Exercise A: Crawling

In this exercise, we will use a Web crawler to visualize depth-first and breadth-first crawling describing the group formation of the documents. In addition, we will describe the difference between crawling the subtree and the Web given a starting URL.

- Download the crawler application *Web Sphinx* from http://www.cs.cmu.edu/~rcm/websphinx/#download. Download the websphinx.jar file and save it to your computer in a directory. Use the command prompt, navigate to the directory where the jar file has been downloaded, and type "java –jar websphinx.jar". Note: You should have the latest version of the Java Runtime Environment (JRE) downloaded and installed on your computer.
- Once the application has been started, read the user manual at http://www.cs.cmu.edu/~rcm/websphinx/#examples to get acquainted with the tool.
- Use Depth-first/Breadth-first crawling, limit number of hops, crawl the sub-tree and describe changes in the visualization. Describe how certain groups are formed and certain pages try to stick together as they are under the same sub-root. Describe the outline (by clicking 'outline' tab next to 'Graph') as to how the pages are arranged in the server.
    - Certain URLs you can try this tool on are:
        - http://curric.dlib.vt.edu
        - http://www.cs.vt.edu
        - http://www.dlib.vt.edu
    - Compare the results following the URLs above
- Change the crawl setting to 'the Web' so that Websphinx could visit sites, which are located outside of the URL provided, by clicking the drop-down arrow and selecting 'the Web' (see the red ellipse in Figure 7).



Crawl: the Web
Using: hyperlinks
Starting URLs: http://curric.dlib.vt.edu
Depth: 5 hops Breadth first

Figure 5. Changing the setting to crawl the Web in Websphinx

- Click 'Start' button to see the documents form groups as they are crawled (you can stop crawling before the formation of groups become too complicated). Mouse-over the central document in each group. Describe the difference between crawling the subtree and crawling the Web. Do you find URLs not based on the starting URLs? What are they?

  * Suggestion to the instructor: different crawlers with different seed URLs can be used in this exercise.

Exercise B: Building a simple index and posting list, then calculating the relevance ranking using the Vector Space Model approach.

Students create a simple inverted index with a posting list as described in *Topic 3.e*, then calculates the document relevance ranking using the Vector Space Model approach, which is explained in *Topic 4*.

Each student develops $D_{1,}$ $D_{2,}$ D_3 as well as Q:

D_1:
D_2:
$D_{3:}$
Q:

Step1: Tokenization. Separate each word in documents.

Term	DocId

Step 2: Remove stop words.

Stop words are removed by comparing the tokens against a list of predefined stop words. Fill in the table below (add more rows as needed):

Term	DocId

Step 3: Stemming. Stem the tokens to their roots. Fill in the table below (add more rows as needed):

Term	**DocId**

Step 4: Remove duplicate terms, keep the document IDs of the removed.

(Create a figure of inverted index and a posting list)

Step 5: Querying. Using the query, find relevance ranking of the documents based on the similarity coefficients (SC) explained in *Topic 4*. Vector Space Model. Make sure that the query also goes through the tokenization, stop word removal and stemming procedure. Show the constructed document and query vectors and details of SC calculation.

Exercise C: Evaluation

Students calculate two performance metrics - precision and recall – to evaluate an IR system as described in *Topic 4. e.*

There are a total of 1000 documents in a document collection. For a given search query, the total number of relevant documents for the query are 400. The query is executed on the IR system, and it retrieves 600 documents. Out of the 600 documents, only 300 are relevant to the query.

Q 1. What is the precision and recall of the IR system?
Q 2. What is the Fall-out and F-measure values with $\beta = 2$?

13. Evaluation of Learning Activity

14. Glossary

15. Additional useful links:

Web Sphinx : http://www.cs.cmu.edu/~rcm/websphinx/

16. Contributors:

a. Initial author: Aniket Arondekar

b. Guidance: Dr. Edward A. Fox and Seungwon Yang

c. Evaluation: DL Curriculum project team