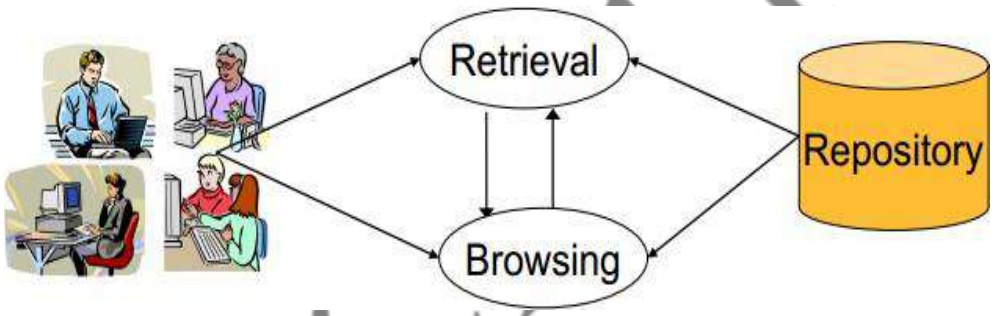
UNIT I – INTRODUCTION

Information Retrieval – Early Developments – The IR Problem – The Users Task – Information versus Data Retrieval – The IR System – The Software Architecture of the IR System – The Retrieval and Ranking Processes – The Web – The e-Publishing Era – How the web changed Search – Practical Issues on the Web – How People Search – Search Interfaces Today – Visualization in Search Interfaces.

1.1 INTRODUCTION:

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).

- Information retrieval (IR) is concerned with representing, searching, and manipulating large collections of electronic text and other human-language data.
- Web search engines — Google, Bing, and others — are by far the most popular and heavily used IR services, providing access to up-to-date technical information, locating people and organizations, summarizing news and events, and simplifying comparison shopping.



Web Search:

- Regular users of Web search engines casually expect to receive accurate and near-instantaneous answers to questions and requests merely by entering a short query — a few words — into a text box and clicking on a search button. Underlying this simple and intuitive interface are clusters of computers, comprising thousands of machines, working cooperatively to generate a ranked list of those Web pages that are likely to satisfy the information need embodied in the query.
- These machines identify a set of Web pages containing the terms in the query, compute a score for each page, eliminate duplicate and redundant pages, generate summaries of the remaining pages, and finally return the summaries and links back to the user for browsing.

Consider a simple example.

- If you have a computer connected to the Internet nearby, pause for a minute to launch a browser and try the query "information retrieval" on one of the major commercial Web search engines.
- It is likely that the search engine responded in well under a second. Take some time to review the top ten results. Each result lists the URL for a Web page and usually provides a title and a short snippet of text extracted from the body of the page.

- Overall, the results are drawn from a variety of different Web sites and include sites associated with leading textbooks, journals, conferences, and researchers. As is common for informational queries such as this one, the Wikipedia article may be present.

Other Search Applications:

- Desktop and file system search provides another example of a widely used IR application. A desktop search engine provides search and browsing facilities for files stored on a local hard disk and possibly on disks connected over a local network. In contrast to Web search engines, these systems require greater awareness of file formats and creation times.
- For example, a user may wish to search only within their e-mail or may know the general time frame in which a file was created or downloaded. Since files may change rapidly, these systems must interface directly with the file system layer of the operating system and must be engineered to handle a heavy update load.

Other IR Applications:

1) Document routing, filtering, and selective distribution reverse the typical IR process.
2) Summarization systems reduce documents to a few key paragraphs, sentences, or phrases describing their content. The snippets of text displayed with Web search results represent one example.
3) Information extraction systems identify named entities, such as places and dates, and combine this information into structured records that describe relationships between these entities — for example, creating lists of books and their authors from Web data.

1.2 History of IR:(Early Developments)

The idea of using computers to search for relevant pieces of information was popularized in the article **"As We May Think"** by **Vannevar Bush** in 1945. It would appear that Bush was inspired by patents for a 'statistical machine' - filed by Emanuel Goldberg in the 1920s and '30s - that searched for documents stored on film. The first description of a computer searching for information was described by **Holmstrom** in 1948, detailing an early mention of the Univac computer.

Automated information retrieval systems were introduced in the 1950s: one even featured in the 1957 romantic comedy, Desk Set. In the 1960s, the first large information retrieval research group was formed by **Gerard Salton** at Cornell. By the 1970s several different retrieval techniques had been shown to perform well on small text corpora such as the Cranfield collection (several thousand documents). Large-scale retrieval systems, such as the Lockheed Dialog system, came into use early in the 1970s.

In 1992, the US Department of Defense along with the National Institute of Standards and Technology (NIST), cosponsored the **Text Retrieval Conference (TREC)** as part of the TIPSTER text program. The aim of this was to look into the information retrieval community by supplying the infrastructure that was needed for evaluation of text retrieval methodologies on a very large text collection. This catalyzed research on methods that scale to huge corpora. The introduction of web search engines has boosted the need for very large scale retrieval systems even further.

1.2.1. Timeline:

1950s:

1950: The term **"information retrieval"** was coined by **Calvin Mooers**.

1951: Philip Bagley conducted the earliest experiment in computerized document retrieval in a master thesis at MIT.

1955: Allen Kent joined from Western Reserve University published a paper in American

Documentation describing the **precision and recall** measures as well as detailing a proposed **"framework" for evaluating an IR system** which included statistical sampling methods for determining the number of relevant documents not retrieved.

1959: Hans Peter Luhn published "Auto-encoding of documents for information retrieval."

1960s:

early 1960s: **Gerard Salton** began work on IR at Harvard, later moved to **Cornell.**

1963: Joseph Becker and Robert M. Hayes published text on information retrieval. Becker, Joseph; Hayes, Robert Mayo. Information storage and retrieval: tools, elements, theories. New York, Wiley (1963).

1964:

- Karen Spärck Jones finished her thesis at Cambridge, Synonymy and Semantic Classification, and continued work on computational linguistics as it applies to IR.
- **The National Bureau of Standards sponsored a symposium titled "Statistical Association Methods for Mechanized Documentation." Several highly significant papers, including G. Salton's first published reference (we believe) to the SMART system.**

mid-1960s:

- National Library of Medicine developed MEDLARS Medical Literature Analysis and Retrieval System, the first major machine-readable database and **batch-retrieval system.**
- Project Intrex at MIT.

1965: J. C. R. Licklider published Libraries of the Future.

late 1960s: F. Wilfrid Lancaster completed evaluation studies of the MEDLARS system and published the first edition of his **text on information retrieval.**

1968: Gerard Salton published Automatic Information Organization and Retrieval. John W. Sammon, Jr.'s RADC Tech report "Some Mathematics of Information Storage and Retrieval..." outlined the vector model.

1969: Sammon's "A nonlinear mapping for data structure analysis" (IEEE Transactions on Computers) was the first proposal for visualization interface to an IR system.

1970s:

early 1970s: First online systems—NLM's AIM-TWX, MEDLINE; Lockheed's Dialog; SDC's ORBIT.

1971: Nicholas Jardine and Cornelis J. van Rijsbergen published "The use of hierarchic clustering in information retrieval", which articulated the "cluster hypothesis."

1975: Three highly influential publications by Salton fully articulated his vector processing framework and term discrimination model: A Theory of Indexing (Society for Industrial and Applied Mathematics)

1979: C. J. van Rijsbergen published Information Retrieval (Butterworths). Heavy emphasis on probabilistic models.

1979: Tamas Doszkocs implemented the CITE natural language user interface for MEDLINE at the National Library of Medicine. The CITE system supported free form query input, ranked output and relevance feedback.

1980s

1982: Nicholas J. Belkin, Robert N. Oddy, and Helen M. Brooks proposed the ASK (Anomalous State of Knowledge) viewpoint for information retrieval. This was an important concept, though their automated analysis tool proved ultimately disappointing.

1983: Salton (and Michael J. McGill) published Introduction to Modern Information Retrieval (McGraw-Hill), with heavy emphasis on vector space models.

Mid-1980s: Efforts to develop end-user versions of commercial IR systems. 1989: First World Wide Web proposals by Tim Berners-Lee at CERN.

1990s

1992: First TREC conference.

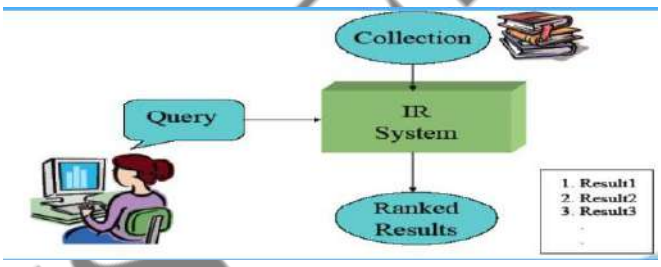
1997: Publication of Korfhage's Information Storage and Retrieval with emphasis on visualization and multi-reference point systems.late 1990s: Web search engines implementation of many features formerly found only in experimental IR systems. Search engines become the most common and maybe best instantiation of IR models.

2000s-present:

More applications, especially Web search and interactions with other fields like Learning to rank, Scalability (e.g., MapReduce), Real-time search

1.3 IR PROBLEMS

Users of modern IR systems, such as search engine users, have information needs of varying complexity, An example of complex information need is as follows:Find all documents that address the role of the Federal Government in financing the operation of the National Railroad Transportation Corporation (AMTRAK)This full description of the user information need is not necessarily a good query to be submitted to the IR system Instead, the user might want to first translate this information need into a queryThis translation process yields a set of keywords, or index terms, which summarize the user information need . Given the user query, the key goal of the IR system is to retrieve information that is useful or relevant to the user



Issues in IR

The main objective of an IR system is to retrieve all the items that are relevant to a user query, while retrieving as few non relevant items as possible.

Main problems in IR:

1. **Document and Query indexing**
   How to best represent their contents?
2. **Query evaluation(or retrieval process)**
   To what extent does a document correspond to a query?
3. **System evaluation**
   How good is a system

Precision & Recall

Are the retrived documents relevant?(precision)

Are all the relevant documents retrieved? (Recall)

Information retrieval is concerned with representing, searching, and manipulating large collections of electronic text and other human- language data.

Three Big Issues in IR

1.Relevance

- It is the fundamental concept in IR.
- A relevant document contains the information that a person was looking for when she submitted a query to the search engine.
- There are many factors that go into a person's decision as to whether a document is relevant.
- These factors must be taken into account when designing algorithms for comparing text and ranking documents.
- Simply comparing the text of a query with the text of a document and looking for an exact match, as might be done in a database system produces very poor results in terms of relevance.
- To address the issue of relevance, **retrieval models** are used.
- A retrieval model is a formal representation of the process of matching
- a query and a document. It is the basis of the ranking algorithm that is used in a search engine to produce the ranked list of documents.
- A good retrieval model will find documents that are likely to be considered relevant by the person who submitted the query.
- The retrieval models used in IR typically model the statistical properties of text rather than the linguistic structure. For example, the ranking algorithms are concerned with the counts of word occurrences than whether the word is a noun or an adjective.

2.Evaluation

- Two of the evaluation measures are precision and recall.
    - Precision is the proportion of retrieved documents that are relevant. Recall is the proportion of relevant documents that are retrieved.

$$\text{Precision} = \frac{\text{Relevant documents} \cap \text{Retrieved documents}}{\text{Retrieved documents}}$$

    - $\text{Recall} = \frac{\text{Relevant documents} \cap \text{Retrieved documents}}{\text{Relevant documents}}$
    - When the recall measure is used, there is an assumption that all the relevant documents for a given query are known. Such an assumption is clearly problematic in a web search environment, but with smaller test collection of documents, this measure can be useful. It is not suitable for large volumes of log data.

3.Emphasis on users and their information needs

- The users of a search engine are the ultimate judges of quality. This has led to numerous studies on how people interact with search engines and in particular, to the development of techniques to help people express their information needs.
- Text queries are often poor descriptions of what the user actually wants compared to the request to a database system, such as for the balance of a bank account.
- Despite their lack of specificity, one-word queries are very common in web search. A one-word query such as "cats" could be a request for information on where to buy

cats or for a description of the Cats (musical).

- Techniques such as query suggestion, query expansion and relevance feedback use interaction and context to refine the initial query in order to produce better ranked results.
- The figure summarizes the major issues involved in search engine design

1.4 USER TASK

The User Task.- The user of a retrieval system has to translate his information need into a query in the language provided by the system. With an information retrieval system, this normally implies specifying a set of words which convey the semantics of the information need.

1. Consider a user who seeks information on a topic of their interest :This user first translates their information need into a query, which requires specifying the words that compose the query In this case, we say that the user is searching or querying for information of their interest

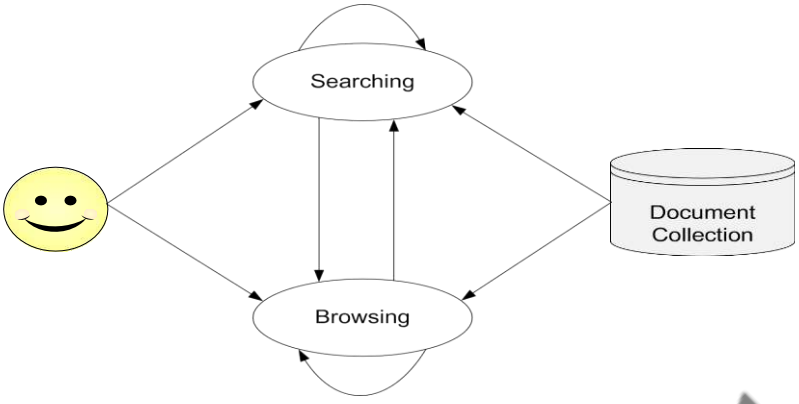
2. Consider now a user who has an interest that is either poorly defined or inherently broad

For instance, the user has an interest in car racing and wants to browse documents on Formula 1 and Formula Indy, In this case, we say that the user is browsing or navigating the documents of the collection

The general objective of an **Information Retrieval System** is to minimize the time it takes for a user to locate the **information** they need. The goal is to provide the **information** needed to satisfy the user's question. Satisfaction does not necessarily mean finding all **information** on a particular issue.

The user of a retrieval system has to translate his information need into a query in the language provided by the system. With an information retrieval system, this normally implies specifying a set of words which convey the semantics of the information need. With a data retrieval system, a query expression (such as, for instance, a regular expression) is used to convey the constraints that must be satisfied by objects in the answer set. In both cases, we say that the user searches for useful information executing a retrieval task.

Consider now a user who has an interest which is either poorly defined or which is inherently broad. For instance, the user might be interested in documents about car racing in general. In this situation, the user might use an interactive interface to simply look around in the collection for documents related to car racing. For instance, he might find interesting documents about Formula 1 racing, about car manufacturers, or about the `24 Hours of Le Mans.' Furthermore, while reading about the `24 Hours of Le Mans', he might turn his attention to a document which provides directions to Le Mans and, from there, to documents which cover tourism in France. In this situation, we say that the user is browsing the documents in the collection, not searching. It is still a process of retrieving information, but one whose main objectives are not clearly defined in the beginning and whose purpose might change during the interaction with the system.



Information versus Data Retrieval

Information Retrieval Vs information Extraction

Information Retrieval:

Given a set of terms and a set of document terms select only most relevant document (precision) ,and preferably all the relevant ones(recall)

Infromation Extraction:

Extract from the text what the document means.Data retrieval**:** the task of determining which documents of a collection contain the keywords in the user query

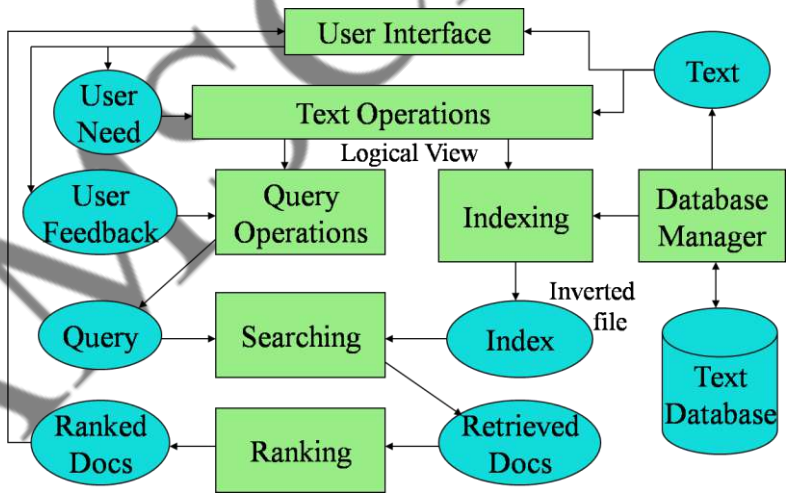
Data retrieval system

Ex: relational databases

Deals with data that has a well defined structure and semantics

Data retrieval does not solve the problem of retrieving information about a subject or topic

1.5 The IR System



COMPONENTS OF IR:

The above figure shows the architecture of IR System with the Specified Components

Components:

- Text operations
- Indexing
- Searching
- Ranking

- User Interface
- Query operations

Text operation:

Text Operations forms index words (tokens).

Stop word removal , Stemming

Indexing:

Indexing constructs an inverted index of word to document pointers.

Searching:

Searching retrieves documents that contain a given query token from the inverted index.

Ranking :

Ranking scores all retrieved documents according to a relevance metric.

User Interface:

User Interface manages interaction with the user:

- Query input and document output.
- Relevance feedback.
- Visualization of results.

Query Operations:

Query Operations transform the query to improve retrieval:

- Query expansion using a thesaurus.
- Query transformation using relevance feedback.

First of all, before the retrieval process can even be initiated, it is necessary to define the text database. This is usually done by the manager of the database, which specifies the following: (a) the documents to be used, (b) the operations to be performed on the text, and (c) the text model (i.e., the text structure and what elements can be retrieved). The text operations transform the original documents and generate a logical view of them.

Once the logical view of the documents is defined, the database manager builds an index of the text. An index is a critical data structure because it allows fast searching over large volumes of data. Different index structures might be used, but the most popular one is the inverted file. The resources (time and storage space) spent on defining the text database and building the index are amortized by querying the retrieval system many times.

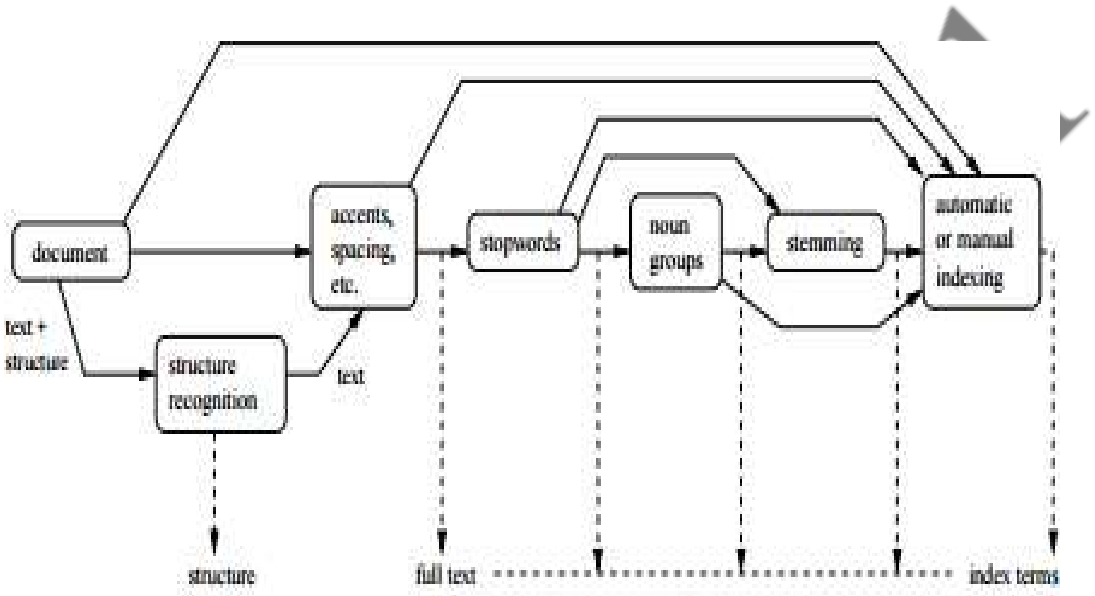
Given that the document database is indexed, the retrieval process can be initiated. The user first specifies a user need which is then parsed and transformed by the same text operations applied to the text. Then, query operations might be applied before the actual query, which provides a system representation for the user need, is generated. The query is then processed to obtain the retrieved documents. Fast query processing is made possible by the index structure previously built.

Before been sent to the user, the retrieved documents are ranked according to a likelihood of relevance. The user then examines the set of ranked documents in the search for useful information. At this point, he might pinpoint a subset of the documents seen as definitely of interest and initiate a user feedback cycle. In such a cycle, the system uses the documents selected by the user to change the query formulation. Hopefully, this modified query is a better representation

1.6The Software Architecture of the IR System

Logical View of the Documents:

Due to historical reasons, documents in a collection are frequently represented through a set of index terms or keywords. Such keywords might be extracted directly from the text of the document or might be specified by a human subject (as frequently done in the information sciences arena). No matter whether these representative keywords are derived automatically



Logical view of a document: from full text to a set of index terms.

With very large collections, however, even modern computers might have to reduce the set of representative keywords. This can be accomplished through the elimination of stopwords (such as articles and connectives), the use of stemming (which reduces distinct words to their common grammatical root),and the identification of noun groups (which eliminates adjectives, adverbs, and verbs). Further, compression might be employed. These operations are called text operations (transformations). Text operations reduce the complexity of the document representation and allow moving the logical view from that of a full text to that of a set of index terms.

The primary data structure of most of the **IR systems** is in the form of inverted index. We can **define** an inverted index as a data structure that list, for every word, all documents that contain it and frequency of the occurrences in document. It makes it easy to search for 'hits' of a query word.

ARCHITECTURE OF IR SYSTEM

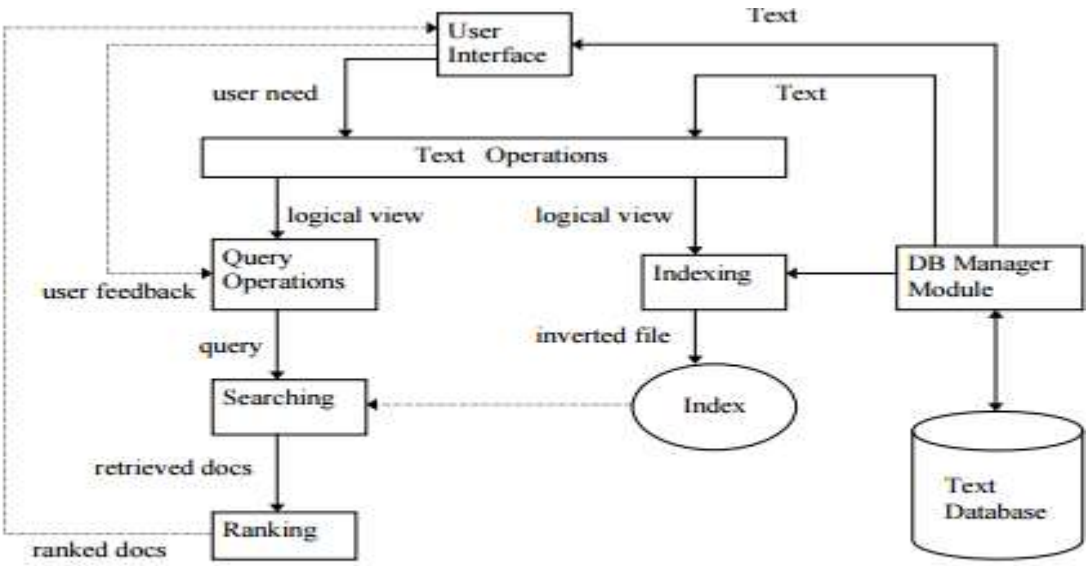
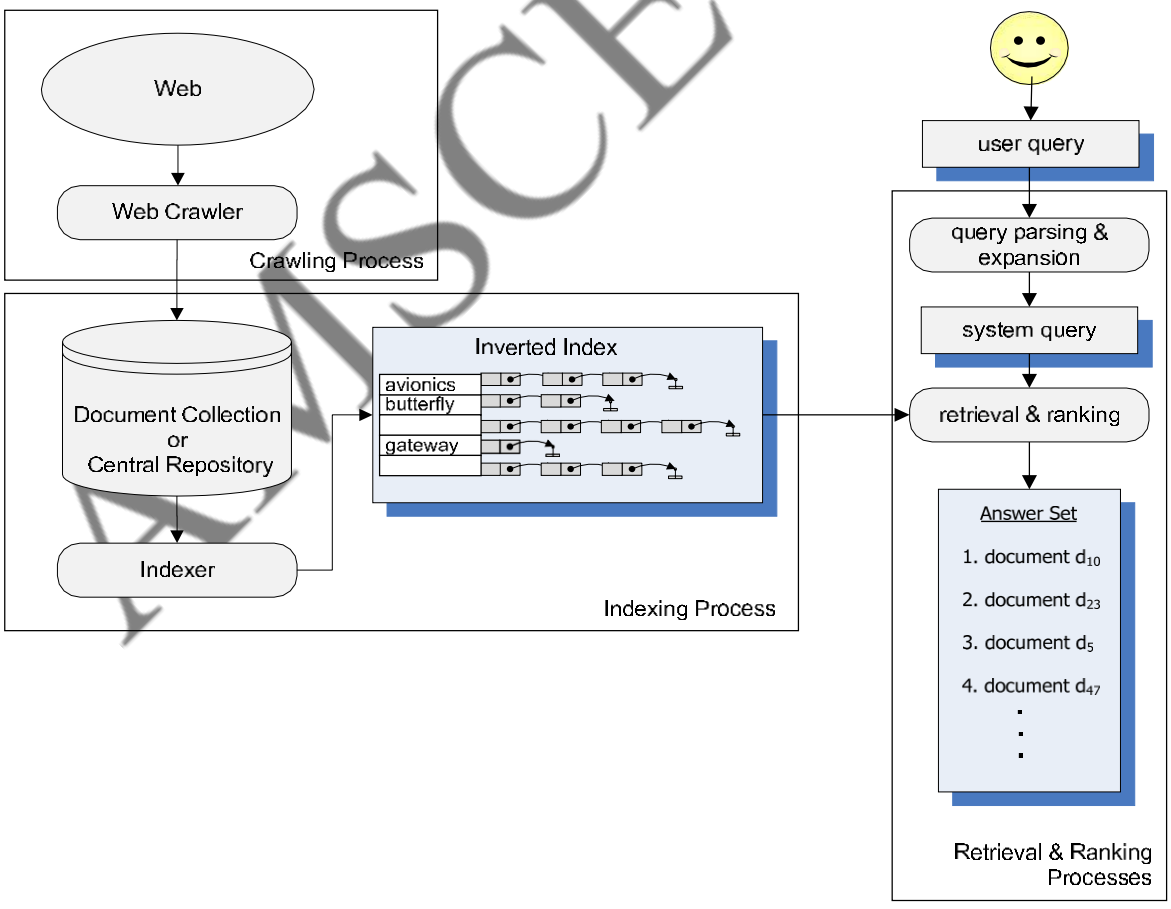
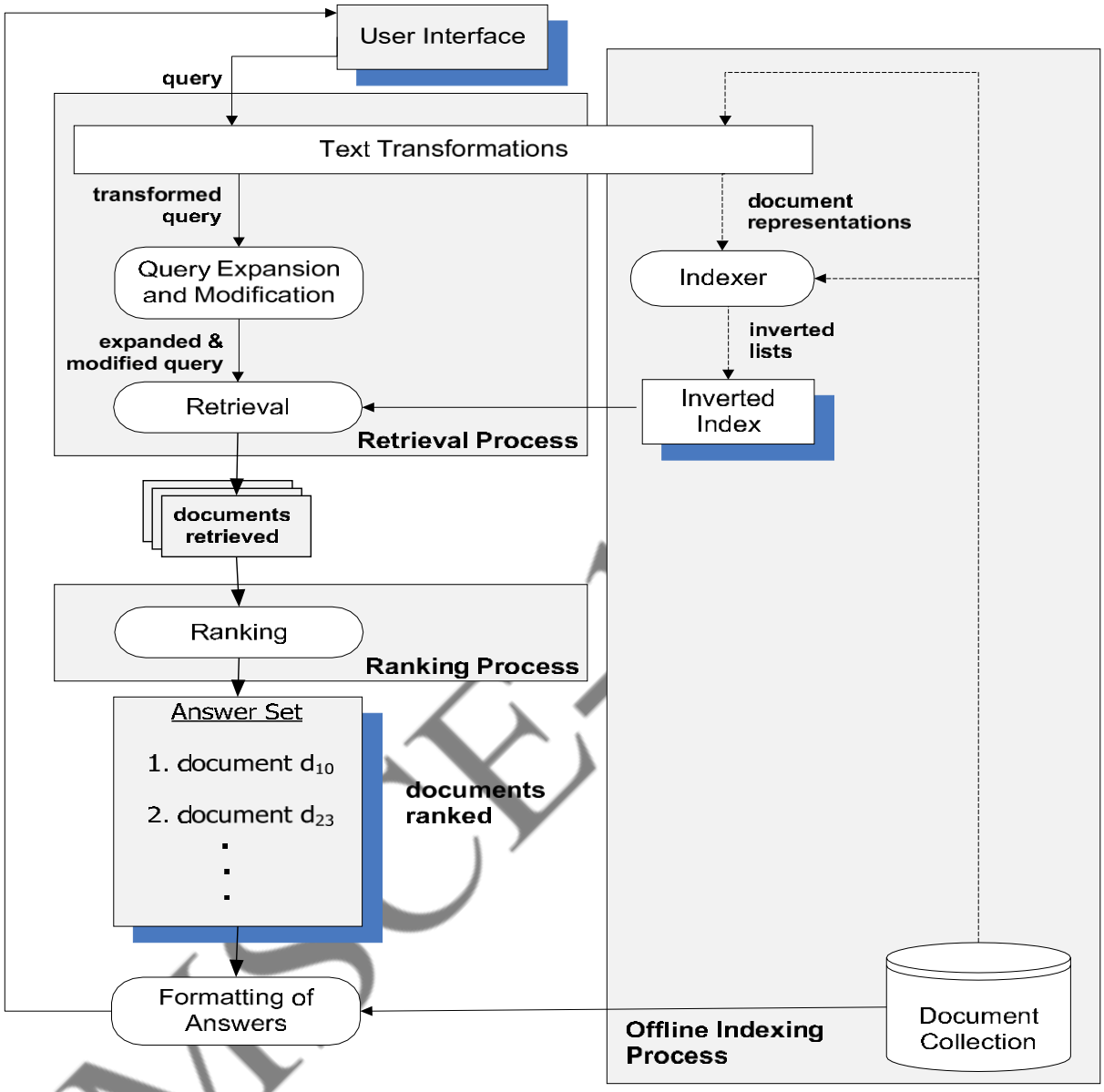


Figure . The process of retrieving information

High level software architecture of an IR system



The processes of indexing, retrieval, and ranking



The Retrieval Process:

To describe the retrieval process, we use a simple and generic software architecture as shown in Figure First of all, before the retrieval process can even be initiated, it is necessary to dene the text database. This is usually done by the manager of the database, which species the following: (a) the documents to be used, (b) the operations to be performed on the text, and (c) the text model (i.e., the text structure and what elements can be retrieved). The text operations transform the original documents and generate a logical view of them.

Once the logical view of the documents is defined, the database manager (using the DB Manager Module) builds an index of the text. An index is a critical data structure because it allows fast searching over large volumes of data. Different index structures might be used, but the most popular one is the inverted index as indicated in Figure. The resources (time and storage space) spent on defining the text database and building the index are amortized by querying the retrieval system many times. Given that the document database is indexed,

the retrieval process can be initiated. The user need which is then parsed and transformed by the same text operations applied to the text. Then, query operations might be applied before the actual query, which provides a system representation for the user need, is generated. The query is then processed to obtain the retrieved documents. Fast query processing is made possible by the index structure previously built. Before been sent to the user, the retrieved documents are ranked according to a likelihood of relevance. The user then examines the set of ranked documents in the search for useful information. At this point, he might pinpoint a subset of the documents seen as definitely of interest and initiate a user feedback cycle. In such a cycle, the system uses the documents selected by the user to change the query formulation. Hopefully, this modified query is a better representation of the real user need.

1.7 THE WEB

With the rapid growth of the Internet, more information is available on the Web and Web information retrieval presents additional technical challenges when compared to classic information retrieval due to the **heterogeneity and size of the web.**

1.Web information retrieval is unique due to the **dynamism, variety of languages used, duplication, high linkage, ill formed query and wide variance in the nature of users.**

2. Another issue is the **rising number of inexperienced users**, they the majority of Web users are not very sophisticated searchers.

3. Many software tools are available for web information retrieval such as search engines (Google and Alta vista), hierarchical directories (Yahoo), many other software agents and collaborative filtering systems. The commonly cited problems in search engines are the slow speed of retrieval, communication delays, and poor quality of retrieved results.

In addition, in the current scenario, **multimedia information is increasingly** becoming available on the Web and modem IR systems must be capable of tackling not only the text but also multimedia information like sound, image and video

IR on the web Vs.IR

Traditional IR systems normally index a closed collection of documents, which are mainly text-based and usually offer little linkage between documents. Traditional IR systems are often referred to as *full-text retrieval systems*. Libraries were among the first to adopt IR to index their catalogs and later, to search through information which was typically imprinted onto CD- ROMs. The main aim of traditional IR was to return relevant documents that satisfy the user's information need. Although the main goal of satisfying the user's need is still the central issue in web IR (or web search), there are some very specific challenges that web search poses that have required new and innovative solutions.

- The first important difference is the scale of web search, as we have seen that the current size of the web is approximately 600 billion pages. This is well beyond the size of traditional document collections.
- The Web is dynamic in a way that was unimaginable to traditional IR in terms of its rate of change and the different types of web pages ranging from static types (HTML, portable document format (PDF), DOC, Postscript, XLS) to a growing number

dynamic pages written in scripting languages such a JSP, PHP or Flash. We also mention that a large number of images, videos, and a growing number of programs are delivered through the Web to our browsers.

- The Web also contains an enormous amount of duplication, estimated at about 30%. Such redundancy is not present in traditional corpora and makes the search engine's task even more difficult.
- The quality of web pages vary dramatically; for example, some web sites create web pages with the sole intention of manipulating the search engine's ranking, documents may contain misleading information, the information on some pages is just out of date, and the overall quality of a web page may be poor in terms of its use of language and the amount of useful information it contains. The issue of quality is of prime importance to web search engines as they would very quickly lose their audience if, in the top-ranked positions, they presented to users poor quality pages.
- The range of topics covered on the Web is completely open, as opposed to the closed collections indexed by traditional IR systems, where the topics such as in library catalogues, are much better defined and constrained.
- Another aspect of the Web is that it is globally distributed. This poses serious logistic problems to search engines in building their indexes, and moreover, in delivering a service that is being used from all over the globe. The sheer size of the problem is daunting, considering that users will not tolerate anything but an immediate response to their query. Users also vary in their level of expertise, interests, information- seeking tasks, the language(s) they understand, and in many other ways.

	Classical IR	**Web IR**
Volume	Large	Huge
Data Quality	Clean ,No Duplicates	Noisy, duplicates Available
Data change rate	Infrequent	In flux
Data accessibility	Accessible	Partially accessible
Format diversity	Homogeneous	Widely Diverse
Documents	Text	HTML
No.of Matches	Small	Large
IR techniques	Content based	Link based

COMPARISION

Sl. No	Differentiator	Web Search	IR
1	Languages	Documents in many different languages. Usually search engines use full text indexing; no additional subject analysis.	Databases usually cover only one language or indexing of documents written in different languages with the same vocabulary.
2	File types	Several file types, some hard to index because of a lack of textual information.	Usually all indexed documents have the same format (e.g. PDF) or only bibliographic information is provided.
3	Document length	Wide range from very short to very long. Longer documents are often divided into parts.	Document length varies, but not to such a high degree as with the Web documents
4	Document structure	HTML documents are semi structures.	Structured documents allow complex field searching
5	Spam	Search engines have to decide which documents are suitable for indexing.	Suitable document typesare defined in the process of database design.
6	Amount of data, size of databases	The actual size of the Web is unknown. Complete indexing of the whole Web is impossible.	Exact amount of data can be determined when using formal criteria.
7	Type of queries	Users have little knowledge how to search; very short queries (2-3 words).	Users know the retrieval language; longer, exact queries.
8	User interface	Easy to use interfaces suitable for laypersons.	Normally complex interfaces; practice needed to conduct searches.
9	Ranking	Due to the large amount of hits relevance ranking is the norm.	Relevance ranking is often not needed because the users know how to constrain the amount of hits.
10	Search functions	Limited possibilities.	Complex query languages allow narrowing searches.

1.8 E-PUBLISHING-ERA

Since its inception, the Web became a huge success - Well over 20 billion pages are now available and accessible in the Web More than one fourth of humanity now access the Web on a regular basis.

Why is the Web such a success? What is the single most important characteristic of the Web that makes it so revolutionary?

In search for an answer, let us dwell into the life of a writer who lived at the end of the 18th Century.

- She finished the first draft of her novel in 1796 The first attempt of publication was refused without a reading. The novel was only published 15 years later! She got a flat fee of $110, which meant that she was not paid anything for the many subsequent editions Further, her authorship was anonymized under the reference "By a Lady"
- Pride and Prejudice is the second or third best loved novel in the UK ever, after The Lord of the Rings and Harry Potter. It has been the subject of six TV series and five film versions The last of these, starring Keira Knightley and Matthew Macfadyen, grossed over 100 million dollars
- Jane Austen published anonymously her entire life Throughout the 20th century, her novels have never been out of print, Jane Austen was discriminated because there was no freedom to publish in the beginning of the 19th century .
- The Web, unleashed by the inventiveness of Tim Berners-Lee, changed this once and for all It did so by universalizing freedom to publish - The Web moved mankind into a new era, into a new time, into The e-Publishing Era.

The term "**electronic publishing**" is primarily used in the 2010s to refer to online and web-based **publishers**, the term has a history of being used to describe the development of new forms of production, distribution, and user interaction in regard to computer-based production of text and other interactivemedia.

The first digitization projects were transferring physical content into digital content. Electronic publishing is aiming to integrate the whole process of editing and publishing (production, layout, publication) in the digital world.

The traditional publishing, and especially the creation part, were first revolutionized by new desktop publishing softwares appearing in the 1980s, and by the text databases created for the encyclopedias and directories. At the same time the multimedia was developing quickly, combining book, audiovisual and computer science characteristics. CDs and DVDs appear, permitting the visualization of these dictionaries and encyclopedias on computers.

The arrival and democratization of Internet is slowly giving small publishing houses the opportunity to publish their books directly online. Some websites, like Amazon, let their users buy eBooks; Internet users can also find many educative platforms (free or not), encyclopedic websites like Wikipedia, and even digital magazines platforms. The eBook then becomes more and more accessible through many different supports, like the e-reader and even smartphones. The digital book had, and still has, an important impact on publishing houses and their economical models; it is still a moving domain, and they yet have to master the new ways of publishing in a digital era

1.9 HOW THE WEB CHANGED SEARCH

Web search is today the most prominent application of IR and its techniques—the ranking and indexing components of any search engine are fundamentally IR pieces of technology

The first major impact of the Web on search is related to the characteristics of the document collection itself

- The Web is composed of pages distributed over millions of sites and connected through hyperlinks
- This requires collecting all documents and storing copies of them in a central repository, prior to indexing
- This new phase in the IR process, introduced by the Web, is called crawling

The second major impact of the Web on search is related to:

- The size of the collection
- The volume of user queries submitted on a daily basis
- As a consequence, performance and scalability have become critical characteristics of the IR system

The third major impact: in a very large collection, predicting relevance is much harder than before

- Fortunately, the Web also includes new sources of evidence
- Ex: hyperlinks and user clicks in documents in the answer set

The fourth major impact derives from the fact that the Web is also a medium to do business

- Search problem has been extended beyond the seeking of text information to also encompass other user needs
- Ex: the price of a book, the phone number of a hotel, the link for downloading a software

The fifth major impact of the Web on search is Web spam

- Web spam: abusive availability of commercial information disguised in the form of informational content
- This difficulty is so large that today we talk of Adversarial Web Retrieval

1.10 PRACTICAL ISSUES IN THE WEB

- **Security**

  Commercial transations over the Internet are not yet a completely safe procedure
- **Privacy**

  Frequently, people are willing to exchange information as long as it does not become public
- **Copyright and patent rights**

  It is far from clear how the wide spread of data on the Web affects copyright and patent laws in the various countries
- **Log In Issue**

  One of the most common problems faced by online businesses is the inability to log in to the control panel. You need easy access to the control panel for additions and deletions of content and for other purposes.
- **Frequent Technical Breakdown:**

Running a website business effectively is only possible when all the functional parameters respond to your input quickly and smoothly. Unfortunately, most of the times, this does not happen.

- **Slow Performance of Web Server**

  Slow web server is one of the biggest headaches that businesses have to deal with. When your customers encounter pages that load slowly, they tend to abandon their search and look for other alternatives.

- **Server Limitations:**

  A few hosting companies follow the undesirable business practice of not disclosing their limit in terms of space and bandwidth. They try to serve more customers with their limited resources which can result in major performance issues in the long term

1.11 HOW PEOPLE SEARCH

User interaction with search interfaces differs depending on

- the type of task
- the domain expertise of the information seeker
- the amount of time and effort available to invest in the process

Marchionini makes a distinction between **information lookup and exploratory search**

Information lookup tasks

1. are akin to fact retrieval or question answering
2. can be satisfied by discrete pieces of information: numbers, dates, names, or Web sites
3. can work well for standard Web search interactions

Exploratory search is divided into learning and investigating tasks

Learning search

1. requires more than single query-response pairs
2. requires the searcher to spend time

- scanning and reading multiple information items
- synthesizing content to form new understanding

Investigating refers to a longer-term process which

- involves multiple iterations that take place over perhaps very long periods of time
- may return results that are critically assessed before being integrated into personal and professional knowledge bases
- may be concerned with finding a large proportion of the relevant information available

Classic × Dynamic Model

Classic notion of the information seeking process:

1. **problem identification**
2. **articulation of information need(s)**
3. **query formulation**

4. results evaluation

More recent models emphasize the dynamic nature of the search process

- The users learn as they search
- Their information needs adjust as they see retrieval results and other document surrogates

This dynamic process is sometimes referred to as theberry picking model of search

Navigation × Search

Navigation: the searcher looks at an information structure and browses among the available information

This browsing strategy is preferrable when the information structure is well-matched to the user's information need

- it is mentally less taxing to recognize a piece of information than it is to recall it
- it works well only so long as appropriate links are available

If the links are not avaliable, then the browsing experience might be frustrating

Search Process

- Numerous studies have been made of people engaged in the search process
- The results of these studies can help guide the design of search interfaces
- One common observation is that users often reformulate their queries with slight modifications
- Another is that searchers often search for information that they have previously accessed

The users' search strategies differ when searching over previously seen materials

Researchers have developed search interfaces support both query history and revisitation

Studies also show that it is difficult for people to determine whether or not a document is relevant to a topic,Other studies found that searchers tend to look at only the top-ranked retrieved results.Further, they are biased towards thinking the top one or two results are better than those beneath them.

Studies also show that people are poor at estimating how much of the relevant material they have found ,Other studies have assessed the effects of knowledge of the search process itself.

These studies have observed that experts use different strategies than novices searchers

1.12 SEARCH INTERFACES TODAY

How does an information seeking session begin in online information systems?

- The most common way is to use a Web search engine
- Another method is to select a Web site from a personal collection of already-visited sites
- Online bookmark systems are popular among a smaller segment of users Ex: Delicious.com
- Web directories are also used as a common starting point, but have been largely replaced by search engines

The primary methods for a searcher to express their information need are either

1. entering words into a search entry form

2. selecting links from a directory or other information organization display

For Web search engines, the query is specified in textual form. Typically, Web queries today are very short consisting of one to three words

Query Specification

Short queries reflect the standard usage scenario in which the user *tests the waters*:

- If the results do not look relevant, then the user reformulates their query
- If the results are promising, then the user navigates to the most relevant-looking web site

Query Specification Interface

The standard interface for a textual query is a **search box entry form**

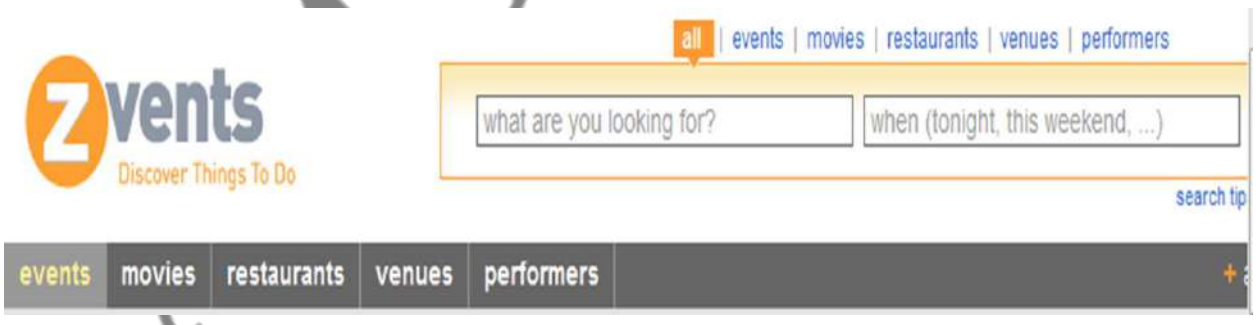
Studies suggest a relationship between query length and the width of the entry form

➢ Results found that either small forms discourage long queries or wide forms encourage longer queries

❖ Some entry forms are followed by a form that filters the query in some way , For instance, at yelp.com, the user can refine the search by location using a second form

Notice that the yelp.com form also shows the user's home location, if it has been specified previously

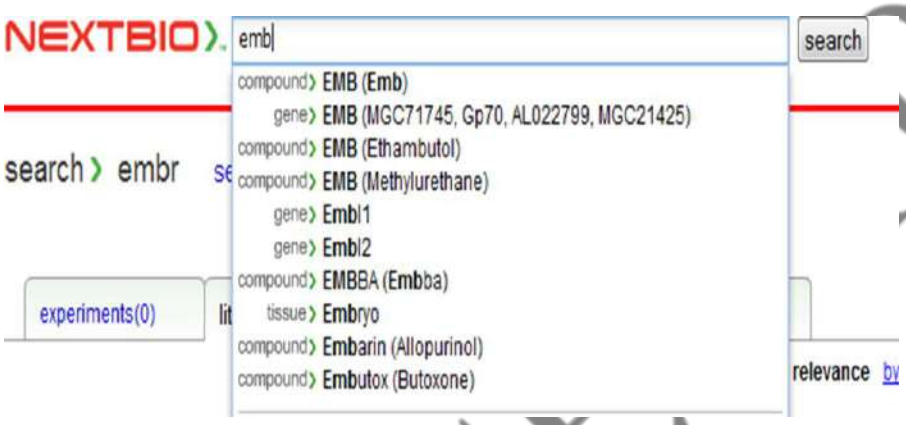
❖ Some search forms show hints on what kind of information should be entered into each form , For instance, in zvents.com search, the first box is labeled "what are you looking for"?



❖ Some interfaces show a list of query suggestions as the user types the query - this is referred to as **auto-complete, auto-suggest**, or **dynamic query suggestions**

❖ Dynamic query suggestions, from Netflix.com

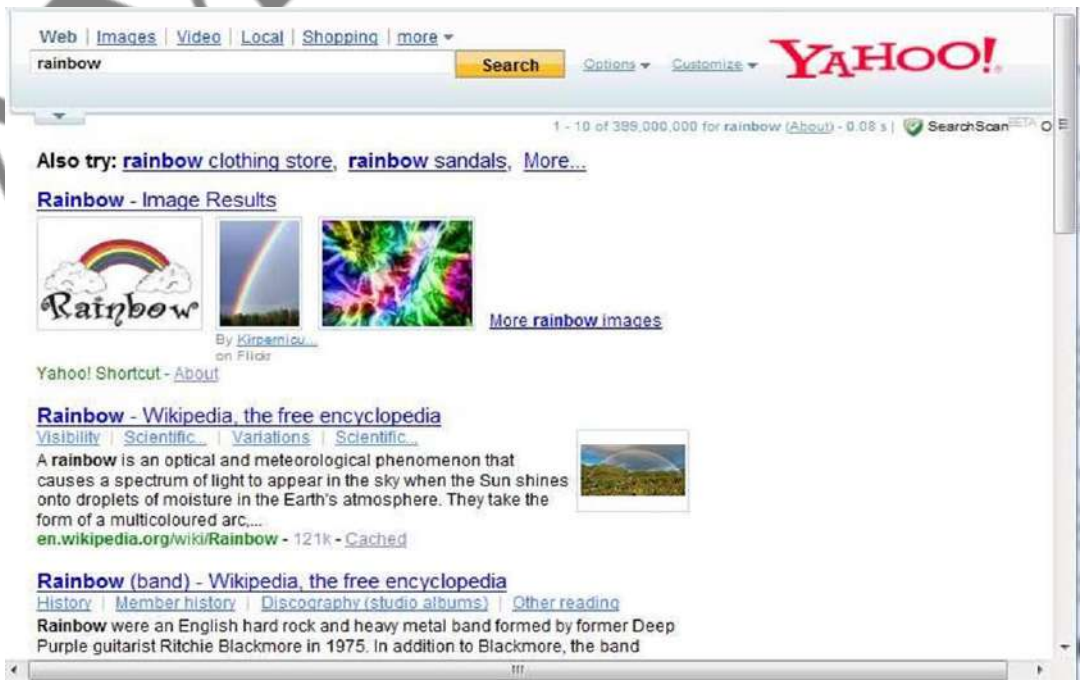
❖ Dynamic query suggestions, grouped by type, from NextBio.com:



RETRIVAL RESULTS DISPLAY

When displaying search results, either the documents must be shown in full, or else the searcher must be presented with some kind of representation of the content of those documents

❖ For example, a query on a term like "rainbow" may return sample images as one entry in the results listing



❖ A query on the name of a sports team might retrieve the latest game scores and a link to buy tickets

QUERY REFORMULATION

There are tools to help users reformulate their query

- One technique consists of showing terms related to the query or to the documents retrieved in response to the query

A special case of this is spelling corrections or suggestions

- Usually only one suggested alternative is shown: clicking on that alternative re-executes the query
- In years back, the search results were shown using the purportedly incorrect spelling
- **Relevance feedback** is another method whose goal is to aid in query reformulation
- The main idea is to have the user indicate which documents are relevant to their query
- In some variations, users also indicate which terms extracted from those documents are relevant
- The system then computes a new query from this information and shows a new retrieval set

ORGANISING SEARCH RESULTS

Organizing results into meaningful groups can help users understand the results and decide what to do next

Popular methods for grouping search results: category systems and clustering

Category system: meaningful labels organized in such a way as to reflect the concepts relevant to a domain

The most commonly used category structures are **flat**, **hierarchical**, and **faceted** categories.Most Web sites organize their information into general categories

Clustering refers to the grouping of items according to some measure of similarity

It groups together documents that are similar to one another but different from the rest of the collection.

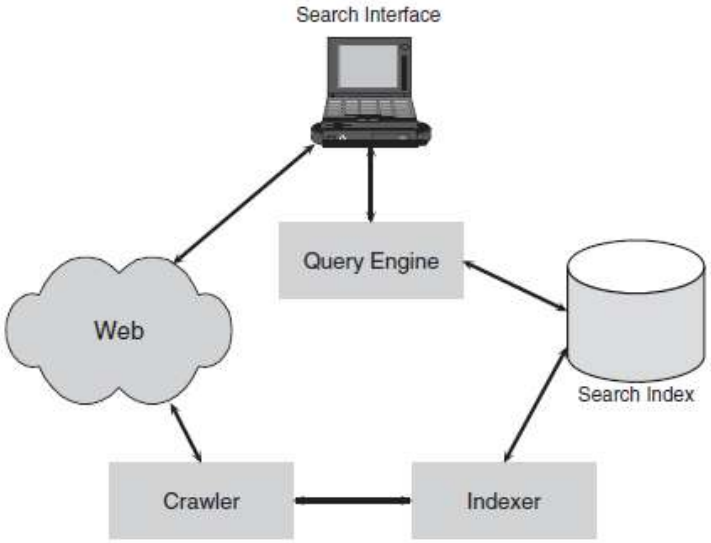
The greatest advantage of clustering- is that it is fully automatable

The disadvantages of clustering include-an unpredictability in the form and quality of results , the difficulty of labeling the groups

COMPONENTS OF SEARCH ENGINE

The main components of a search engine are

1) Crawler
2) Indexer
3) Search index
4) Query engine
5) Search interface.



1) **Crawler**:

A *web crawler* is a software program that traverses web pages, downloads them for indexing, and follows the hyperlinks that are referenced on the downloaded pages; a web crawler is also known as a *spider*, a *wanderer* or a *software robot*.

2) I**ndexer**:The second component is the *indexer* which is responsible for creating the search index from the web pages it receives from the crawler

3) Search Index:

The *search index* is a data repository containing all the information the search engine needs to match and retrieve web pages. The type of data structure used to organize the index is known as an *inverted file*.

4) Query Engine:

The *query engine* is the algorithmic heart of the search engine. The inner working of a commercial query engine is a well-guarded secret, since search engines are rightly paranoid, fearing web sites who wish to increase their ranking by unfairly taking advantage of the algorithms the search engine uses to rank result pages.

5) Search Interface:

Once the query is processed, the query engine sends the results list to the *search interface*, which displays the results on the user's screen. The user interface provides the look and feel of the search engine, allowing the user to submit queries, browse the results list, and click on chosen web pages for further browsing.

1.13 VISUALIZATION IN SEARCH INTERFACES

Experimentation with visualization for search has been primarily applied in the following ways:

- Visualizing Boolean syntax
- Visualizing query terms within retrieval results
- Visualizing relationships among words and documents
- Visualization for text mining

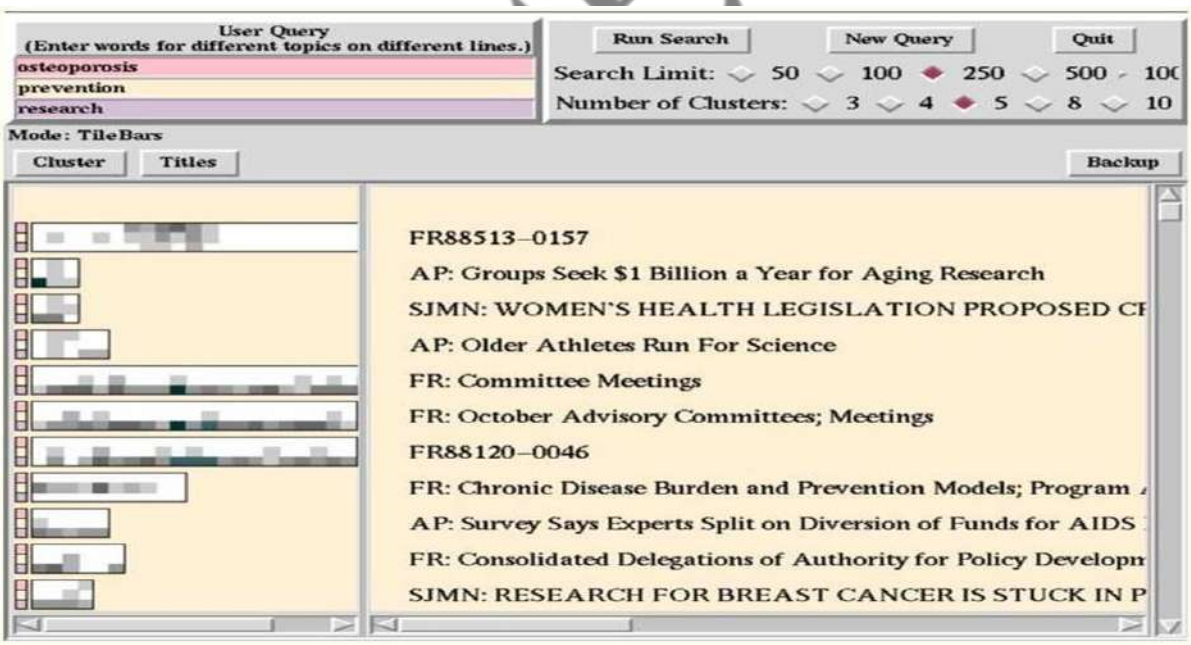
Visualizing Boolean syntax

Boolean query syntax is difficult for most users and is rarely used in Web search ,For many years, researchers have experimented with how to visualize Boolean query specification. A common approach is to show Venn diagrams. A more flexible version of this idea was seen in the VQuery system, proposed by Steve Jones

Visualizing Query Terms

Understanding the role of the query terms within the retrieved docs can help relevance assessment , Experimental visualizations have been designed that make this role more explicit. In the TileBars interface, for instance, documents are shown as horizontal glyphs ,the locations of the query term hits marked along the glyph. The user is encouraged to break the query into its different facets, with one concept per line. Then, the lines show the frequency of occurrence of query terms within each topic.

The TileBars interface Representation



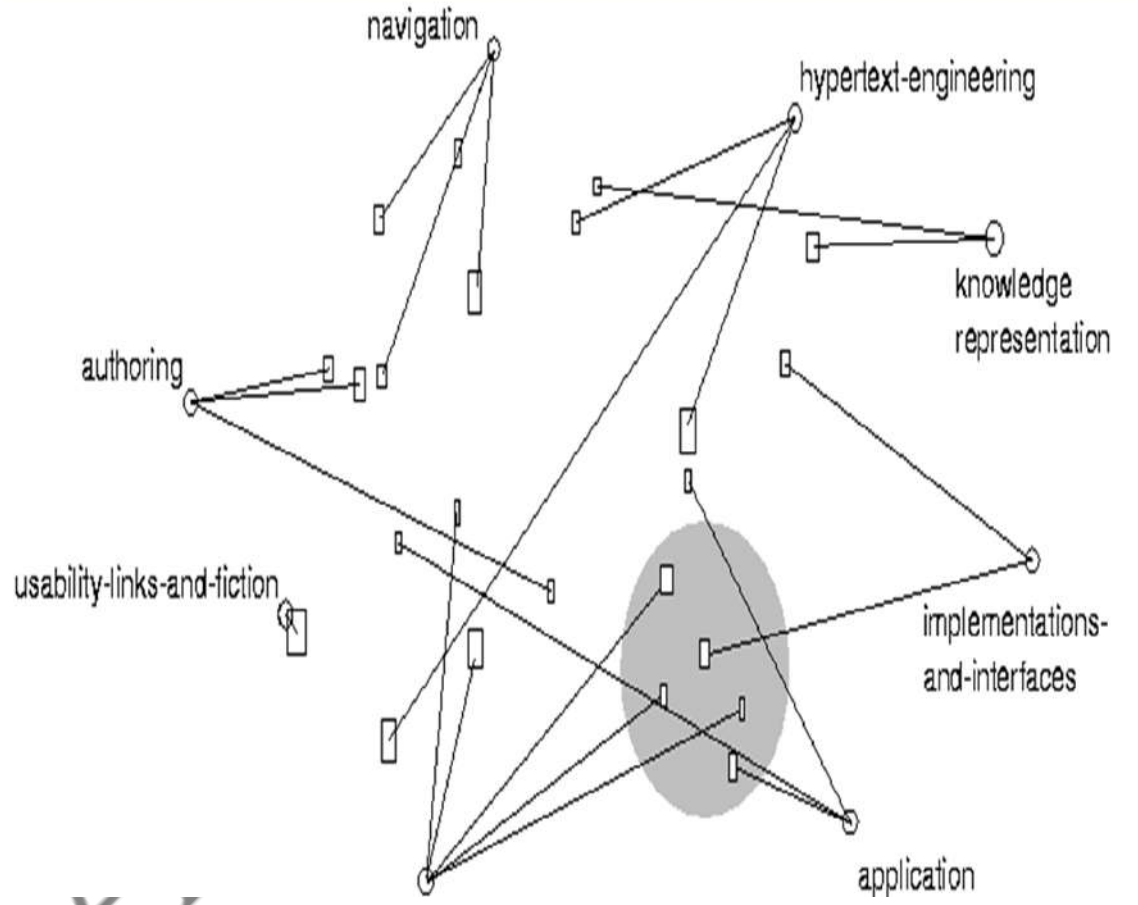
Words and Docs Relationships

- Numerous works proposed variations on the idea of placing words and docs on a two-

dimensional canvas

- In these works, proximity of glyphs represents semantic relationships among the terms or documents
- An early version of this idea is the **VIBE interface**
- Documents containing combinations of the query terms are placed midway between the icons representing those terms
- The Aduna Autofocus and the Lyberworld projects presented a 3D version of the ideas behind

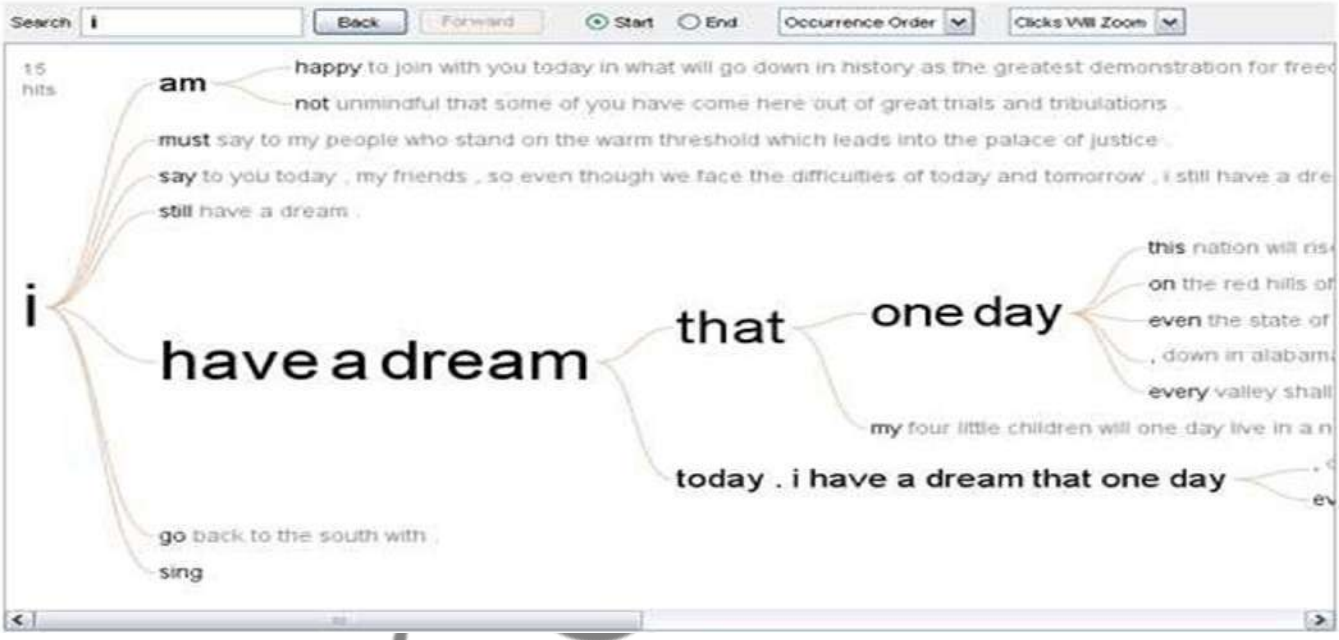
VIBE DISPLAY REPRESENTATION



Another idea is to map docs or words from a very high- dimensional term space down into a 2D plane ,The docs or words fall within that plane, using 2D or 3D

Visualization for Text Mining

- ❖ Visualization is also used for purposes of analysis and exploration of textual data
- ❖ Visualizations such as the Word Tree show a piece of a text concordance
- ❖ It allows the user to view which words and phrases commonly precede or follow a given word
- ❖ The Word Tree visualization, on Martin Luther King's , *I have a dream* speech, from Wattenberg *et al*



Visualization is also used in search interfaces intended for analysts, an example is the TRIST information *triage* system, from Proulx *et al* ,In this system, search results is represented as document icons- thousands of documents can be viewed in one display.

UNIT II MODELING AND RETRIEVAL EVALUATION

Basic IR Models – Boolean Model – TF-IDF (Term Frequency/Inverse Document Frequency) Weighting – Vector Model – Probabilistic Model – Latent Semantic Indexing Model – Neural Network Model – Retrieval Evaluation – Retrieval Metrics – Precision and Recall – Reference Collection – User-based Evaluation – Relevance Feedback and Query Expansion – Explicit Relevance Feedback

2.1 BOOLEAN RETRIEVAL MODEL:

The Boolean retrieval model was used by the earliest search engines and is still in use today. It is also known as **exact-match retrieval** since documents are retrieved if they exactly match the query specification, and otherwise are not retrieved. Although this defines a very simple form of ranking, Boolean retrieval is not generally described as a ranking algorithm. This is because the Boolean retrieval model assumes that all documents in the retrieved set are equivalent in terms of relevance, in addition to the assumption that relevance is binary. The name Boolean comes from the fact that there only two possible outcomes for query evaluation (TRUE and FALSE) and because the query is usually specified using operators **from Boolean logic (AND, OR, NOT).** Searching with a regular expression utility such as **unix grep** is another example of exact-match retrieval.

A fat book which many people own is Shakespeare's Collected Works. Suppose you wanted to determine which plays of Shakespeare contain the words **Brutus AND Caesar AND NOT Calpurnia.** One way to do that is to start at the beginning and to read through all the text, noting for each play whether it contains Brutus and Caesar and excluding it from consideration if it contains Calpurnia. The simplest form of document retrieval is for a computer to do this sort of linear scan through documents.

With modern computers, for simple querying of modest collections (the size of Shakespeare's Collected Works is a bit under one million words of text in total), you really need nothing more. But for many purposes, you do need more:

1. **To process large document collections quickly.** The amount of online data has grown at least as quickly as the speed of computers, and we would now like to be able to

search collections that total in the order of billions to trillions of words.

2. **To allow more flexible matching operations**. For example, it is impractical to perform the query **Romans NEAR countrymen** with grep, where NEAR might be defined as "**within 5 words"** or "within the same sentence".

3. **To allow ranked retrieval:** in many cases you want the best answer to an information need among many documents that contain certain words.

The way to avoid linearly scanning the texts for each query is to index the documents in advance. Let us stick with Shakespeare's Collected Works, and use it to introduce the basics of the Boolean retrieval model. Suppose we record for each document – here a play of Shakespeare's – whether it contains each word out of all the words Shakespeare used (Shakespeare used about 32,000 different words). The result is a binary term-document incidence, as in Figure. Terms are the indexed units; they are usually words, and for the moment you can think of them as words.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	…
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
…							

Figure : A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise.

To answer the query Brutus AND Caesar AND NOT Calpurnia, we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND:

110100 AND 110111 AND 101111 = 100100

Answer : The answers for this query are thus Antony and Cleopatra and Hamlet

Let us now consider a more realistic scenario, simultaneously using the opportunity to introduce some terminology and notation. Suppose we have N = 1 million documents. **By documents we mean whatever units we have decided to build a retrieval system over**. They might be individual memos or chapters of a book. We will refer to the group of

documents over which we perform retrieval as the **COLLECTION**. It is sometimes also referred to as a **Corpus.**

we assume an average of **6 bytes per word** including spaces and punctuation, then this is a document collection about 6 GB in size. Typically, there might be about **M = 500,000** distinct terms in these documents. There is nothing special about the numbers we have chosen, and they might vary by an order of magnitude or more, but they give us some idea of the dimensions of the kinds of problems we need to handle.

Advantages:

1. The results of the model are **very predictable** and easy to explain to users.
2. The operands of a Boolean query can be any document feature, not just words, so it is straightforward to incorporate metadata such as a document date or document type in the query specification.
3. From an implementation point of view, **Boolean retrieval is usually more efficient** than ranked retrieval because documents can be rapidly eliminated from consideration in the scoring process

Disadvantages:

1. The major drawback of this approach to search is that the **effectiveness depends entirely on the user**. Because of the lack of a sophisticated ranking algorithm, simple queries will not work well.
2. All documents containing the specified query words will be retrieved, and this retrieved set will be presented to the user in some order, such as by publication date, that has little to do with relevance. It is possible to construct complex Boolean queries that narrow the retrieved set to mostly relevant documents, but this is a difficult task.
3. No ranking of returned result.

Effectiveness of IR:

RELEVANCE: A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need.

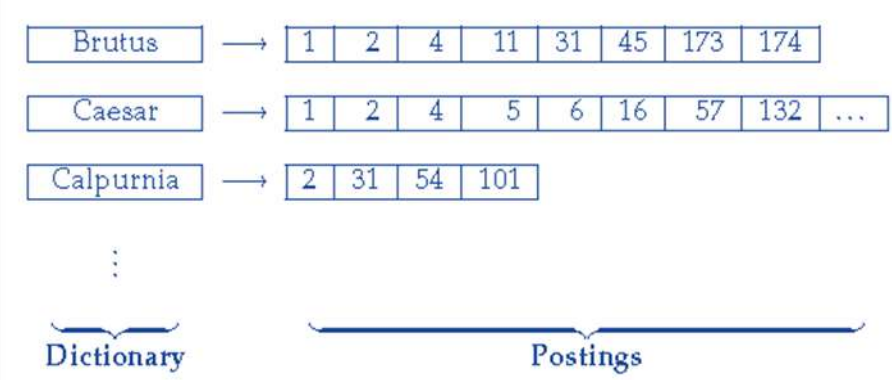
To assess the effectiveness of an IR system (i.e., the quality of its search results), a user will usually want to know two key statistics about the system's returned results for a query:

PRECISION: What fraction of the returned results is relevant to the information need?

RECALL: What fraction of the relevant documents in the collection were returned by the system?

INVERTED INDEX:

Each term in a document is called as index. **Inverted index**, or sometimes **inverted file**, has become the standard term in information retrieval. The basic idea of an inverted index is shown in Figure.



We keep a **dictionary** of terms (sometimes also referred to as a **vocabulary** or**lexicon).** Then for each term, we have a list that records whichdocuments the term occurs in. Each item in the list – which records that aterm appeared in a document (and, later, often, the positions in the document) – is conventionally called a **posting**. The list is then called a **postingslist** (or **inverted list**), and all the postings lists taken together are referred to as the **postings**. The dictionary in above figure has been sorted alphabetically andeach postings list is sorted by document ID.

Building an inverted index:

To gain the speed benefits of indexing at retrieval time, we have to build the index in advance. The major steps in this are:

1. Collect the documents to be indexed:

2. Tokenize the text, turning each document into a list of tokens:

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Within a document collection, we assume that each document has a uniqueserial number, known as the **document identifier (docID)**. During index construction,we can simply assign successive integers to each new documentwhen it is first encountered. The input to indexing is a list of normalizedtokens for each document, which we can equally think of as a list of pairs ofterm and docID, as in following figure.

The core indexing step is **sorting** this list so that the terms are alphabetical, giving us the representation in the middle column of Figure. Multiple occurrences of the same term from the same document are then **merged.** Instances of the same term are then grouped, and the result is split into a dictionary and postings, as shown in the right column of Figure. The dictionary also records some statistics, such as the number of documents which contain each term (the **Document Frequency**, which is here also the length of each postings list). The postings are secondarily sorted by docID.

In the resulting index, we pay for storage of both the dictionary and the postings lists. The latter are much larger, but **the dictionary is commonly kept in memory, while postings lists are normally kept on disk**.

What data structure should be used for a postings list? A fixed length array would be wasteful as some words occur in many documents, and others in very few. For an in-memory postings list, two good alternatives are **singly linked lists or variable length arrays**. Singly linked lists allow cheap insertion of documents into postings lists (following updates, such as when recrawling the web for updated documents), and naturally extend to more

advanced indexing strategies such as skip lists, which require additional pointers. Variable length arrays win in space requirements by avoiding the overhead for pointers and in time requirements because their use of contiguous memory increases speed on modern processors with memory caches. Extra pointers can in practice be encoded into the lists as offsets. If updates are relatively infrequent, variable length arrays will be more compact and faster to traverse. **We can also use a hybrid scheme with a linked list of fixed length arrays for each term.**

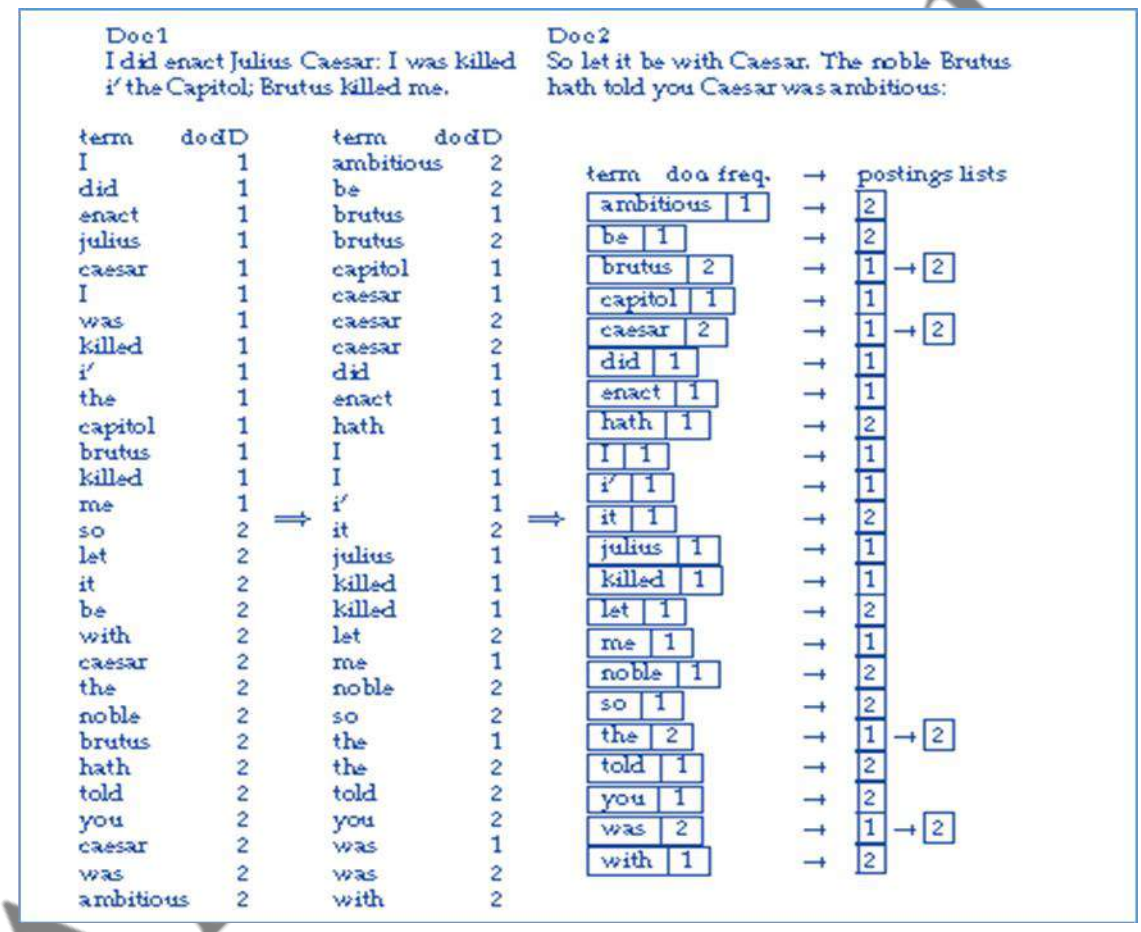


Figure: Building an index by sorting and grouping.

Processing Boolean Queries:

How do we process a query using an inverted index and the basic Booleanretrieval model? Consider processing the **simple conjunctive query**:

Brutus AND Calpurnia

We:

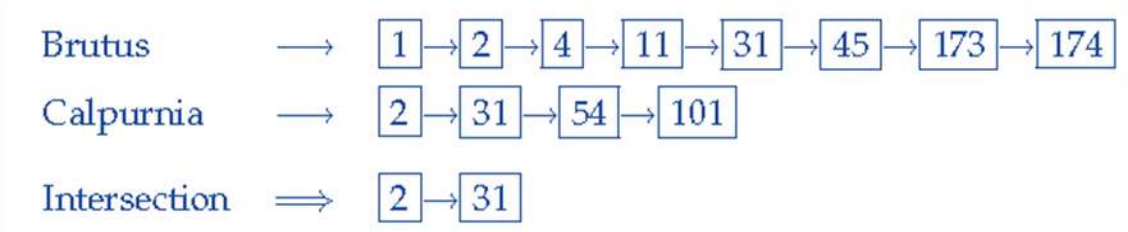
1. Locate Brutus in the Dictionary

2. Retrieve its postings

3. Locate Calpurnia in the Dictionary

4. Retrieve its postings

5. Intersect the two postings lists, as shown in Figure



The **intersection** operation is the crucial one: we need to efficiently **intersect**postings lists so as to be able to quickly find documents that contain bothterms. (This operation is sometimes referred to as **merging postings lists).**

There is a simple and effective method of intersecting postings lists usingthe merge algorithm (see Figure):

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

We maintain pointers into both lists and walk through the two postings lists simultaneously, in time linear inthe total number of postings entries. At each step, we compare the docIDpointed to by both pointers. If they are the same, we put that docID in theresults list, and advance both pointers. Otherwise we advance the pointerpointing to the smaller docID. **If the lengths of the postings lists are x andy, the intersection takes O(x + y)**

operations. Formally, **the complexity ofquerying is Θ(N), where N is the number of documents in the collection**. Our indexing methods gain us just a constant, not a difference in Θ timecomplexity compared to a linear scan, but in practice the constant is huge.

We can extend the intersection operation to process more complicated queries like:

(Brutus OR Caesar) AND NOT Calpurnia

QUERY OPTIMIZATION: Query optimization is the process of selecting how to organize the work of answering a query so that the least total amount of work needs to be done bythe system. A major element of this for Boolean queries is the order in which postings lists are accessed.

What is the best order for query processing? Consider a query:

Brutus AND Caesar AND Calpurnia

For each of the t terms, we need to get its postings, then AND them together. The standard heuristic is to **process terms in order of increasing document frequency:** if we start by intersecting the two smallest postings lists, then all intermediate results must be no bigger than the smallest postings list, and we are therefore likely to do the least amount of total work. We execute the above query as:

(Calpurnia AND Brutus) AND Caesar

The extended Boolean model:

A strict Boolean expression over terms with an unordered results set is too limited for many of the information needs that people have, and these systems implemented extended Boolean retrieval models by incorporating additional operators such as term proximity operators. **A proximity operator** is a way of specifying that two terms in a query must occur close to each other in a document, where closeness may be measured by limiting the allowed number of intervening words or by reference to a structural unit such as a sentence or paragraph.

Text Pre-Processing

Document delineation and character sequence decoding Obtaining the character sequence in a document.

Digital documents that are the input to an indexing process are typically bytes in a file or on a web server. **The first step of processing is to convert this byte sequence into a linear sequence of characters**. For the case of plain English text in ASCII **encoding**, this is trivial. The sequence of characters may be encoded by one of various single byte or multibyte encoding schemes, such as Unicode UTF-8, or various national or vendor-specific standards. We need to determine the correct encoding. But it is often handled by heuristic methods, user selection, or by using provided document metadata. Once the encoding is determined, we decode the byte sequence to a character sequence. The characters may have to be decoded out of some binary representation like Microsoft Word DOC files and/or a compressed format such as zip files.

Again, we must determine the **document format,** and then an appropriate decoder has to be used. Even for plain text documents, additional **decoding** may need to be done. In XML documents, character entities, such as &, need to be decoded to give the correct character, namely & for &. Finally, the textual part of the document may need to be extracted out of other material that will not be processed. This might be the desired handling for XML files, if the markup is going to be ignored; we would almost certainly want to do this with postscript or PDF files.

Choosing a document unit:

The next phase is to determine what the document unit for indexing is. Thus far we have assumed that documents are fixed units for the purposes of indexing. A traditional Unix (mbox-format) email file stores a sequence of email messages (an email folder) in one file, but you might wish to regard each email message as a separate document.

More generally, for very long documents, the issue of INDEXING GRANULARITY arises. For a collection of books, it would usually be a bad idea to index an entire book as a document. A search for Chinese toys might bring up a book that mentions China in the first

chapter and toys in the last chapter, but this does not make it relevant to the query. The problems with large document units can be alleviated by use of explicitor implicit proximity search

Determining the vocabulary of terms:

Tokenization:

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;

Output:

Friends	Romans	Countrymen	lend	Me	your	Ears

A token is an instance of a sequence of characters in some particular document that are grouped TYPE together as a useful semantic unit for processing.

A type is the class of all tokens containing the same character sequence.

A term is a (perhaps normalized) type that is included in the IR system's dictionary

The major question of the tokenization phase is what are the correct tokens to use? In this example, it looks fairly trivial: you chop on whitespace and throw away **punctuation characters.** This is a starting point, but even for English there are a number of tricky cases. For example, what do you do about the various uses of the apostrophe for possession and contractions?

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

For O'Neill, which of the following is the desired tokenization?

neill

oneill

o'neill

o' neill

o neill ?

And for aren't, is it:

aren't

arent

are n't

aren t ?

A simple strategy is to just split on all non-alphanumeric characters, but while o neill looks okay, aren t looks intuitively bad.

These issues of tokenization are language-specific. It thus requires the language of the document to be known. Language identification based on classifiers that use short character subsequence as features is highly effective; most languages have distinctive signature patterns.

In English, hyphenation is used for various purposes ranging from splitting up vowels in words (co-education) to joining nouns as names (Hewlett- Packard) to a copyediting device to show word grouping (the hold-him-backand- drag-him-awaymaneuver). It is easy to feel that the first example should be regarded as one token (and is indeed more commonly written as just coeducation), the last should be separated into words, and that the middle case is unclear. Handling hyphens automatically can thus be complex: it can either be done as a classification problem, or more commonly by some heuristic rules, such as allowing short hyphenated prefixes on words, but not longer hyphenated forms.

German writes compound nouns without spaces (e.g., Computerlinguistik COMPOUNDS 'computational linguistics'; Lebensversicherungsgesellschaftsangestellter 'life insurance company employee'). Retrieval systems for German greatly benefit from the use of a **compound-splitter** module, which is usually implemented by seeing if a word can be subdivided into multiple words that appear in a vocabulary. This phenomenon reaches its limit case with major East Asian Languages (e.g., Chinese, Japanese, Korean, and Thai), where text is written without any spaces between words. One approach here is to perform word segmentation as prior linguistic processing.

Dropping common terms: stop words:

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely.These words are called **STOP WORDS**. The general strategy for determining a stop list is to sort the terms by **collection frequency** (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as STOP LIST a stop list, the members of which are then discarded during indexing.

The phrase query "President of the United States", which contains two stop words, is more precise than President AND "United States". The meaning of flights to London is likely to be lost if the word to is stopped out.

The general trend in IR systems over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever.

(for complete details Refer Chapter 2 in text book 1)

2.3. TERM WEIGHTING:

Each term in a document a weight for thatterm, that depends on the number of occurrences of the term in the document. Assign the weight to be equal to the number of occurrences of term t in document d. This weighting scheme is referred to as ***TERM FREQUENCY*** and is denoted $\mathbf{tf_{t,d}}$, with the subscripts denoting the term(t) and the document (d) in order.

Document frequency: The ***document frequency*** $\boldsymbol{df_t}$, defined to be the number of documents in the collection that contain a **term t**.

Denoting as usual the total number of documents in a collection by N, we define the **inverse document frequency** (idf) of a term t as follows:

$$idf_t = log \frac{N}{df_t}$$

Tf-idf weighting:

We now combine the definitions of term frequency and inverse document frequency, to produce a composite weight for each term in each document.

The tf-idf weighting scheme assigns to term t a weight in document d given by

$$tf - idf_{t,d} = tf_{t,d} \times idf_t$$

Vector Space Model

The vector space model computes a measure of similarity by defining a vector that represents each document, and a vector that represents the query. The model is based on the idea that, in some rough sense, the meaning of a document is conveyed by the words used. If one can represent the words in the document by a vector, it is possible to compare documents with queries to determine how similar their content is. If a query is considered to be like a document, a similarity coefficient (SC) that measures the similarity between a document and a query can be computed. Documents whose content, as measured by the terms in the document, correspond most closely to the content of the query are judged to be the most relevant.

This model involves constructing a vector that represents the terms in the document and another vector that represents the terms in the query. Next, a method must be chosen to measure the closeness of any document vector to the query vector.

One could look at the magnitude of the difference vector between two vectors, but this would tend to make any large document appear to be not relevant to most queries, which typically are short. The traditional method of determining closeness of two vectors is to use the size of the angle between them. This angle is computed by using the inner product (or dot product); however, it is not necessary to use the actual angle.Often the

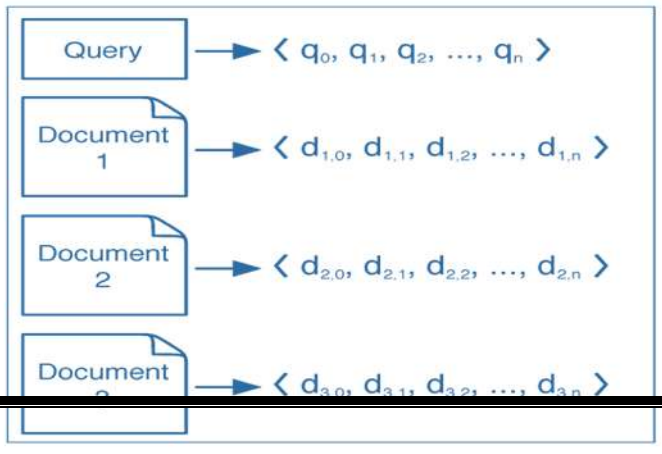
expression "**similarity coefficient"** is used instead of an angle. Computing this number is done in a variety of ways, but the inner product generally plays a prominent role.

There is one component in these vectors for every distinct term or concept that occurs in the document collection. Consider a document collection with only two distinct terms, α and β. All vectors contain only two components, the first component represents occurrences of α, and the second represents occurrences of β. The simplest means of constructing a vector is to place a one in the corresponding vector component if the term appears and a zero if the term does not appear. Consider a document, D1, that contains two occurrences of term α and zero occurrences of term β. The vector < 1,0 > represents this document using a binary representation. This binary representation can be used to produce a similarity coefficient, but it does not take into account the frequency of a term within a document. By extending the representation to include a count of the number of occurrences of the terms in each component, the frequency of the terms can be considered. In this example, the vector would now appear as < 2,0 >.

A simple example is given in the following Figure. A component of each vector is required for each distinct term in the collection. Using the toy example of a language with a two word vocabulary (only A and I are valid terms), all queries and documents can be represented in two dimensional spaces. A query and three documents are given along with their corresponding vectors and a graph of these vectors. The similarity coefficient between the query and the documents can be computed as the distance from the query to the two vectors.

In this example, it can be seen that document one is represented by the same vector as the query so it will have the highest rank in the result set. Instead of simply specifying a list of terms in the query, a user is often given the opportunity to indicate that one term is more important than another. This was done initially with manually assigned term weights selected by users.

Another approach uses automatically assigned weights - typically based on the frequency of a term as it occurs across the entire document collection. The idea was



that a term that occurs infrequently should be given a highereight than a term that occurs frequently. Similarity coefficients that employed automatically assigned weights were compared to manually assigned weights. It was shown that automatically assigned weights perform at least as well as manually assigned weights. Unfortunately, these results did not include the relative weight of the term across the entire collection.

The value of a collection weight was studied in the 1970's. The conclusion was that relevance rankings improved if collection-wide weights were included. Although relatively small document collections were used to conduct the experiments, the authors still concluded that, "in so far as anything can be called a solid result in information retrieval research, this is one".

This more formal definition, and slightly larger example, illustrates the use of weights based on the collection frequency. Weight is computed using the **Inverse Document Frequency (IDF)** corresponding to a given term.

To construct a vector that corresponds to each document, consider the followingdefinitions:

t = number of distinct terms in the document collection

tf_{ij}= number of occurrences of term tj in document Di . (This is referred to as the term frequency.)

df_j = number of documents which 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.)

The vector for each document has n components and contains an entry for each distinct term in the entire document collection. The components in the vector are filled with weights computed for each term in the document collection. The terms in each document are automatically assigned weights based on how frequently they occur in the entire document collection and how often a term appears in a particular document. The weight of

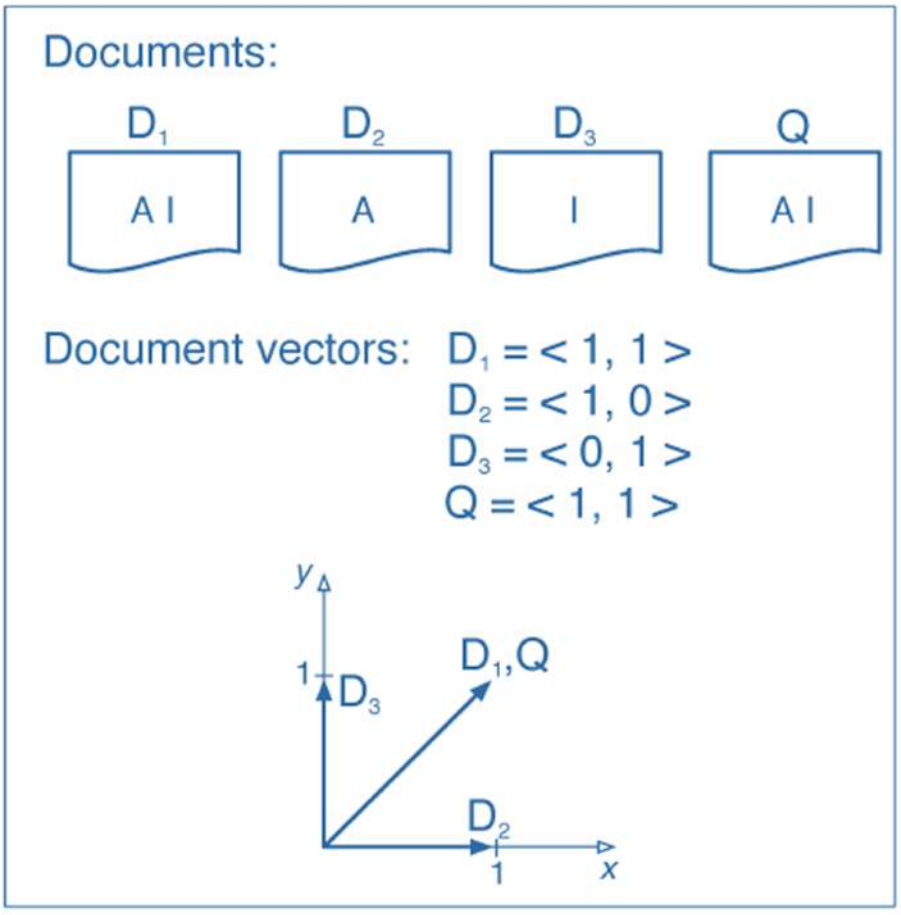
a term in a document increases the more often the term appears in one document and decreases the more often it appears in all other documents.

A weight computed for a term in a document vector is non-zero only if the term appears in the document. For a large document collection consisting of numerous small documents, the document vectors are likely to contain mostly zeros. For example, a document collection with 10,000 distinct terms results in a 1O,000-dimensional vector for each document. A given document that has only 100 distinct terms will have a document vector that contains 9,900zero-valued components.

The weighting factor for a term in a document is defined as a combination of term frequency, and inverse document frequency. That is, to compute the value of the j^{th} entry in the vector corresponding to document i, the followingequation is used:

$$d_{ij} = tf_{ij} \times idf_j$$

Consider a document collection that contains a document, D1 , with ten occurrencesof the term green and a document, D2, with only five occurrences ofthe term green. If green is the only term found in the query, then document D1 is ranked higher than D2 .



When a document retrieval system is used to query a collection of documents with t distinct collection-wide terms, the system computes a vector $D(d_{il}, d_{i2}, \ldots, d_{it})$ of size t for each document. The vectors are filled with term weights

as described above. Similarly, a vector Q (W_{ql}, W_{q2}, ... , W_{qt}) is constructed for the terms found in the query.

A simple similarity coefficient (SC) between a query Q and a document D_i is defined by the dot product of two vectors. Since a query vector is similar in length to a document vector, this same measure is often used to compute the similarity between two documents.

$$SC(Q, D_i) = \sum_{j=1}^{t} w_{qj} \times d_{ij}$$

Example of Similarity Coefficient

Consider a case insensitive query and document collection with a query Q and a document collection consisting of the following three documents:

Q: "gold silver truck"

D1 : "Shipment of gold damaged in a fire"

D2 : "Delivery of silver arrived in a silver truck"

D3: "Shipment of gold arrived in a truck"

In this collection, there are three documents, so d = 3. If a term appears in only one of the three documents, its idf is log d/df_j = log 3/1 = 0.477. Similarly, if a term appears in two of the three documents its idf is log d/df_j = log 3/2 = 0.176, and a term which appears in all three documents has an idf of log d/df_j = log 3/3 = 0.

The idf for the terms in the three documents is given below:

idf_a = 0	$idf_{arrived}$ = 0.176	$idf_{damaged}$ = 0.477
$idf_{delivery}$ = 0.477	idf_{fire} = 0.477	idf_{in} = 0
idf_{of} = 0	idf_{silver} = 0.477	$idf_{shipment}$ = 0.176
$idft_{ruck}$ = 0.176	idf_{gold} = 0.176	

Document vectors can now be constructed. Since eleven terms appear in the document collection, an eleven-dimensional document vector is constructed. The alphabetical ordering given above is used to construct the document vector so that h

corresponds to term number one which is a and t_2 is arrived, etc. The weight for term i in vector j is computed as the idf_i x tf_{ij}. The document vectors are shown in Table.

docid	a	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D_1	0	0	.477	0	.477	.176	0	0	.176	0	0
D_2	0	.176	0	.477	0	0	0	0	0	.954	.176
D_3	0	.176	0	0	0	.176	0	0	.176	0	.176
Q	0	0	0	0	0	.176	0	0	0	.477	.176

SC(Q, D1) = (0)(0) + (0)(0) + (0)(0.477) + (0)(0)+(0)(0.477) + (0.176)(0.176) + (0)(0) + (0)(0)+(0)(0.176) + (0.477)(0) + (0.176)(0)= $(0.176)^2 \approx$ **0.031**

Similarly,

SC(Q, D2) = (0.954)(0.477) + $(0.176)^2 \approx$ **0.486**

SC(Q, D3) = $(0.176)^2$ + $(0.176)^2 \approx$ **0.062**

Hence, the ranking would be D2 , D3 , D1 .

Advantages:

1. Its term-weighting scheme improves retrieval performance.
2. Its partial matching strategy allows retrieval of documents that approximate the query conditions.
3. Its cosine ranking formula sorts the documents according to their degree of similarity to the query.

Disadvantages:

1. The assumption of mutual independence between index terms.
2. It cannot denote the “velar logic view” like Boolean model.

COSINE SIMILARITY

The effect of document length, the standard way of quantifying the similarity between two documents d1 and d2 is to compute the cosine similarity of their vector representations $\vec{v}(d1) and \vec{v}(d2)$

Sim(d1,d2)= $\frac{\vec{v}(d1).\vec{v}(d2)}{|\vec{v}(d1)||\vec{v}(d2)|}$

where the numerator represents the dot product (also known as the inner product) of the vectors $\vec{v}(d1) and \vec{v}(d2)$, while the denominator is the product of their Euclidean lengths. The dot product $\vec{x}.\vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$. Let $\vec{v}(d1)$ denote the document vector for d, with M components $\vec{v}(d1) \ldots . \vec{v}_{\mathrm{M}}(d1)$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{v}_i^2}$. The effect of the denominator of Equation is thus to length-normalize the vectors $\vec{v}(d1) and \vec{v}(d2)$ to unit vectors $\vec{v}(d1)=\vec{v}(d1)/|\vec{v}(d1)|$ and $\vec{v}(d2)=\vec{v}(d2)/|\vec{v}(d2)|$. We can then rewrite as

sim(d_1,d_2)= $\vec{v}(d1).\vec{v}(d2)$

PROBABILISTIC INFORMATION RETRIEVAL:

The probabilistic model computes the similarity coefficient (SC) between a query and a document as the probability that the document will be relevant to the query. This reduces the relevance ranking problem to an application of probability theory. Probability theory can be used to compute a measure of relevance between a query and a document. ***Two fundamentally different approaches were proposed***. The first relies on usage patterns to predict relevance, the second uses each term in the query as clues as to whether or not a document is relevant.

Simple Term Weights:

The use of term weights is based on the Probability Ranking Principle (PRP), which assumes that optimal effectiveness occurs when documents are ranked based on an estimate of the probability of their relevance to a query. **The key is to assign probabilities to components of the query and then use each of these as evidence in computing the final probability that a document is relevant to the query.** The terms in the query are assigned weights which correspond to the probability that a particular term, in a match with a given query, will retrieve are relevant document. The weights for each term in the query are combined to obtain a final measure of relevance.

Suppose we are trying to predict whether or not a softball team called the Salamanders will win one of its games. We might observe, based on past experience, that they usually win on sunny days when their best shortstop plays. This means that two pieces of evidence, outdoor-conditions and presence of good-shortstop, might be used. For any given game, there is a seventy five percent chance that the team will win if the weather is sunny and a sixty percent chance that the team will win if the shortstop plays. Therefore, we write:

P(win I sunny) = 0.75

P(win I good-shortstop) = 0.6

The conditional probability that the team will win given both situations is written as p(win I sunny, good-shortstop). This is read "the probability that the team will win given that there is a sunny day and the good-shortstop plays."We have two pieces of evidence indicating that the Salamanders will win. Intuition says that together the two pieces should be stronger than either alone. This method of combining them is to "look at the odds." A seventy-five percent chance of winning is a twenty-five percent chance of losing, and a sixty percent chance of winning is a forty percent chance of losing. Let us assume the independence of the pieces of evidence.

P(win I sunny, good-shortstop) = α

P(win I sunny) = β

P(win I good-shortstop) = γ

By Bayes' Theorem:

$$\alpha = \frac{P(\mathrm{win}, sunny, good\text{–}shortstop)}{P(sunny, good\text{–}shortstop)} = \frac{P(sunny, good\text{–}shortstop|\mathrm{win})P(\mathrm{win})}{P(sunny, good\text{–}shortstop)}$$

Therefore:

$$\frac{\alpha}{1-\alpha} = \frac{P(sunny, good\text{–}shortstop|\mathrm{win})P(\mathrm{win})}{P(sunny, good\text{–}shortstop|\mathrm{lose})P(\mathrm{lose})}$$

Solving for the first term (because of the independence assumptions):

$$\frac{P(sunny, good\text{–}shortstop|\text{win})}{P(sunny, good\text{–}shortstop|\text{lose})} = \frac{P(sunny|\text{win})P(\text{good–}shortstop|\text{win})}{P(sunny|\text{lose})P(\text{good–}shortstop|\text{lose})}$$

$$\frac{\beta}{1-\beta} = \frac{P(sunny|\text{win})P(\text{win})}{P(sunny|\text{lose})P(\text{lose})}$$

$$\frac{\gamma}{1-\gamma} = \frac{P(\text{good–}shortstop|\text{win})P(\text{win})}{P(\text{good–}shortstop|\text{lose})P(\text{lose})}$$

Similarly,

Making all of the appropriate substitutions, we obtain:

$$\frac{\alpha}{1-\alpha} = \left(\frac{\beta}{1-\beta}\right)\left(\frac{P(\text{lose})}{P(\text{win})}\right)\left(\frac{\gamma}{1-\gamma}\right)\left(\frac{P(\text{lose})}{P(\text{win})}\right)\left(\frac{P(\text{win})}{P(\text{lose})}\right)$$

Simplifying:

$$\frac{\alpha}{1-\alpha} = \left(\frac{\beta}{1-\beta}\right)\left(\frac{\gamma}{1-\gamma}\right)\left(\frac{P(\text{lose})}{P(\text{win})}\right)$$

Assume the Salamanders are a 0.500 ball club (that is they win as often as they lose) and assume numeric values for β and γ of 0.6 and 0.75, respectively. We then obtain:

$$\frac{\alpha}{1-\alpha} = \left(\frac{0.6}{0.4}\right)\left(\frac{0.75}{0.25}\right)\left(\frac{0.500}{0.500}\right) = (1.5)(3.0)(1.0) = 4.5$$

Solving for α gives a value of $\frac{9}{11} = 0.818$.

For an information retrieval query, the terms in the query can be viewed as indicators that a given document is relevant. The presence or absence of query term A can be used to predict whether or not a document is relevant. Hence, after a period of observation, it is found that when term A is in both the query and the document, there is an *x* percent chance the document is relevant. We then assign a probability to term A. Assuming independence of terms; this can be done for each of the terms in the query. Ultimately, the product of all the weights can be used to compute the probability of relevance.

In the following figure, we illustrate the need for training data with most probabilistic models. A query with two terms, *ql* and *q2,* is executed. Five documents are returned and an assessment is made that documents two and four are relevant. From this assessment, the probability that a document is relevant (or non-relevant) given that it contains term *ql* is computed. Likewise, the same probabilities are computed for term *q2.* Clearly, these probabilities are estimates based on training data. The idea is that sufficient training data can be obtained so that when a user issues a query, a good estimate of which documents are relevant to the query can be obtained.

Consider a document, *di,* consisting of t terms *(WI, W2, ... , Wt),* where *Wi* is the estimate that term i will result in this document being relevant. The weight or "odds" that document *di* is relevant is based on the probability of relevance for each term in the document. For a given term in a document, its contribution to the estimate of relevance for the entire document is computed as:

$$\frac{P(w_i|rel)}{P(w_i|nonrel)}$$

The question is then: How do we combine the odds of relevance for each term in to an estimate for the entire document? Given our independence assumptions, we can multiply the odds for each term in a document to obtain the odds that the document is relevant. Taking the log of the product yields:

$$\log\left(\prod_{i=1}^{t}\frac{P(w_i|\ rel)}{P(w_i|\ nonrel)}\right)=\sum_{i=1}^{t}\log\left(\frac{P(w_i|rel)}{P(w_i|nonrel)}\right)$$

The assumption is also made that if one term appears in a document, then it has no impact on whether or not another term will appear in the same document.

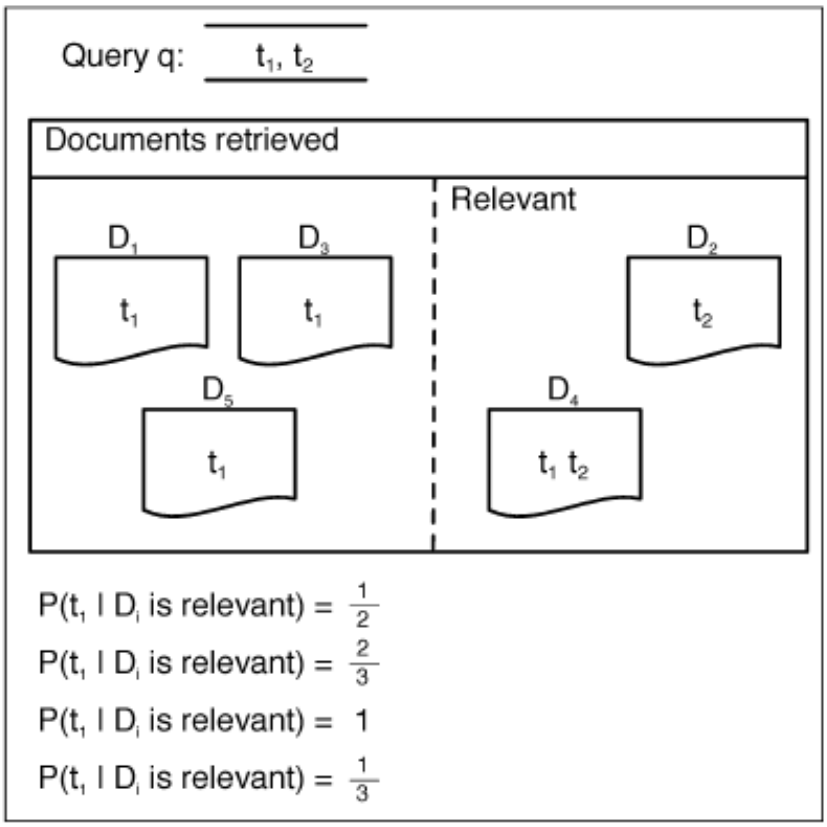


Fig: Training Data for Probabilistic IR

Now that we have described how the individual term estimates can be combined into a total estimate of relevance for the document, it is necessary to describe a means of estimating the individual term weights. Several different means of computing the probability of relevance and non-relevance for a given term were studied since the introduction of the probabilistic retrieval model. **In** their 1976 paper, Robertson and Sparck Jones considered several methods[Robertson and Sparck Jones, 1976]. They began by presenting two mutually exclusive independence assumptions:

11: The distribution of terms in relevant documents is independent and their distribution in all documents is independent.

12: The distribution of terms in relevant documents is independent and their distribution in non-relevant documents is independent.

They also presented two methods, referred to as ordering principles, for presenting the result set:

01: Probable relevance is based only on the presence of search terms in the documents.

02: Probable relevance is based on both the presence of search terms in documents and their absence from documents.

11 indicates that terms occur randomly within a document-that is, the presence of one term in a document in no way impacts the presence of another term in the same document. This is analogous to our example in which the presence of the good-shortstop had no impact on the weather given a win. This also states that the distribution of terms across all documents is independent unconditionally for all documents-that is, the presence of one term in a document in no way impacts the presence of the same term in other documents. This is analogous to saying that the presence of a good-shortstop in one game has no impact on whether or not a good-shortstop will play in any other game.

Similarly, the presence of good-shortstop in one game has no impact on the weatherfor any other game.12 indicates that terms in relevant documents are independent-that is, theysatisfy 11 and terms in non-relevant documents also satisfy 11. Returning to ourexample, this is analogous to saying that the independence of a good-shortstopand sunny weather holds regardless of whether the team wins or loses.01 indicates that documents should be highly ranked only if they containmatching terms in the query (i.e., the only evidence used is which query termsare actually present in the document). We note that this ordering assumption isnot commonly held today because it is also important to consider when queryterms are not found in the document. This is inconvenient in practice. Mostsystems use an inverted index that identifies for each term, all occurrencesof that term in a given document. If absence from a document is required,the index would have to identify all terms *not* in a document. To avoid the need to track theabsence of a term in a document, the estimate makes the zero point correspondto the probability of relevance of a document lacking all the query terms-asopposed to the probability of relevance of a random document. The zero pointdoes not mean that we do not know anything: it simply means that we havesome evidence for non-relevance. This has the effect of converting the 02based weights to presence-only weights.02 takes 01 a little further and says that we should consider both the *presence*and the *absence* of search terms in the query. Hence, for

a query that asksfor term *tl* and term t2-a document with just one of these terms should beranked lower than a document with both terms.

Four weights are then derived based on different combinations of these orderingprinciples and independence assumptions. Given a term, *t,* consider thefollowing quantities:

N =number of documents in the collection

R =number of relevant documents for a given query *q*

n =number of documents that contain term *t*

r =number of relevant documents that contain term *t*

Choosing I1 and O1 yields the following weight:

$$w_1 = \log\left(\frac{\frac{r}{R}}{\frac{n}{N}}\right)$$

Choosing I2 and O1 yields the following weight:

$$w_2 = \log\left(\frac{\frac{r}{R}}{\frac{n-r}{N-R}}\right)$$

Choosing I1 and O2 yields the following weight:

$$w_3 = \log\left(\frac{\frac{r}{R-r}}{\frac{n}{N-n}}\right)$$

Choosing I2 and O2 yields the following weight:

$$w_4 = \log\left(\frac{\frac{r}{R-r}}{\frac{n-r}{(N-n)-(R-r)}}\right)$$

Robertson and Sparck Jones argue that 02 is correct and that 12 is morelikely than 11 to describe what actually occurs. Hence, *W4* is most likely toyield the best results. They then present results that indicate that *W4* and *W3*performed better than *WI* and *W2.* Most subsequent work starts with *W4* andextends it to contain other important components such as the within-documentfrequency of the term and the relative length of a document. When

incomplete relevance information is available, 0.5 is added to theweights to account for the uncertainty involved in estimating relevance. Robertsonand Sparck Jones suggest that, "This procedure may seem somewhat arbitrary,but it does in fact have some statistical justification." The modifiedweighting function appears as:

$$w = \log\left(\frac{\frac{r+0.5}{(R-r)+0.5}}{\frac{(n-r)+0.5}{(N-n)-(R-r)+0.5}}\right)$$

Advantage:

The claimed advantage to the probabilistic model is that it is entirely basedon probability theory. The implication is that other models have a certain arbitrarycharacteristic. They might perform well experimentally, but they lack asound theoretical basis because the parameters are not easy to estimate.

Disadvantage:

1. Theyneed to guess the initial relevant and non-relevant sets.
2. Term frequency is not considered
3. Independence assumption for index terms

Example:

Using the same example we used previously with the vector space model,

we now show how the four different weights can be used for relevance ranking.Again, the documents and the query are:

Q: "gold silver truck"

D1: "Shipment of gold damaged in a fire."

D2: "Delivery of silver arrived in a silver truck."

D3: "Shipment of gold arrived in a truck."

Since training data are needed for the probabilistic model, we assume thatthese three documents are the training data and we deem documents *D2* and*D3* as relevant to the query.To compute the similarity coefficient, we assign term weights to each term

in the query. We then sum the weights of matching terms. There are fourquantities we are interested in:

N=number of documents in the collection

n =number of documents indexed by a given term

R =number of relevant documents for the query

r =number of relevant documents indexed by the given term

Table: Frequencies of each term

	gold	*silver*	*truck*
N	3	3	3
n	2	1	2
R	2	2	2
r	1	1	2

$$w_1 = \log\left[\frac{\frac{r}{R}}{\frac{n}{N}}\right]$$

$$w_2 = \log\left[\frac{\frac{r}{R}}{\frac{(n-r)}{(N-R)}}\right]$$

$$w_3 = \log\left[\frac{\frac{r}{(R-r)}}{\frac{n}{(N-n)}}\right]$$

$$w_4 = \log\left[\frac{\frac{r}{(R-r)}}{\frac{(n-r)}{(N-n)-(R-r)}}\right]$$

Note that with our collection, the weight for *silver* is infinite, since *(n-r)* =O. This is because "silver" only appears in relevant documents. Since weare using this procedure in a predictive manner, Robertson and Sparck Jonesrecommended adding constants to each quantity

[Robertson and Sparck Jones,1976]. The new weights

$$w_1 = \log \left[\frac{\frac{(r+0.5)}{(R+1)}}{\frac{(n+1)}{(N+2)}} \right]$$

$$w_2 = \log \left[\frac{\frac{(r+0.5)}{(R+1)}}{\frac{(n-r+0.5)}{(N-R+1)}} \right]$$

$$w_3 = \log \left[\frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n+1)}{(N-n+1)}} \right]$$

are:

$$w_4 = \log \left[\frac{\frac{(r+0.5)}{(R-r+0.5)}}{\frac{(n-r+0.5)}{(N-n-(R-r)+0.5}} \right]$$

Using these equations, we derive the following weights:

gold

$$w_1 = \log \left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(2+1)}{(3+2)}} \right] = \log \frac{0.5}{0.6} = -0.079$$

silver

$$w_1 = \log \left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1+1)}{(3+2)}} \right] = \log \frac{0.5}{0.4} = 0.097$$

truck

$$w_1 = \log \left[\frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2+1)}{(3+2)}} \right] = \log \frac{0.833}{0.6} = 0.143$$

gold

$$w_2 = \log \left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(2-1+0.5)}{(3-2+1)}} \right] = \log \frac{0.5}{0.75} = -0.176$$

silver

$$w_2 = \log \left[\frac{\frac{(1+0.5)}{(2+1)}}{\frac{(1-1+0.5)}{3-2+1}} \right] = \log \frac{0.5}{0.25} = 0.301$$

truck

$$w_2 = \log \left[\frac{\frac{(2+0.5)}{(2+1)}}{\frac{(2-2+0.5)}{3-2+1}} \right] = \log \frac{0.833}{0.25} = 0.523$$

gold

$$w_3 = \log \left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2+1)}{(3-2+1)}} \right] = \log \frac{1.0}{1.5} = -0.176$$

silver

$$w_3 = \log \left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1+1)}{(3-1+1)}} \right] = \log \frac{1.0}{0.667} = 0.176$$

truck

$$w_3 = \log \left[\frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2+1)}{(3-2+1)}} \right] = \log \frac{5}{1.5} = 0.523$$

gold

$$w_4 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(2-1+0.5)}{(3-2-2+1+0.5)}}\right] = \log\frac{1}{3} = -0.477$$

silver

$$w_4 = \log\left[\frac{\frac{(1+0.5)}{(2-1+0.5)}}{\frac{(1-1+0.5)}{(3-1-2+1+0.5)}}\right] = \log\frac{1}{0.333} = 0.477$$

truck

$$w_4 = \log\left[\frac{\frac{(2+0.5)}{(2-2+0.5)}}{\frac{(2-2+0.5)}{(3-2-2+2+0.5)}}\right] = \log\left(\frac{5}{0.333}\right) = 1.176$$

Result:

Term Weight:

	w_1	w_2	w_3	w_4
gold	-0.079	-0.176	-0.176	-0.477
silver	0.097	0.301	0.176	0.477
truck	0.143	0.523	0.523	1.176

Document Weight:

	w_1	w_2	w_3	w_4
D_1	-0.079	-0.176	-0.176	-0.477
D_2	0.240	0.824	0.699	1.653
D_3	0.064	0.347	0.347	0.699

The similarity coefficient for a given document is obtained by summing theweights of the terms present. Table gives the similarity coefficients for eachof the four different weighting schemes. For *D1, gold* is the only term to appearso the weight for *D1* is just the weight for *gold,* which is -0.079. For *D2 , silver*and *truck* appear so the weight for *D2* is the sum of the weights for *silver* and*truck,* which is 0.097 + 0.143 = 0.240. For *D3 , gold* and

truck appear so theweight for *D3* is the sum for *gold* and *truck,* which is -0.079+0.143 = 0.064.

Language Model based IR:

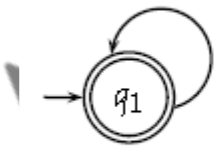
Finite automata and language models:

What do we mean by a document model generating a query? A traditional *generative model* of a language, of the kind familiar from formal language theory, can be used either to recognize or to generate strings. For example, the finite automaton shown in Figure can generate strings that include the examples shown. The full set of strings that can be generated is called the *language* of the automaton

I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
...

CANNOT GENERATE: wish I wish

Figure: A simple finite automaton and some of the strings in the language it generates. → shows the start state of the automaton and a double circle indicates a(possible) finishing state.



$P(\text{STOP}|q_1) = 0.2$

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
...	...

Figure: A one-state finite automaton that acts as a unigram language model. We show a partial specification of the state emission probabilities.

If instead each node has a probability distribution over generating different terms, we have a language model. The notion of a language model is inherently probabilistic. **A**

language model **is a function that puts a probability measure over strings drawn from some vocabulary.** That is, for a language model *M* over an alphabet Σ:

$$\sum_{s \in \Sigma^*} P(s) = 1$$

One simple kind of language model is equivalent to a probabilistic finite automaton consisting of just a single node with a single probability distribution over producing different terms, so that Σ$t \in V$ $P(t)$ = 1, as shown in Figure. After generating each word, we decide whether to stop or to loop around and then produce another word, and so the model also requires a probability of stopping in the finishing state. Such a model places a probability distribution over any sequence of words. By construction, it also provides a model for generating text according to its distribution.

Types of language models:

The simplest form of language model simply throws away all conditioning context, and estimates each term independently. Such a model is called a *unigram language model*:

$P\text{uni}(t1t2t3t4) = P(t1)P(t2)P(t3)P(t4)$

There are many more complex kinds of language models, such as *bigram language models*, which condition on the previous term,

$P\text{bi}(t1t2t3t4) = P(t1)P(t2|t1)P(t3|t2)P(t4|t3)$

and even more complex grammar-based language models such as probabilistic context-free grammars. Such models are vital for tasks like speech recognition, spelling correction, and machine translation, where you need the probability of a term conditioned on surrounding context. However, most language-modeling work in IR has used unigram language models.

Multinomial distributions over words:

Under the unigram language model the order of words is irrelevant, and so such models are often called "bag of words" models. Even though there is no conditioning on preceding context, this model nevertheless still gives the probability of a particular ordering of terms. However, any other ordering of this bag of terms will have the same probability. So, really, we have a ***multinomial distribution*** over words. So long as we stick to unigram

models, the language model name and motivation could be viewed as historical rather than necessary. We could instead just refer to the model as a multinomial model. From this perspective, the equationspresented above do not present the multinomial probability of a bag ofwords, since they do not sum over all possible orderings of those words, asis done by the multinomial coefficient (the first term on the right-hand side)in the standard presentation of a multinomial model:

$$P(d) = \frac{L_d!}{\mathrm{tf}_{t_1,d}!\mathrm{tf}_{t_2,d}!\cdots\mathrm{tf}_{t_M,d}!}P(t_1)^{\mathrm{tf}_{t_1,d}}P(t_2)^{\mathrm{tf}_{t_2,d}}\cdots P(t_M)^{\mathrm{tf}_{t_M,d}}$$

Here $L_d = \sum_{1<i<M}\mathrm{tf}_{t_i,d}$ is the length of document *d*, *M* is the size of the termvocabulary, and the products are now over the terms in the vocabulary, notthe positions in the document. However, just as with STOP probabilities, inpracticewe can also leave out the multinomial coefficient in our calculations,since, for a particular bag of words, itwill be a constant, and so it has no effecton the likelihood ratio of two differentmodels generating a particular bag ofwords.

The fundamental problem in designing language models is that we do notknow what exactly we should use as the model M_d. However, we do generallyhave a sample of text that is representative of that model. This problemmakes a lot of sense in the original, primary uses of languagemodels. For example,in speech recognition, we have a training sample of (spoken) text.

Latent Semantic Indexing:

Matrix computation is used as a basis for information retrieval in the retrievalstrategy called Latent Semantic Indexing.Thepremise is that more conventional retrieval strategies (i.e., vector space, probabilisticand extended Boolean) all have problems because they match directlyon keywords. Since the same concept can be described using many differentkeywords, this type of matching is prone to failure. The authors cite a studyin which two people used the same word for the same concept only twentypercent of the time.

Searching for something that is closer to representing the underlying semanticsof a document is not a new goal. Canonical forms were proposed fornatural language processing. Applied here, the idea is not to find a canonical knowledge representation,but to use matrix computation, in particular **Singular Value Decomposition (SVD).** This filters out the noise found in a document, such that twodocuments that have the same semantics (whether or not they have matchingterms) will be located close to one another in a multi-dimensional space.

The process is relatively straightforward. A term-document matrix A is constructedsuch that location (i, j) indicates the number of times term i appearsin document j. A SVD of this matrix results in matrices $U\ \Sigma V^T$such that Σis a diagonal matrix. *A* is a matrix that represents each term in a row. Eachcolumn of *A* represents documents. The values in Σare referred to as thesingular values. The singular values can then be sorted by magnitude and thetop *k* values are selected as a means of developing a "latent semantic" representationof the *A* matrix. The remaining singular values are then set to O.Only the first *k* columns are kept in U_k; only the first *k* rows are recorded inV_K^T. After setting the results to 0, a new *A*′ matrix is generated to approximate$A= U\Sigma V^T$

Comparison of two terms is done via an inner product of the two correspondingrows in U_k. Comparison of two documents is done as an inner product oftwo corresponding rows in V_K^T.

A query-document similarity coefficient treats the query as a document andcomputes the SVD. However, the SVD is computationally expensive; so, it isnot recommended that this be done as a solution. Techniques that approximateΣ and avoid the overhead of the SVD exist. For an infrequently updateddocument collection, it is often pragmatic to periodically compute the SVD.

Example:

Q: "gold silver truck"

Dl "Shipment of gold damaged in a fire."

D2: "Delivery of silver arrived in a silver truck."

D3: "Shipment of gold arrived in a truck."

The *A* matrix is obtained from the numeric columns in the term-documenttable given below:

	D_1	D_2	D_3
a	1	1	1
arrived	0	1	1
damaged	1	0	0
delivery	0	1	0
fire	1	0	0
gold	1	0	1
in	1	1	1
of	1	1	1
shipment	1	0	1
silver	0	2	0
truck	0	1	1

This step computes the singular value decompositions (SVD) on *A.* This resultsin an expression of *A* as the product of $U\Sigma V^T$. In our example, *A* isequal to the product of:

$$\begin{bmatrix} -0.4201 & 0.0748 & -0.0460 \\ -0.2995 & -0.2001 & 0.4078 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.1576 & -0.3046 & -0.2006 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.3151 & -0.6093 & -0.4013 \\ -0.2995 & -0.2001 & 0.4078 \end{bmatrix} \begin{bmatrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2737 \end{bmatrix} \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & -0.2469 \\ -0.5780 & -0.2556 & 0.7750 \end{bmatrix}$$

However, it is not the intent to reproduce *A* exactly. What is desired, is tofind the best rank *k* approximation of *A.* We only want the largest *k* singularvalues *(k* < 3). The choice of *k* and the number of singular values in Σ touse is somewhat arbitrary. For our example, we choose *k* = 2. We now have$A2 = U_2\Sigma_2V_2^T$. Essentially, we take only the first two columns of *U* and thefirst two rows of Σ and V^T .*This new product is:*

$$\begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} 4.0989 & 0 \\ 0 & 2.3616 \end{bmatrix} \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & -0.2469 \end{bmatrix}$$

To obtain a $k \times 1$ dimensional array, we now incorporate the query. The query vector q^T is constructed in the same manner as the original A matrix. The query vector is now mapped into a 2-space by the transformation $q^T U_2 \sum_2^{-1}$.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} 0.2440 & 0 \\ 0 & 0.4234 \end{bmatrix} = \begin{bmatrix} -0.2140 & -0.1821 \end{bmatrix}$$

We could use the same transformation to map our document vectors into 2-space, but the rows of V2 contain the co-ordinates of the documents. Therefore:

D1 = (-0.4945 -0.0688)

D2 = (-0.64580.9417)

D3 = (-0.58171.2976)

Finally, we are ready to compute our relevance value using the cosine similaritycoefficient. This yields the following:

$$D_1 = \frac{(-0.2140)(-0.4945) + (-0.1821)(0.6492)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.4945)^2 + (0.6492)^2)}} = -0.0541$$

$$D_2 = \frac{(-0.2140)(-0.6458) + (-0.1821)(-0.7194)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.6458)^2 + (-0.7194)^2)}} = 0.9910$$

$$D_3 = \frac{(-0.2140)(-0.5817) + (-0.1821)(-0.2469)}{\sqrt{(-0.2140)^2 + (-0.1821)^2)}\sqrt{(-0.5817)^2 + (-0.2469)^2)}} = 0.9543$$

Relevance feedback and query expansion:

Relevance feedback is a query expansion and refinement technique with a long history. First proposed in the 1960s, it relies on user interaction to identify relevant documents in a ranking based on the initial query. Other semi-automatic techniques were discussed in the last section, but instead of choosing from lists of terms or alternative queries, in relevance feedback the user indicates which documents are interesting (i.e., relevant) and possibly which documents are completely off-topic (i.e., non-relevant). Based on this information, the system automatically reformulates the query by adding terms and reweighting the original terms, and a new ranking is generated using this modified query.

This process is a simple example of using *machine learning* in information retrieval, where *training data* (the identified relevant and non-relevant documents) is used to improve the system's performance. Modifying the query is in fact equivalent to learning a classifier that distinguishes between relevant and non-relevant documents.

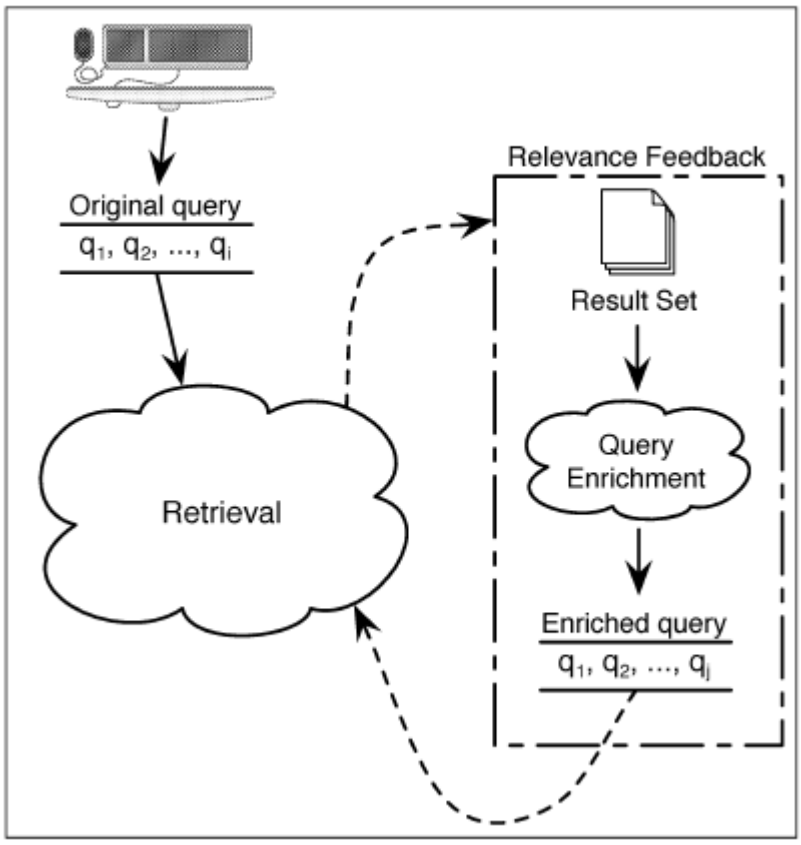


Fig: Relevance Feedback Process

For example, an initial query "find information surrounding the various conspiracy theories about the assassination of John F. Kennedy" has both useful keywords and noise. The most useful keywords are probably *assassination.* Like many queries (in terms of retrieval) there is some meaningless information. Terms such as *various* and *information* are probably not stop words (i.e., frequently used words that are typically ignored by an information retrieval system such as *a, an, and, the),* but they are more than likely not going to help retrieve relevant documents. The idea is to use all terms in the initial query and ask the user if the top ranked documents are relevant. The hope is that the terms in the top ranked documents that are said to be relevant will be "good" terms to use in a subsequent query.

Assume a highly ranked document contains the term *Oswald.* It is reasonable to expect that adding the term *Oswald* to the initial query would improve both precision and recall. Similarly, if a top ranked document that is deemed relevant by the user contains

many occurrences of the term *assassination,* the weight used in the initial query for this term should be increased.

With the vector space model, the addition of new terms to the original query, the deletion of terms from the query, and the modification of existing term weights has been done. With the probabilistic model, relevance feedback initially was only able to re-weight existing terms, and there was no acceptedmeans of adding terms to the original query. The exact means by which relevancefeedback is implemented is fairly dependent on the retrieval strategybeing employed.

Relevance Feedback in the Vector Space Model: /(The Rocchio algorithm for relevance feedback):

Rocchio's approach used the vector space model to rank documents. Thequery is represented by a vector *Q,* each document is represented by a vector*Di,* and a measure of relevance between the query and the document vector iscomputed as *SC(Q, Di),* where SC is the similarity coefficient. The SC is computed as an inner product of the document andquery vector or the cosine of the angle between the two vectors. The basicassumption is that the user has issued a query *Q* and retrieved a set of documents.The user is then asked whether or not the documents are relevant. Afterthe user responds, the set R contains the nl relevant document vectors, and theset S contains the *n2* non-relevant document vectors. Rocchio builds the newquery *Q'* from the old query *Q* using the equation given below:

$$Q' = Q + \frac{1}{n_1}\sum_{i=1}^{n_1} R_i - \frac{1}{n_2}\sum_{i=1}^{n_2} S_i$$

Ri and *Si* are individual components of R and S, respectively.The document vectors from the relevant documents are added to the initialquery vector, and the vectors from the non-relevant documents are subtracted.If all documents are relevant, the third term does not appear. To ensure thatthe new information does not completely override the original query, all vectormodifications are normalized by the number of relevant and non-relevant documents.The process can be repeated such that *Qi+1* is derived from *Qi* for asmany iterations as desired.

The idea is that the relevant documents have terms matching those in theoriginal query. The weights corresponding to these terms are increased byadding the relevant document vector. Terms in the query that are in the nonrelevantdocuments have their weights decreased. Also, terms that are not inthe original query (had an initial component value of zero) are now added tothe original query.

In addition to using values *n1* and *n2,* it is possible to use arbitrary weights.The equation now becomes:

$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2}$$

Not all of the relevant or non-relevant documents must be used. Addingthresholds *na* and *nb* to indicate the thresholds for relevant and non-relevantvectors results in:

$$Q' = \alpha Q + \beta \sum_{i=1}^{min(n_a,n_1)} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{min(n_b,n_2)} \frac{S_i}{n_2}$$

The weights α,β and γ and, are referred to as Rocchio weights and are frequentlymentioned in the annual proceedings of TREC. The optimal values were experimentallyobtained, but it is considered common today to drop the use of nonrelevantdocuments (assign zero to γ) and only use the relevant documents.This basic theme was used by Ide in follow-up research to Rocchio where thefollowing equation was defined:

$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} R_i - S_1$$

Only the top ranked non-relevant document is used, instead of the sum of allnon-relevant documents. Ide refers to this as the *Dec-Hi* (decrease using highestranking non-relevant document) approach. Also, a more simplistic weightis described in which the

normalization, based on the number of documentvectors is removed, and α,β and γ are set to one [Salton, 1971a]. This newequation is:

$$Q' = Q + \sum_{i=1}^{n_1} R_i - \sum_{i=1}^{n_2} S_i$$

An interesting case occurs when the original query retrieves only non-relevantdocuments. Kelly addresses this case in [Salton, 1971b]. The approach suggeststhat an arbitrary weight should be added to the most frequently occurring*concept* in the document collection. This can be generalized to increase thecomponent with the highest weight. The hope is that the term was important,but it was drowned out by all of the surrounding noise. By increasing theweight, the term now rings true and yields some relevant documents. Note thatthis approach is applied only in manual relevance feedback approaches. It isnot applicable to automatic feedback as the top *n* documents are assumed, bydefinition, to be relevant.

UNIT III TEXT CLASSIFICATION AND CLUSTERING

A Characterization of Text Classification – Unsupervised Algorithms: Clustering – Naïve Text Classification – Supervised Algorithms – Decision Tree – k-NN Classifier – SVM Classifier – Feature Selection or Dimensionality Reduction – Evaluation metrics – Accuracy and Error – Organizing the classes – Indexing and Searching – Inverted Indexes – Sequential Searching – Multi-dimensional Indexing

3.1. Web search overview:

- The IR system contains a collection of documents, with each document represented by a sequence of tokens. Markup may indicate titles, authors, and other structural elements.
- Consider IR in the specific context of Web search. Assuming this specific context provides us with the benefit of document features that cannot be assumed in the generic context. One of the most important of these features is the structure supplied by hyperlinks. These links from one page to another, often labeled by an image or anchor text, provide us with valuable information concerning both the individual pages and the relationship between them.
- Along with these benefits come various problems, primarily associated with the relative "quality", "authority" or "popularity" of Web pages and sites, which can range from the carefully edited pages of a major international news agency to the personal pages of a high school student.
- Many Web pages are actually *spam* — malicious pages deliberately posing as something that they are not in order to attract unwarranted attention of a commercial or other nature.
- Although the owners of most Web sites wish to enjoy a high ranking from the major commercial search engines, and may take whatever steps are available to maximize their ranking, the creators of spam pages are in an adversarial relationship with the search engine's operators. The creators of these pages may actively attempt to subvert the features used for ranking, by presenting a false impression of content and quality.
- Other problems derive from the scale of the Web — billions of pages scattered among millions of hosts. In order to index these pages, they must be gathered from across the Web by a *crawler* and stored locally by the search engine for processing.
- Because many pages may change daily or hourly, this snapshot of the Web must be refreshed on a regular basis. While gathering data the crawler may detect duplicates and near-duplicates of pages, which must be dealt with appropriately. For example, the standard documentation for the Java programming language may be found on many Web sites, but in response to a query such as "java", "vector", "class" it might be best for a search engine to return only the official version on the java.sun.com site.
- Another consideration is the volume and variety of queries commercial Web search engines receive, which direct reflect the volume and variety of information on the Web itself. Queries are often short — one or two terms — and the search engine may know little or nothing about the user entering a query or the context of the user's search.
- A user entering the query "UPS" may be interested in tracking a package sent by the courier service, purchasing a universal power supply, or attending night classes at the University of Puget Sound. Although such query ambiguity is a consideration in all IR applications, it reaches an extreme level in Web IR.

3.2. Structure of the Web

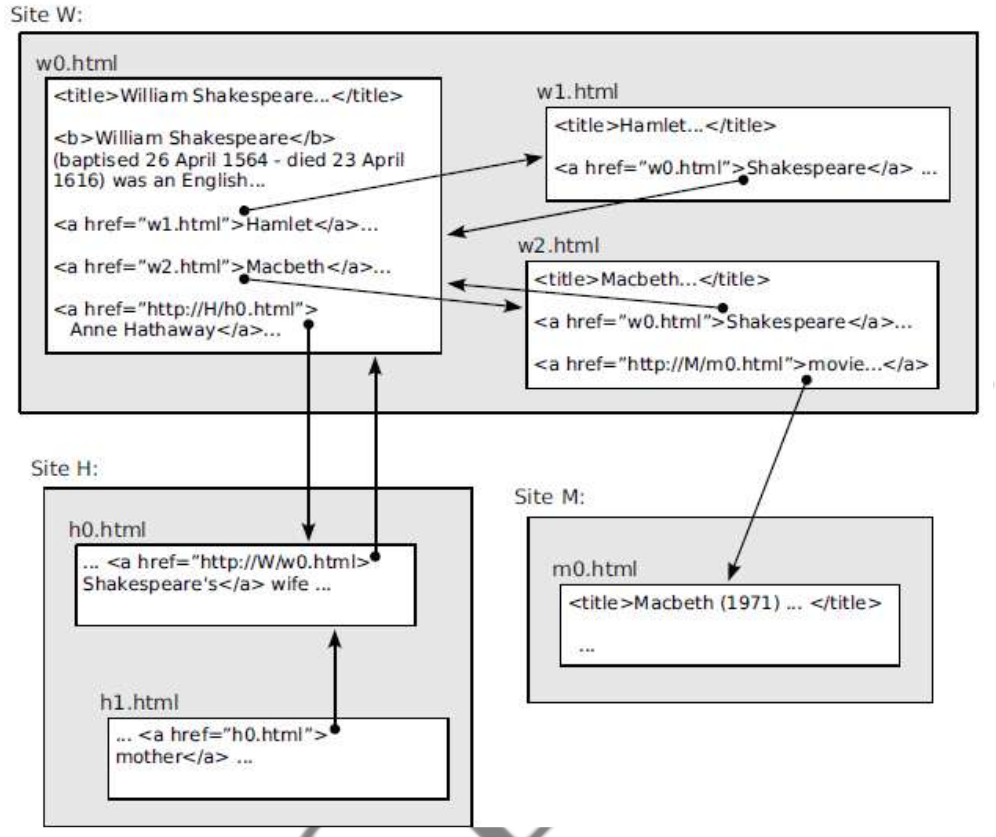


Figure. Structure on the Web. Pages can link to each other (<a href=...>) and can associate each link with anchor text (e.g., "mother").

- Figure provides an example of the most important features related to the structure of the Web. It shows Web pages on three sites: W, M, and H. Site W provides a general encyclopedia including pages on Shakespeare (w0.html) and two of his plays, *Hamlet* (w1.html) and *Macbeth* (w2.html). Site H provides historical information including information on Shakespeare's wife (h0.html) and son (h1.html). Site M provides movie and TV information including a page on the 1971 movie version of *Macbeth* directed by Roman Polanski (m0.html).
- The figure illustrates the link structure existing between these pages and sites. For example, the HTML *anchor* on page w0.html

  Anne Hathaway
- Establishes a link between that page and h0.html on site H. The anchor text associated with this link ("Anne Hathaway") provides an indication of the content on the target page.

The Web Graph:

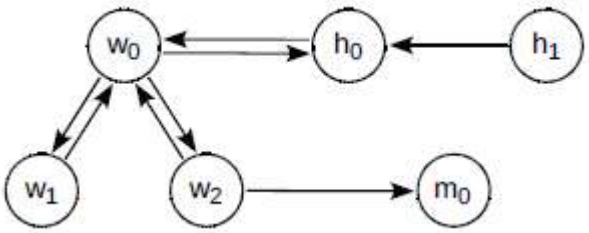


Figure. A Web graph. Each link corresponds to a directed edge in the graph.

- The relationship between sites and pages indicated by these hyperlinks gives rise to what is called a *Web graph*. When it is viewed as a purely mathematical object, each page forms a node in this graph and each hyperlink forms a directed edge from one node to another. For convenience we have simplified the labels on the pages, with http://W/w0.html becoming w0, for example.
- From a practical standpoint it is important to remember that when we refer to a Web graph we are referring to more than just this mathematical abstraction — that pages are grouped into sites, and that anchor text and images may be associated with each link. Links may reference a labeled position within a page as well as the page as a whole.
- The highly dynamic and fluid nature of the Web must also be remembered. Pages and links are continuously added and removed at sites around the world; it is never possible to capture more than a rough approximation of the entire Web graph.
- At times our interest may be restricted to a subset of the Web, perhaps to all the pages from a single site or organization. Under these circumstances, capturing an accurate snapshot of the Web graph for that site or organization is easier than for the entire Web. Nonetheless, most Web sites of any size grow and change on a continuous basis, and must be regularly recrawled for the snapshot to remain accurate.

- We now introduce a formal notation for Web graphs that we use in later sections. Let α be the set of all pages in a Web graph, let N = |α| be the number of pages, and E the number of links (or edges) in the graph. Given a page α ∈ _, we define *out(α)* as the number of *out-links* from α to other pages (the *out-degree*). Similarly, we define *in(α)* as the number of *in-links* from other pages to α (the *in-degree*). In the Web graph of Figure , *out(*w0*)* = 3 and *in(*h0*)* = 2.
- If *in(α)* = 0, then α is called a *source*; if *out(α)* = 0, it is called a *sink*. We define I' as the set of sinks. In the Web graph of Figure, page m0 is a sink and page h1 is a source.
- The individual Web pages themselves may be highly complex and structured objects. They often include menus, images, and advertising. Scripts may be used to generate content when the page is first loaded and to update the content as the user interacts with it. A page may serve as a frame to hold other pages or may simply redirect the browser to another page. These redirections may be implemented through scripts or through special HTTP responses, and may be repeated multiple times, thus adding further complexity. Along with HTML pages the Web includes pages in PDF, Microsoft Word, and many other formats.

Static and Dynamic Pages:

- It is not uncommon to hear Web pages described as "static" or "dynamic". The HTML for a static Web page is assumed to be generated in advance of any request, placed on disk, and transferred to the browser or Web crawler on demand. The home page of an organization is a typical example of a static page.
- A dynamic Web page is assumed to be generated at the time the request is made, with the contents of the page partially determined by the details of the request. A search engine result page (SERP) is a typical example of a dynamic Web page for which the user's query itself helps to determine the content.
- There is often the implication in this dichotomy that static Web pages are more important for crawling and indexing than dynamic Web pages, and it is certainly the case that many types of dynamic Web pages are not suitable for crawling and indexing.
- For example, on-line calendar systems, which maintain appointments and schedules for people and events, will often serve up dynamically generated pages for dates far into the past and the future.
- You can reach the page for any date if you follow the links far enough. Although the flexibility to book appointments 25 years from now is appropriate for an online calendar; indexing an endless series of empty months is not appropriate for a general Web search engine.
- Nonetheless, although many dynamic Web pages should not be crawled and indexed, many should. For example, catalog pages on retail sites are often generated dynamically by accessing current products and prices in a relational database. The result is then formatted into HTML and wrapped with menus and other fixed information for display in a browser. To meet the needs of a consumer searching for a product, a search engine must crawl and index these pages.
- A dynamic page can sometimes be identified by features of its URL. For example, servers accessed through the Common Gateway Interface (CGI) may contain the path element cgi-bin in their URLs.
- Pages dynamically generated by Microsoft's Active Server Pages technology include the extension .asp or .aspx in their URLs. Unfortunately, these URL features are not always present.
- In principle any Web page can be static or dynamic, and there is no sure way to tell. For crawling and indexing, it is the content that is important, not the static or dynamic nature of the page.

The Hidden Web:

- Many pages are part of the so-called "hidden" or "invisible" or "deep" Web. This hidden Web includes pages that have no links referencing them, those that are protected by passwords, and those that are available only by querying a digital library or database. Although these pages can contain valuable content, they are difficult or impossible for a Web crawler to locate.
- Pages in *intranets* represent a special case of the hidden Web. Pages in a given intranet are accessible only within a corporation or similar entity.
- An enterprise search engine that indexes an intranet may incorporate many of the same retrieval techniques that are found in general Web search engines, but may be tuned to exploit specific features of that intranet.

3.3. QUERIES AND USERS:

- Web queries are short. Although the exact numbers differ from study to study, these studies consistently reveal that many queries are just one or two terms long, with a mean query length between two and three terms. The topics of these queries range across the full breadth of human interests, with sex, health, commerce, and entertainment representing some of the major themes.
- Perhaps it is not surprising that Web queries are short. Until quite recently, the query processing strategies of Web search engines actually discouraged longer queries.
- These processing strategies filter the collection against a Boolean conjunction of the query terms prior to ranking. For a page to be included in the result set, all query terms must be associated with it in some way, either by appearing on the page itself or by appearing in the anchor text of links referencing it.
- As a result, increasing the length of the query by adding related terms could cause relevant pages to be excluded if they are missing one or more of the added terms. This strict filtering has been relaxed in recent years.
- For example, synonyms may be accepted in place of exact matches — the term "howto" might be accepted in place of the query term "FAQ". Nonetheless, for efficiency reasons filtering still plays a role in query processing, and Web search engines may still perform poorly on longer queries.
- The distribution of Web queries follows Zipf's law.

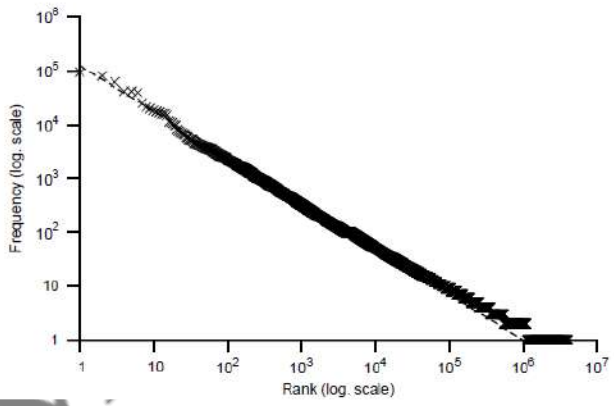


Figure. Frequency of queries by rank order, based on 10 million queries taken from the logs of a commercial search engine. The dashed line corresponds to Zipf's law with $\alpha = 0.85$.

- In a representative log of 10 million queries, the single most frequent query may represent more than 1% of the total, but nearly half of the queries will occur only once. The tuning and evaluation of a search engine must consider this "long tail" of Zipf's law. In the aggregate, infrequent queries are at least as important as the frequent ones.

User Intent:

Web queries are classified into three categories reflecting users' apparent intent, as follows:

1) *Navigational* query:

- The intent behind a *navigational* query is to locate a specific page or site on the Web. For example, a user intending to locate the home page of the CNN news network might enter the query "CNN". A navigational query usually has a single correct result. However, this correct result may vary from user to user.

2) *Informational* query:

- A user issuing an *informational* query is interested in learning something about a particular topic and has a lesser regard for the source of the information, provided it is reliable.
- A user entering the query "president", "obama" might find the information she is seeking on the CNN Web site, on Wikipedia, or elsewhere. Perhaps she will browse and read a combination of these before finding all the information she requires.
- The need behind an informational query such as this one may vary from user to user, and may be broad or narrow in scope. A user may be seeking a detailed description of government policy, a short biography, or just a birth date.

3) *Transactional* query:

- A user issuing a *transactional* query intends to interact with a Web site once she finds it. This interaction may involve activities such as playing games, purchasing items, booking travel, or downloading images, music, and videos. This category may also include queries seeking services such as maps and weather, provided the user is not searching for a specific site providing that service.

- Under both the informational and the transactional categories they identified a number of subcategories. For example, the goal of a user issuing a *directed informational* query is the answer to a particular question ("When was President Obama born?"), whereas the goal of a user issuing an *undirected informational* query is simply to learn about the topic ("Tell me about President Obama."). In turn, directed informational queries may be classified as being *open* or *closed*, depending on whether the question is open-ended or has a specific answer.
- The distinction between navigational and informational queries is relatively straightforward:
- Is the user seeking a specific site or not? The distinction between transactional queries and the other two categories is not always as clear. We can assume the query "mapquest" is navigational, seeking the site www.mapquest.com, but it is likely that the user will then interact with the site to obtain directions and maps.
- A user entering the query "travel", "washington" may be seeking both tourist information about Washington, D.C., and planning to book a hotel, making the intent both informational and transactional. In cases such as these, it may be reasonable to view such queries as falling under a combination of categories, as navigational/transactional and informational/transactional, respectively.
- According to Broder (2002), navigational queries comprise 20–25% of all Web queries. Of the remainder, transactional queries comprise at least 22%, and the rest are informational queries. According to Rose and Levinson (2004), 12–15% of queries are navigational and 24– 27% are transactional.

- A more recent study (2007) slightly contradicts these numbers. Their results indicate that more than 80% of queries are informational, with the remaining queries split roughly equally between the navigational and transactional categories.
- Nonetheless, these studies show that all three categories represent a substantial fraction of Web queries and suggest that Web search engines must be explicitly aware of the differences in user intent.
- Referring to "navigational queries" and "informational queries" is common jargon. This usage is understandable when the goal underlying a query is the same, or similar, regardless of the user issuing it.
- But for some queries the category may differ from user to user. As a result we stress that these query categories fundamentally describe the goals and intentions of the users issuing the queries, and are not inherent properties of the queries themselves. For example, a user issuing our example query "UPS" may have

✓ Informational intent, wanting to know how universal power supplies work
✓ Transactional intent, wanting to purchase an inexpensive UPS for a personal computer
✓ Transactional/navigational intent, wanting to track a package or
✓ Navigational/informational intent, wanting information on programs offered by the University of Puget Sound.

- Although this query is atypical, falling into so many possible categories, the assignment of any given query to a category (by anyone other than the user) may be little more than an educated guess.

Clickthrough Curves:

- The distinction between navigational and informational queries is visible in user behavior. *clickthroughs* may be used to infer user intent. A clickthrough is the act of clicking on a search result on a search engine result page. Clickthroughs are often logged by commercial search engines as a method for measuring performance.
- Although almost all clicks occur on the top ten results, the pattern of clicks varies from query to query. By examining clickthroughs from a large number of users issuing the same query, you can identified clickthrough distributions that are typical of informational and navigational queries. For navigational queries the clickthroughs are skewed toward a single result; for informational queries the clickthrough distribution is flatter.

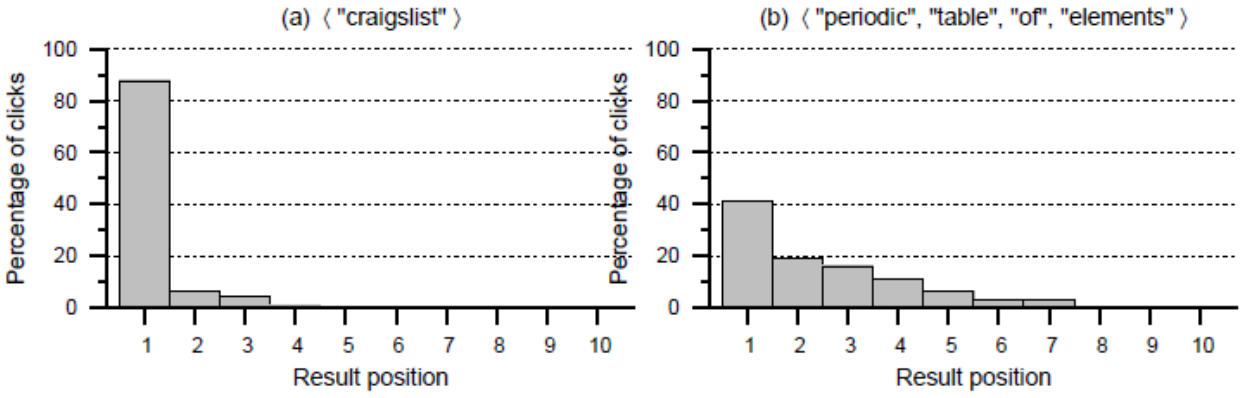


Fig. Clickthrough curve for a typical navigational query ("craigslist") and a typical informational query ("periodic", "table", "of", "elements").

- Both plots show the percentage of clickthroughs for results ranked 1 to 10. Each clickthrough represents the first result clicked by a different user entering the query. Figure (a) shows a clickthrough distribution for a stereotypical navigational query, exhibiting a spike at www.craigslist.org, a classified advertising site and the presumed target of the query. Figure (b) shows a clickthrough distribution for a typical informational query. For both queries the number of clickthroughs decreases with rank; the informational query receives proportionally more clicks at lower ranks.

3.4 SPONSORED SEARCH AND PAID PLACEMENT

- A far more effective and profitable form of advertising for search engines, pioneered by GoTo.com (that was renamed to Overture in 2001 and acquired by Yahoo in 2003), is called *paid placement* also known as *sponsored search*.

In this scheme the search engine separates its query results list into two parts:

(i) An organic list, which contains the free unbiased results, displayed according to the search engine's ranking algorithm, and

(ii) A sponsored list, which is paid for by advertising managed with the aid of an online auction mechanism. This method of payment is called *pay per click* (PPC), also known as *cost per click* (CPC), since payment is made by the advertiser each time a user clicks on the link in the sponsored listing.

- In most cases the organic and sponsored lists are kept separate but an alternative model is to interleave the organic and sponsored results within a single listing. (Most notably the metasearch engine Dogpile (www.dogpile.com) combines its organic and sponsored lists).
- The links in the sponsored list are ranked according to the product of the highest bid for the keywords associated with the query and the *quality score*. The quality score is determined by a weighted combination of the Click Through Rate(CTR) of the ad, its quality as determined by the relevancy of the keywords bid to the actual query, and the quality of the landing page behind the sponsored link. Although the weights of the quality features have not been disclosed by the search engines running the online auctions, the CTR is the major factor in the formula for computing the quality score.
- In the context of CTRs it is worth mentioning that query popularity gives rise to a long tail, more formally known as a *power law distribution*. A distribution is long tailed when a few data items are very popular and most of the items, comprising the long tail, are rare.
- 63.7% of queries occurred only once, 16.2% occurred twice and 6.5% occurred thrice (totaling 86.4%), which is consistent with the size of the long tail reported in other studies of search logs. This phenomenon has enabled small advertisers to show their presence by bidding for rare queries, which are related to the products they are trying to sell.
- Overture's listing was founded on the paid placement model, so it displayed as many sponsored results as it had to answer a query, before displaying any organic results, which were delivered by another search engine; see Fig. noting that each sponsored result is clearly marked as such at the end of the entry in the list. (The Overture listing was eventually phased out by Yahoo after they rebranded Overture's services and transformed them into their current search engine marketing platform.)

- The main web search engines display sponsored links on a results page in three possible places:
  1) above the organic list
  2) below the organic list
  3) In a separate, reduced width, area on the right-hand side of the organic results.

- See Figures, for sponsored links on Google, Yahoo, and (Microsoft's) Bing, respectively, highlighted by surrounding rectangles. Both Yahoo and Bing display additional sponsored links below the organic results, while as of early 2010 Google did not.
- All search engines have a heading above or besides the sponsored links to demarcate them from the organic results. We observe that the ads above and below the organic list come in horizontal blocks, while the ones on the side come in vertical blocks and are known as *skyscraper ads*.
- It is worth noting that the sponsored ads are geographically sensitive to the country from the search emanates, in this case from the United Kingdom.
- As we have seen search engines clearly mark results that are paid for, to distinguish them from organic results output from their ranking algorithm. This keeps them in line with a warning issued by the US Federal Trade Commission (FTC) to search engine companies, that they should "clearly and conspicuously disclose" that certain query results are paid for by web sites in order to have a higher placement in the ranked results.
- The FTC also urged search engine companies to provide clear descriptions of their paid inclusion and paid placement programs, and their impact on search results, so that consumers would be in a better position to use this knowledge when choosing which search engine to use.
- All commercial search engines we know of, now, clearly distinguish between organic results and sponsored results or links. This is good news for users, many of whom may be unaware of paid placement.

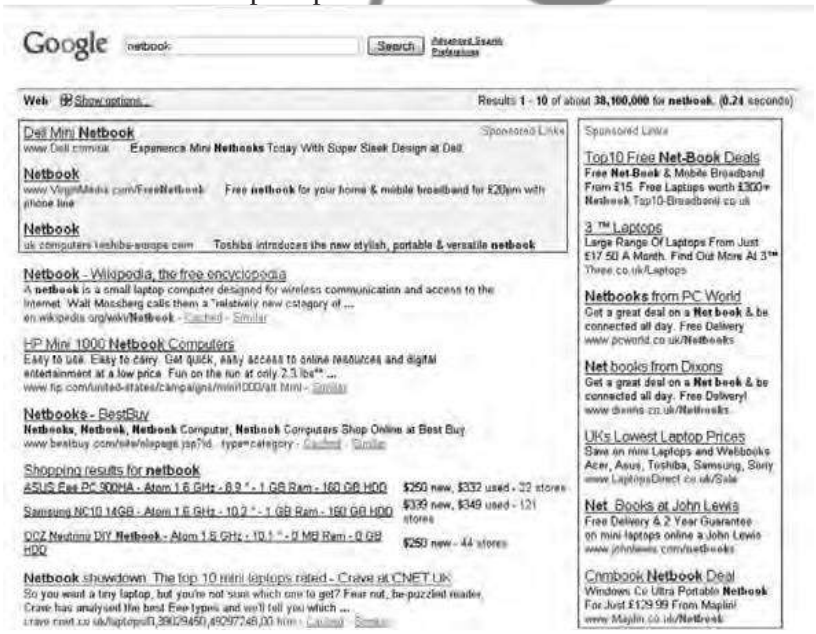


Figure . Query "netbook" submitted to Google.

3.5 THE SIZE OF THE WEB:

- Even if we exclude the hidden Web, it is difficult to compute a meaningful estimate for the size of the Web. Adding or removing a single host can change the number of accessible pages by an arbitrary amount, depending on the contents of that host. Some hosts contain millions of pages with valuable content; others contain millions of pages with little or no useful content .
- However, many Web pages would rarely, if ever, appear near the top of search results. Excluding those pages from a search engine would have little impact. Thus, we may informally define what is called the *indexable Web* as being those pages that should be considered for inclusion in a general-purpose Web search engine. This indexable Web would comprise all those pages that could have a substantial impact on search results.
- If we may assume that any page included in the index of a major search engine forms part of the indexable Web, a lower bound for the size of the indexable Web may be determined from the combined coverage of the major search engines. More specifically, if the sets A1,A2, . . . represent the sets of pages indexed by each of these search engines, a lower bound on the size of the indexable Web is the size of the union of these sets $|\cup Ai|$.
- Unfortunately, it is difficult to explicitly compute this union. Major search engines do not publish lists of the pages they index, or even provide a count of the number of pages, although it is usually possible to check if a given URL is included in the index. Even if we know the number of pages each engine contains, the size of the union will be smaller than the sum of the sizes because there is considerable overlap between engines.

Technique for estimating the combined coverage of major search engines:

1) First, a test set of URLs is generated. This step may be achieved by issuing a series of random queries to the engines and selecting a random URL from the results returned by each. We assume that this test set represents a uniform sample of the pages indexed by the engines.
2) Second, each URL from the test set is checked against each engine and the engines that contain it are recorded.

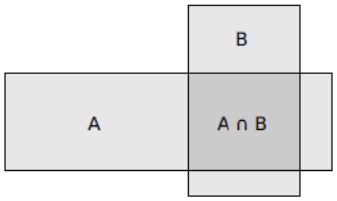


Figure.The collection overlap between search engines A and B can be used to estimate the size of the indexable Web.

Given two engines, A and B, the relationship between their collections is illustrated by Figure . Sampling with our test set of URLs allows us to estimate Pr[A ∩ B|A], the probability that a URL is contained in the intersection if it is contained in A:

$$\Pr[A \cap B \,|\, A] = \frac{\#\text{ of test URLs contained in both } A \text{ and } B}{\#\text{ of test URLs contained in } A}.$$

We may estimate Pr[A ∩ B|B] in a similar fashion. If we know the size of A, we may then estimate the size of the intersection as

$$|A \cap B| = |A| \cdot \Pr[A \cap B|A] .$$

and the size of B as

$$|B| = |A \cap B|$$
$$\Pr[A \cap B|B]$$

Thus, the size of the union may be estimated as

$$|A \cup B| = |A| + |B| - |A \cap B| .$$

If sizes for both A and B are available, the size of the intersection may be estimated from both sets, and the average of these estimates may be used to estimate the size of the union

$$|A \cup B| = |A| + |B| - 1/2 (|A| \cdot \Pr[A \cap B|A] + |B| \cdot \Pr[A \cap B|B]) .$$

- This technique may be extended to multiple engines. Using a variant of this method, Bharat and Broder (1998) estimated the size of the indexable Web in mid-1997 at 160 million pages. In a study conducted at roughly the same time, estimated the size at 320 million pages. Approximately a year later the size had grown to 800 million pages . By mid-2005 it was 11.5 billion.

3.6. SEARCH ENGINE OPTIMIZATION / SPAM:

- Early in the history of web search, it became clear that web search engines were an important means for connecting advertisers to prospective buyers.
- A user searching for maui golf real estate is not merely seeking news or entertainment on the subject of housing on golf courses on the island of Maui, but instead likely to be seeking to purchase such a property. Sellers of such property and their agents, therefore, have a strong incentive to create web pages that rank highly on this query.
- In a search engine whose scoring was based on term frequencies, a web page with numerous repetitions of maui golf spam real estate would rank highly. This led to the first generation of *spam*, which is the manipulation of web page content for the purpose of appearing high up in search results for selected keywords.

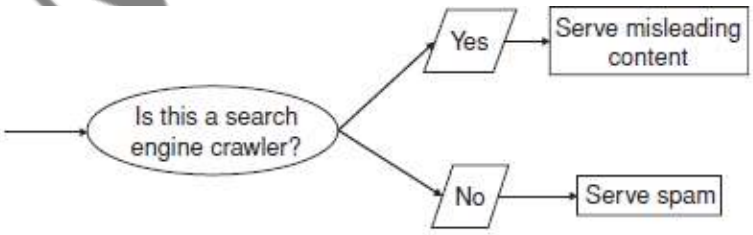


Figure. Cloaking as used by spammers.

- To avoid irritating users with these repetitions, sophisticated *spammers* resorted to such tricks as rendering these repeated terms in the same color as the background.

- Despite these words being consequently invisible to the human user, a search engine indexer would parse the invisible words out of the HTML representation of the web page and index these words as being present in the page.
- At its root, spam stems from the heterogeneity of motives in content creation on the Web. In particular, many web content creators have commercial motives and therefore stand to gain from manipulating search engine results.
- In many search engines, it is possible to pay to have one's web page included paid in the search engine's index – a model known as *paid inclusion*. Different inclusion search engines have different policies on whether to allow paid inclusion, and whether such a payment has any effect on ranking in search results.
- Search engines soon became sophisticated enough in their spam detection to screen out a large number of repetitions of particular keywords. Spammers responded with a richer set of spam techniques.

***Cloaking*:**

- The spammer's web server returns different pages depending on whether the http request comes from a web search engine's crawler or from a human user's browser. The former causes the web page to be indexed by the search engine under misleading keywords.
- When the user searches for these keywords and elects to view the page, he receives a web page that has altogether different content than that indexed by the search engine. Such deception of search indexers is unknown in the traditional world of IR; it stems from the fact that the relationship between page publishers and web search engines is not completely collaborative.

Doorway page:

- A *doorway page* contains text and metadata carefully chosen to rank highly on selected search keywords. When a browser requests the doorway page, it is redirected to a page containing content of a more commercial nature. More complex spamming techniques involve manipulation of the metadata related to a page including the links into a web page.

Search engine optimizers:

- Given that spamming is inherently an economically motivated activity, there has sprung search around it an industry of *search engine optimizers*, or engine optimizers SEOs, to provide consultancy services for clients who seek to have their web pages rank highly on selected keywords.
- Web search engines frown on this business of attempting to decipher and adapt to their proprietary ranking techniques and indeed announce policies on forms of SEO behavior they do not tolerate (and have been known to shut down search requests from certain SEOs for violation of these).
- Inevitably, the parrying between such SEOs (who gradually infer features of each web search engine's ranking methods) and the web search engines (who adapt in response) is an unending struggle; indeed, the research subarea of *adversarial information retrieval* has sprung up around this battle.
- To combat spammers who manipulate the text of their web pages is the exploitation of the link structure of the Web – a technique known as *link analysis*.
- The first web search engine known to apply link analysis on a large scale was Google, although all web search engines currently make use of it (and correspondingly, spammers link spam now invest considerable effort in subverting it – this is known as *link spam*).

3.7. ARCHITECTURE OF A SEARCH ENGINE

- Search engine architecture is used to present high-level descriptions of the important components of the system and the relationships between them.

The two primary goals of a search engine are:
• Effectiveness (quality): We want to be able to retrieve the most relevant set of documents possible for a query.
• Efficiency (speed): We want to process queries from users as quickly as possible

- The collection of documents we want to search may be changing; making sure that the search engine immediately reacts to changes in documents is both an effectiveness issue and an efficiency issue.

3.7.1. Basic Building Blocks

Search engine components support two major functions

1) *Indexing process*
2) *Query process.*

- The indexing process builds the structures that enable searching, and the query process uses those structures and a person's query to produce a ranked list of documents.

1) Indexing process:

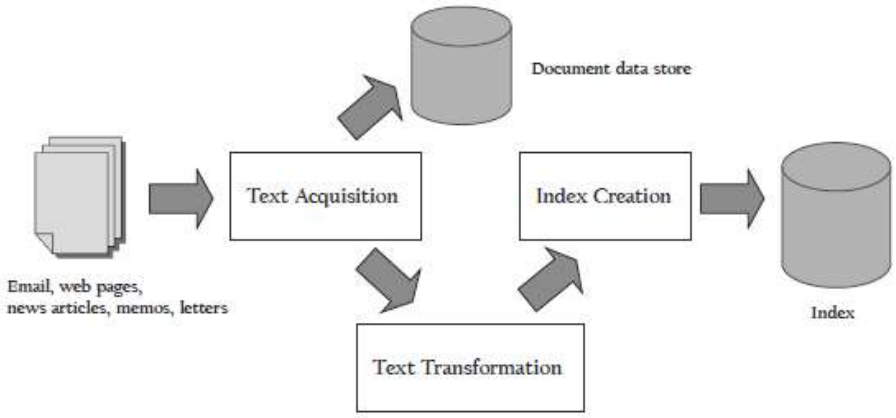


Fig.The indexing process

Figure shows the high-level "building blocks" of the indexing process. These major components are

- *Text acquisition*
- *Text transformation*
- *Index creation.*

Text acquisition:

- The task of the text acquisition component is to identify and make available the documents that will be searched. Although in some cases this will involve simply using an existing collection, text acquisition will more often require building a collection by

crawling or scanning the Web, a corporate intranet, a desktop, or other sources of information.

- In addition to passing documents to the next component in the indexing process, the text acquisition component creates a document data store, which contains the text and *metadata* for all the documents.
- Metadata is information about a document that is not part of the text content, such the document type (e.g., email or web page), document structure, and other features, such as document length.

- The text transformation component transforms documents into *index terms* or *features*. Index terms are the parts of a document that are stored in the index and used in searching. The simplest index term is a word, but not every word may be used for searching.
- A "feature" is more often used in the field of machine learning to refer to a part of a text document that is used to represent its content, which also describes an index term. Examples of other types of index terms or features are phrases, names of people, dates, and links in a web page. Index terms are sometimes simply referred to as "terms." The set of all the terms that are indexed for a document collection is called the *index vocabulary*.

- The index creation component takes the output of the text transformation component and creates the indexes or data structures that enable fast searching.
- Given the large number of documents in many search applications, index creation must be efficient, both in terms of time and space. Indexes must also be able to be efficiently *updated* when new documents are acquired.
- *Inverted indexes*, or sometimes *inverted files*, are by far the most common form of index used by search engines. An inverted index, very simply, contains a list for every index term of the documents that contain that index term. It is inverted in the sense of being the opposite of a document file that lists, for every document, the index terms they contain. There are many variations of inverted indexes, and the particular form of index used is one of the most important aspects of a search engine.

2) Query process:

- Figure shows the building blocks of the query process. The major components are *user interaction*, *ranking*, and *evaluation*.
- The user interaction component provides the interface between the person doing the searching and the search engine. One task for this component is accepting the user's query and transforming it into index terms.
- Another task is to take the ranked list of documents from the search engine and organize it into the results shown to the user. This includes, for example, generating the *snippets* used to summarize documents.
- The document data store is one of the sources of information used in generating the results. Finally, this component also provides a range of techniques for refining the query so that it better represents the information need.

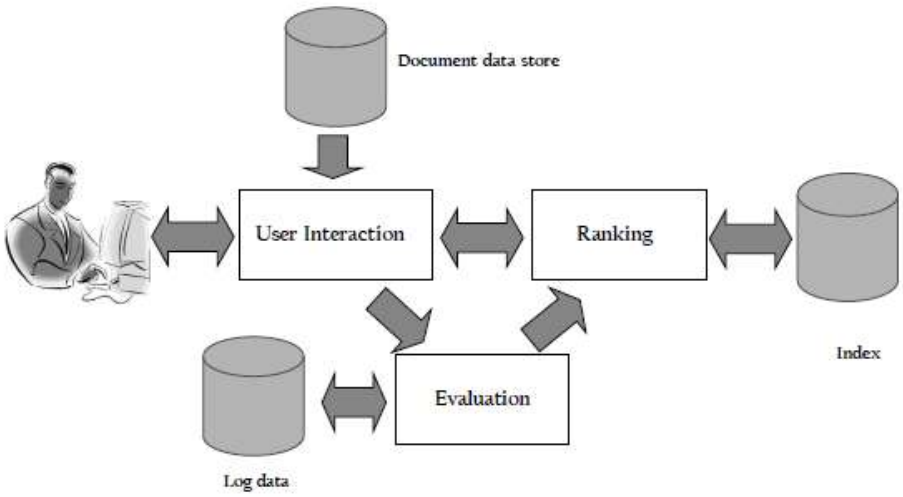


Fig. The query process

- The ranking component is the core of the search engine. It takes the transformed query from the user interaction component and generates a ranked list of documents using scores based on a retrieval model.
- Ranking must be both efficient, since many queries may need to be processed in a short time, and effective, since the quality of the ranking determines whether the search engine accomplishes the goal of finding relevant information. The efficiency of ranking depends on the indexes, and the effectiveness depends on the retrieval model.
- The task of the evaluation component is to measure and monitor effectiveness and efficiency. An important part of that is to record and analyze user behavior using *log data*.
- The results of evaluation are used to tune and improve the ranking component. Most of the evaluation component is not part of the online search engine, apart from logging user and system data. Evaluation is primarily an offline activity, but it is a critical part of any search application.

3.7.2. Breaking It Down

- More detail at the components of each of the basic building blocks. Not all of these components will be part of every search engine, but together they cover what we consider to be the most important functions for a broad range of search applications.

Text Acquisition:

i) Crawler:

- In many applications, the *crawler* component has the primary responsibility for identifying and acquiring documents for the search engine. There are a number of different types of crawlers, but the most common is the general web crawler.
- A web crawler is designed to follow the links on web pages to discover and download new pages. Although this sounds deceptively simple, there are significant challenges in designing a web crawler that can efficiently handle the huge volume of new pages on the Web, while at the same time ensuring that pages that may have changed since the last time a crawler visited a site are kept “fresh” for the search engine.

- A web crawler can be restricted to a single site, such as a university, as the basis for *site search*. *Focused*, or *topical*, web crawlers use classification techniques to restrict the pages that are visited to those that are likely to be about a specific topic. This type of crawler may be used by a *vertical* or *topical* search application, such as a search engine that provides access to medical information on web pages.
- For enterprise search, the crawler is adapted to discover and update all documents and web pages related to a company's operation.
- An enterprise *document crawler* follows links to discover both external and internal (i.e., restricted to the corporate intranet) pages, but also must scan both corporate and personal directories to identify email, word processing documents, presentations, database records, and other company information. Document crawlers are also used for desktop search, although in this case only the user's personal directories need to be scanned.

ii) Feeds

- *Document feeds* are a mechanism for accessing a real-time stream of documents.
- For example, a news feed is a constant stream of news stories and updates. In contrast to a crawler, which must discover new documents, a search engine acquires new documents from a feed simply by monitoring it.
- *RSS* is a common standard used for web feeds for content such as news, blogs, or video. An RSS "reader" is used to subscribe to RSS feeds, which are formatted using *XML*.
- XML is a language for describing data formats, similar to HTML.
- The reader monitors those feeds and provides new content when it arrives. Radio and television feeds are also used in some search applications, where the "documents" contain automatically segmented audio and video streams, together with associated text from closed captions or speech recognition.

iii) Conversion

- The documents found by a crawler or provided by a feed are rarely in plain text.
- Instead, they come in a variety of formats, such as HTML, XML, Adobe PDF, Microsoft Word™, Microsoft PowerPoint®, and so on. Most search engines require that these documents be converted into a consistent text plus metadata format. In this conversion, the control sequences and non-content data associated with a particular format are either removed or recorded as metadata. In the case of HTML and XML, much of this process can be described as part of the text transformation component. For other formats, the conversion process is a basic step that prepares the document for further processing. PDF documents, for example, must be converted to text. Various utilities are available that perform this conversion, with varying degrees of accuracy. Similarly, utilities are available to convert the various Microsoft Office® formats into text.
- Another common conversion problem comes from the way text is *encoded* in a document. ASCII is a common standard single-byte character encoding scheme used for text. ASCII uses either 7 or 8 bits (extended ASCII) to represent either 128 or 256 possible characters. Some languages, however, such as Chinese, have many more characters than English and use a number of other encoding schemes.
- Unicode is a standard encoding scheme that uses 16 bits (typically) to represent most of the world's languages. Any application that deals with documents in different languages has to ensure that they are converted into a consistent encoding scheme before further processing.

iv) Document data store

- The document data store is a database used to manage large numbers of documents and the structured data that is associated with them. The document contents are typically stored in compressed form for efficiency. The structured data consists of document metadata and other information extracted from the documents, such as links and *anchor text* (the text associated with a link).
- A *relational database system* can be used to store the documents and metadata. Some applications, however, use a simpler, more efficient storage system to provide very fast retrieval times for very large document stores.
- Although the original documents are available on the Web, in the enterprise database, the document data store is necessary to provide fast access to the document contents for a range of search engine components. Generating summaries of retrieved documents, for example, would take far too long if the search engine had to access the original documents and reprocess them.

Text Transformation:

i) Parser

- The parsing component is responsible for processing the sequence of text *tokens* in the document to recognize structural elements such as titles, figures, links, and headings. *Tokenizing* the text is an important first step in this process. In many cases, tokens are the same as words. Both document and query text must be transformed into tokens in the same manner so that they can be easily compared. There are a number of decisions that potentially affect retrieval that make tokenizing non-trivial. For example, a simple definition for tokens could be strings of alphanumeric characters that are separated by spaces. This does not tell us, however, how to deal with special characters such as capital letters, hyphens, and apostrophes.
- Should we treat “apple” the same as “Apple”? Is “on-line” two words or one word? Should the apostrophe in “O’Connor” be treated the same as the one in “owner’s”? In some languages, tokenizing gets even more interesting. Chinese, for example, has no obvious word separator like a space in English.
- Document structure is often specified by a markup language such as HTML or XML. HTML is the default language used for specifying the structure of web pages. XML has much more flexibility and is used as a data interchange format for many applications. The document parser uses knowledge of the *syntax* of the markup language to identify the structure.
- Both HTML and XML use *tags* to define document *elements*. For example, <h2> Search </h2> defines “Search” as a second-level heading in HTML. Tags and other control sequences must be treated appropriately when tokenizing. Other types of documents, such as email and presentations, have a specific syntax and methods for specifying structure, but much of this may be removed or simplified by the conversion component.

ii) Stopping

- The stopping component has the simple task of removing common words from the stream of tokens that become index terms. The most common words are typically *function* words that help form sentence structure but contribute little on their own to the description of the topics covered by the text. Examples are “the”, “of ”, “to”, and “for”. Because they are so common, removing them can reduce the size of the indexes considerably. Depending on the retrieval model that is used as the basis of the ranking, removing these

words usually has no impact on the search engine's effectiveness, and may even improve it somewhat. Despite these potential advantages, it can be difficult to decide how many words to include on the *stopword list*. Some stopword lists used in research contain hundreds of words. The problem with using such lists is that it becomes impossible to search with queries like "to be or not to be" or "down under". To avoid this, search applications may use very small stopword lists (perhaps just containing "the") when processing document text, but then use longer lists for the default processing of query text.

iii) Stemming

- Stemming is another word-level transformation. The task of the stemming component (or *stemmer*) is to group words that are derived from a common *stem*. Grouping "fish", "fishes", and "fishing" is one example.
- By replacing each member of a group with one designated word (for example, the shortest, which in this case is "fish"), we increase the likelihood that words used in queries and documents will match.
- Stemming produces small improvements in ranking effectiveness. Similar to stopping, stemming can be done aggressively, conservatively, or not at all. Aggressive stemming can cause search problems. It may not be appropriate, for example, to retrieve documents about different varieties of fish in response to the query "fishing".
- Some search applications use more conservative stemming, such as simply identifying plural forms using the letter "s", or they may word variants to the query.
- Some languages, such as Arabic, have more complicated *morphology* than English, and stemming is consequently more important. An effective stemming component in Arabic has a huge impact on search effectiveness. In contrast, there is little word variation in other languages, such as Chinese, and for these languages stemming is not effective.

iv) Link extraction and analysis

- Links and the corresponding anchor text in web pages can readily be identified and extracted during document parsing. Extraction means that this information is recorded in the document data store, and can be indexed separately from the general text content. Web search engines make extensive use of this information through *link analysis* algorithms such as PageRank (Brin & Page, 1998). Link analysis provides the search engine with a rating of the popularity, and to some extent, the *authority* of a page (in other words, how important it is). *Anchor text*, which is the clickable text of a web link, can be used to enhance the text content of a page that the link points to. These two factors can significantly improve the effectiveness of web search for some types of queries.

v) Information extraction

- Information extraction is used to identify index terms that are more complex than single words. This may be as simple as words in bold or words in headings, but in general may require significant additional computation. Extracting syntactic features such as noun phrases, for example, requires some form of syntactic analysis or *part-of-speech tagging*. Research in this area has focused on techniques for extracting features with specific semantic content, such as *named entity* recognizers, which can reliably identify information such as person names, company names, dates, and locations.

vi) Classifier

- The classifier component identifies class-related metadata for documents or parts of documents. This covers a range of functions that are often described separately.

Classification techniques assign predefined class labels to documents. These labels typically represent topical categories such as "sports", "politics", or "business". Two important examples of other types of classification are identifying documents as spam, and identifying the non-content parts of documents, such as advertising.

- Clustering techniques are used to group related documents without predefined categories. These document groups can be used in a variety of ways during ranking or user interaction.

Index Creation:

- The task of the document statistics component is simply to gather and record statistical information about words, features, and documents. This information is used by the ranking component to compute scores for documents.
- The types of data generally required are the counts of index term occurrences (both words and more complex features) in individual documents, the positions in the documents where the index terms occurred, the counts of occurrences over groups of documents (such as all documents labeled "sports" or the entire collection of documents), and the lengths of documents in terms of the number of tokens.
- The actual data required is determined by the retrieval model and associated ranking algorithm. The document statistics are stored in *lookup tables*, which are data structures designed for fast retrieval.

i) Weighting

- Index term *weights* reflect the relative importance of words in documents, and are used in computing scores for ranking. The specific form of a weight is determined by the retrieval model.
- The weighting component calculates weights using the document statistics and stores them in lookup tables. Weights could be calculated as part of the query process, and some types of weights require information about the query, but by doing as much calculation as possible during the indexing process, the efficiency of the query process will be improved.
- One of the most common types used in older retrieval models is known as *tf.idf* weighting. There are many variations of these weights, but they are all based on a combination of the frequency or count of index term occurrences in a document (the *term frequency*, or *tf*) and the frequency of index term occurrence over the entire collection of documents (*inverse document frequency*, or *idf*).
- The *idf* weight is called inverse document frequency because it gives high weights to terms that occur in very few documents. A typical formula for *idf* is log*N*/*n*, where*N* is the total number of documents indexed by the search engine and *n* is the number of documents that contain a particular term.

ii) Inversion

- The *inversion* component is the core of the indexing process. Its task is to change the stream of document-term information coming from the text transformation component into term-document information for the creation of inverted indexes.
- The challenge is to do this efficiently, not only for large numbers of documents when the inverted indexes are initially created, but also when the indexes are updated with new documents from feeds or crawls. The format of the inverted indexes is designed for fast query processing and depends to some extent on the ranking algorithm used. The indexes are also compressed to further enhance efficiency.

iii) Index distribution

- The index distribution component distributes indexes across multiple computers and potentially across multiple sites on a network. Distribution is essential for efficient performance with web search engines. By distributing the indexes for a subset of the documents (*document distribution*), both indexing and query processing can be done in *parallel*. Distributing the indexes for a subset of terms (*term distribution*) can also support parallel processing of queries.
- *Replication* is a form of distribution where copies of indexes or parts of indexes are stored in multiple sites so that query processing can be made more efficient by reducing communication delays. Peer-to-peer search involves a less organized form of distribution where each node in a network maintains its own indexes and collection of documents.

User Interaction:

- The query input component provides an interface and a parser for a *query language*. The simplest query languages, such as those used in most web search interfaces, have only a small number of *operators*. An operator is a command in the query language that is used to indicate text that should be treated in a special way.
- In general, operators help to clarify the meaning of the query by constraining how text in the document can match text in the query. An example of an operator in a simple query language is the use of quotes to indicate that the enclosed words should occur as a phrase in the document, rather than as individual words with no relationship.
- A typical web query, however, consists of a small number of *keywords* with no operators. A keyword is simply a word that is important for specifying the topic of a query. Because the ranking algorithms for most web search engines are designed for keyword queries, longer queries that may contain a lower proportion of keywords typically do not work well.
- For example, the query "search engines" may produce a better result with a web search engine than the query "what are typical implementation techniques and data structures used in search engines".
- One of the challenges for search engine design is to give good results for a range of queries, and better results for more specific queries. More complex query languages are available, either for people who want to have a lot of control over the search results or for applications using a search engine.
- In the same way that the SQL query language is not designed for the typical user of a database application (the *end user*), these query languages are not designed for the end users of search applications.
- *Boolean* query languages have a long history in information retrieval. The operators in this language include Boolean AND, OR, and NOT, and some form of *proximity* operator that specifies that words must occur together within a specific distance (usually in terms of word count). Other query languages include these and other operators in a probabilistic framework designed to allow specification of features related to both document structure and content.

i) Query transformation

- The query transformation component includes a range of techniques that are designed to improve the initial query, both before and after producing a document ranking. The simplest processing involves some of the same text transformation techniques used on

document text. Tokenizing, stopping, and stemming must be done on the query text to produce index terms that are comparable to the document terms.

- *Spell checking* and *query suggestion* are query transformation techniques that produce similar output. In both cases, the user is presented with alternatives to the initial query that are likely to either correct spelling errors or be more specific descriptions of their information needs. These techniques often leverage the extensive *query logs* collected for web applications.
- *Query expansion* techniques also suggest or add additional terms to the query, but usually based on an analysis of term occurrences in documents. This analysis may use different sources of information, such as the whole document collection, the retrieved documents, or documents on the user's computer. *Relevance feedback* is a technique that expands queries based on term occurrences in documents that are identified as relevant by the user.

ii) Results output

- The results output component is responsible for constructing the display of ranked documents coming from the ranking component. This may include tasks such as generating *snippets* to summarize the retrieved documents, *highlighting* important words and passages in documents, clustering the output to identify related groups of documents, and finding appropriate advertising to add to the results display. In applications that involve documents in multiple languages, the results may be translated into a common language.

Ranking:

- The scoring component, also called *query processing*, calculates scores for documents using the ranking algorithm, which is based on a retrieval model. The designers of some search engines explicitly state the retrieval model they use. For other search engines, only the ranking algorithm is discussed (if any details at all are revealed), but all ranking algorithms are based implicitly on a retrieval model.
- The features and weights used in a ranking algorithm, which may have been derived *empirically* (by testing and evaluation), must be related to topical and user relevance, or the search engine would not work.
- Many different retrieval models and methods of deriving ranking algorithms have been proposed. The basic form of the document score calculated by many of these models is

$$\sum_i q_i.d_i$$

- Where the summation is over all of the terms in the vocabulary of the collection, *qi* is the query term weight of the *i*th term, and *di* is the document term weight. The term weights depend on the particular retrieval model being used, but are generally similar to *tf.idf* weights. The document scores must be calculated and compared very rapidly in order to determine the ranked order of the documents that are given to the results output component. This is the task of the performance optimization component.

i) Performance optimization

- Performance optimization involves the design of ranking algorithms and the associated indexes to decrease response time and increase query throughput. Given a particular form of document scoring, there are a number of ways to calculate those scores and produce the ranked document output. For example, scores can be computed by accessing the

index for a query term, computing the contribution for that term to a document's score, adding this contribution to a score accumulator, and then accessing the next index. This is referred to as *term-at-a-time* scoring.

- Another alternative is to access all the indexes for the query terms simultaneously, and compute scores by moving pointers through the indexes to find the terms present in a document. In this *document-at-a-time* scoring, the final document score is calculated immediately instead of being accumulated one term at a time.
- In both cases, further optimizations are possible that significantly decrease the time required to compute the top-ranked documents. *Safe* optimizations guarantee that the scores calculated will be the same as the scores without optimization. *Unsafe* optimizations, which do not have this property, can in some cases be faster, so it is important to carefully evaluate the impact of the optimization.

ii) Distribution

- Given some form of index distribution, ranking can also be distributed. A *query broker* decides how to allocate queries to processors in a network and is responsible for assembling the final ranked list for the query. The operation of the broker depends on the form of index distribution. *Caching* is another form of distribution here indexes or even ranked document lists from previous queries are left in local memory. If the query or index term is popular, there is a significant chance that this information can be reused with substantial time savings.

Evaluation:

i) Logging

- Logs of the users' queries and their interactions with the search engine are one of the most valuable sources of information for tuning and improving search effectiveness and efficiency. Query logs can be used for spell checking, query suggestions, query caching, and other tasks, such as helping to match advertising to searches. Documents in a result list that are clicked on and browsed tend to be relevant. This means that logs of user clicks on documents (clickthrough data) and information such as the *dwell time* (time spent looking at a document) can be used to evaluate and train ranking algorithms.
- Given either log data or explicit relevance judgments for a large number of (query, document) pairs, the effectiveness of a ranking algorithm can be measured and compared to alternatives. This is a critical part of improving a search engine and selecting values for parameters that are appropriate for the application. A variety of evaluation measures are commonly used, and these should also be selected to measure outcomes that make sense for the application. Measures that emphasize the quality of the top-ranked documents, rather than the whole list, for example, are appropriate for many types of web queries.

ii) Performance analysis

- The performance analysis component involves monitoring and improving overall system performance, in the same way that the ranking analysis component monitors effectiveness.
- A variety of performance measures are used, such as response time and throughput, but the measures used also depend on the application. For example, a distributed search application should monitor network usage and efficiency in addition to other measures.

For ranking analysis, test collections are often used to provide a controlled experimental environment.

- The equivalent for performance analysis is *simulations*, where actual networks, processors, storage devices, and data are replaced with mathematical models that can be adjusted using parameters.

3.8. WEB CRAWLING AND INDEXES

- *Web crawling* is the process by which we gather pages from the Web to index them and support a search engine.
- The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them.

Features of a crawler:

1) **Robustness:**
    - The Web contains servers that create *spider traps*, which are generators of web pages that mislead crawlers into getting stuck fetching an infinite number of pages in a particular domain. Crawlers must be designed to be resilient to such traps. Not all such traps are malicious; some are the inadvertent side effect of faulty website development.
2) **Politeness:**
    - Web servers have both implicit and explicit policies regulating the rate at which a crawler can visit them. These politeness policies must be respected.
3) **Distributed:**
    - The crawler should have the ability to execute in a distributed fashion across multiple machines.
4) **Scalable:**
    - The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.
5) **Performance and efficiency:**
    - The crawl system should make efficient use of various system resources including processor, storage, and network bandwidth.
6) **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.
7) **Freshness:**
    - In many applications, the crawler should operate in continuous mode: It should obtain fresh copies of previously fetched pages. A search engine crawler should 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.

8) **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.

Crawling:

- The basic operation of any hypertext crawler is as follows.
- The crawler begins with one or more URLs that constitute a *seed set*. It picks a URL from this seed set and then fetches the web page at that URL. The fetched page is then parsed, to extract both the text and the links from the page (each of which points to another URL).
- The extracted text is fed to a text indexer. The extracted links (URLs) are then added to a *URL frontier*, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler.
- Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. The entire process may be viewed as traversing the web graph. In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future.
- This, seemingly simple, recursive traversal of the web graph is complicated by the many demands on a practical web crawling system: The crawler has to be distributed, scalable, efficient, polite, robust, and extensible while fetching pages of high quality.
- *Mercator* crawler has formed the basis of a number of research and commercial crawlers.

Crawler architecture:

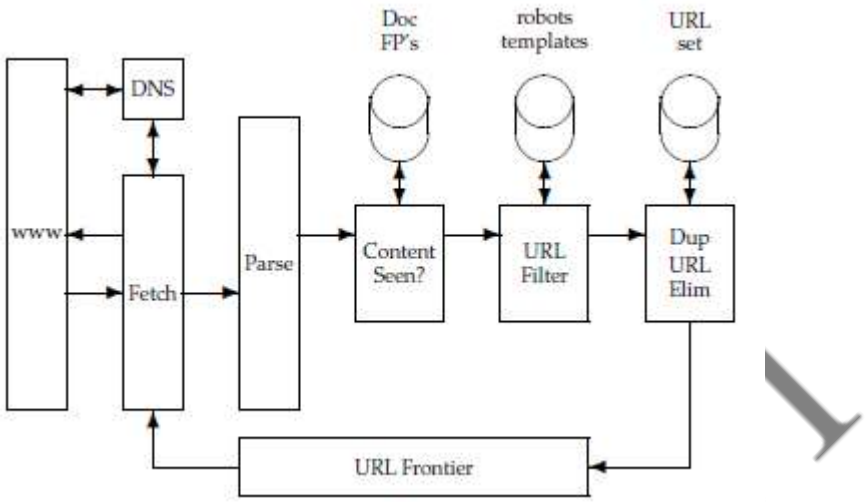


Figure. Basic crawler architecture.

1) The URL frontier, containing URLs yet to be fetched in the current crawl.
2) A *DNS resolution* module determines the web server from which to fetch the page specified by a URL.
3) A fetch module that uses the http protocol to retrieve the web page at a URL.
4) A parsing module that extracts the text and set of links from a fetched web page.
5) A duplicate elimination module that determines whether an extracted link is already in the URL frontier or has recently been fetched.

- Crawling is performed by anywhere from one to potentially hundreds of threads, each of which loops through the logical cycle in Figure. These threads may be run in a single process, or be partitioned among multiple processes running at different nodes of a distributed system.
- We begin by assuming that the URL frontier is in place and nonempty. We follow the progress of a single URL through the cycle of being fetched, passing through various checks and filters, then finally (for continuous crawling) being returned to the URL frontier.
- A crawler thread begins by taking a URL from the frontier and fetching the web page at that URL, generally using the http protocol. The fetched page is then written into a temporary store, where a number of operations are performed on it.
- Next, the page is parsed and the text as well as the links in it are extracted. The text (with any tag information – e.g., terms in boldface) is passed on to the indexer. Link information, including anchor text, is also passed on to the indexer for use in ranking in ways. In addition, each extracted link goes through a series of tests to determine whether the link should be added to the URL frontier.

- First, the thread tests whether a web page with the same content has already been seen at another URL. The simplest implementation for this would use a simple fingerprint such as a checksum (placed in a store labeled ”Doc FP’s” in Figure).
- Next, a *URL filter* is used to determine whether the extracted URL should be excluded from the frontier based on one of several tests. For instance, the crawl may seek to exclude certain domains (say, all .com URLs) – in this case the test would simply filter out the URL if it were from the .com domain.
- A similar test could be inclusive rather than exclusive. Many hosts on the Web place certain portions of their websites off-limits to crawling, under a standard known as the *robots exclusion protocol*. This is done by placing a file with the name robots.txt at the root of the URL hierarchy at the site.
- Here is an example robots.txt file that specifies that no robot should visit any URL whose position in the file hierarchy starts with /yoursite/temp/, except for the robot called “searchengine.”

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```

- The robots.txt file must be fetched from a website to test whether the URL under consideration passes the robot restrictions, and can therefore be added to the URL frontier.
- Rather than fetch it afresh for testing on each URL to be added to the frontier, a cache can be used to obtain a recently fetched copy of the file for the host. This is especially important because many of the links extracted from a page fall within the host from which the page was fetched and therefore can be tested against the host’s robots.txt file.
- Thus, by performing the filtering during the link extraction process, we would have especially high locality in the stream of hosts that we need to test for robots.txt files, leading to high cache hit rates.
- Unfortunately, this runs afoul of webmasters’ politeness expectations. A URL (particularly one referring to a low-quality or rarely changing document) may be in the frontier for days or even weeks. If we were to perform the robots filtering *before* adding such a URL to the frontier, its robots.txt file could have changed by the time the URL is dequeued from the frontier and fetched.
- We must consequently perform robots-filtering immediately before attempting to fetch a web page. As it turns out, maintaining a cache of robots.txt files is still highly effective; there is sufficient locality even in the stream of URLs dequeued from the URL frontier.

- Next, a URL should be *normalized* in the following sense: Often the HTML encoding of a link from a web page *p* indicates the target of that link relative to the page *p*. Thus, there is a relative link encoded thus in the HTML of the page en.wikipedia.org/wiki/Main Page:

```
<a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General
disclaimer">Disclaimers</a>
```

- Points to the URL http://en.wikipedia.org/wiki/Wikipedia:General disclaimer. Finally, the URL is checked for duplicate elimination: If the URL is already in the frontier or (in the case of a noncontinuous crawl) already crawled, we do not add it to the frontier. When the URL is added to the frontier, it is assigned a priority based on which it is eventually removed from the frontier for fetching.
- Certain housekeeping tasks are typically performed by a dedicated thread. This thread wakes up once every few seconds to log crawl progress statistics (URLs crawled, frontier size, etc.), decide whether to terminate the crawl, or (once every few hours of crawling) checkpoint the crawl.
- In *check pointing*, a snapshot of the crawler's state (say, the URL frontier) is committed to disk. In the event of a catastrophic crawler failure, the crawl is restarted from the most recent checkpoint.

Distributing the crawler

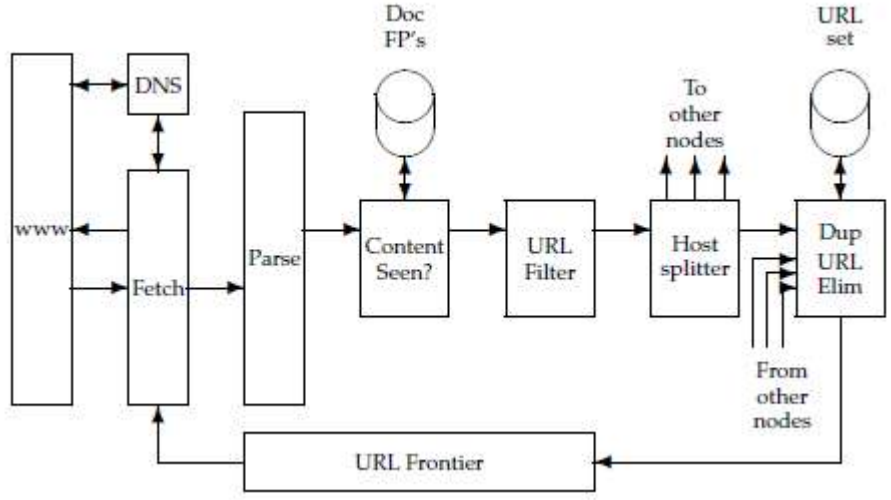


Figure . Distributing the basic crawl architecture.

- The threads in a crawler could run under different processes, each at a different node of a distributed crawling system. Such distribution is essential for scaling; it

can also be of use in a geographically distributed crawler system where each node crawls hosts "near" it.

- Partitioning the hosts being crawled among the crawler nodes can be done by a hash function, or by some more specifically tailored policy. For instance, we may locate a crawler node in Europe to focus on European domains, although this is not dependable for several reasons – the routes that packets take through the Internet do not always reflect geographic proximity, and in any case the domain of a host does not always reflect its physical location.
- Following the URL filter, we use a *host splitter* to dispatch each surviving URL to the crawler node responsible for the URL; thus the set of hosts being crawled is partitioned among the nodes. The output of the host splitter goes into the duplicate URL eliminator block of each other node in the distributed system.

The "Content Seen?" module in the distributed architecture is complicated by several factors:

1) Unlike the URL frontier and the duplicate elimination module, document fingerprints/shingles cannot be partitioned based on host name. There is nothing preventing the same content from appearing on different web servers. Consequently, the set of fingerprints/shingles must be partitioned across the nodes based on some property of the fingerprint/ shingle (say by taking the fingerprint modulo the number of nodes).The result of this locality mismatch is that most "Content Seen?" tests result in a remote procedure call.
2) There is very little locality in the stream of document fingerprints/ shingles. Thus, caching popular fingerprints does not help.
3) Documents change over time and so, in the context of continuous crawling, we must be able to delete their outdated fingerprints/shingles from the content-seen set(s). To do so, it is necessary to save the fingerprint/ shingle of the document in the URL frontier, along with the URL itself.

DNS resolution

- Each web server has a unique *IP address*: a sequence of four bytes generally represented as four integers separated by dots; for instance 207.142.131.248 is the numerical IP address associated with the host www.wikipedia.org.

- Given a URL such as www.wikipedia.org in textual form, translating it to an IP address (in this DNS case, 207.142.131.248) is a process known as *DNS resolution* or DNS lookup; here DNS stands for *domain name service*.

- During DNS resolution, the program that wishes to perform this translation contacts a *DNS server* that returns the translated IP address.
- (In practice, the entire translation may not occur at a single DNS server; rather, the DNS server contacted initially may recursively call upon other DNS servers to complete the translation.) For a more complex URL such as en.wikipedia.org/wiki/Domain Name System, the crawler component responsible for DNS resolution extracts the host name – in this case en.wikipedia.org – and looks up the IP address for the host en.wikipedia.org. DNS resolution is a well-known bottleneck in web crawling.
- Due to the distributed nature of the domain name service, DNS resolution may entail multiple requests and roundtrips across the Internet, requiring seconds and sometimes even longer. Right away, this puts in difficulty for our goal of fetching several hundred documents a second.
- A standard remedy is to introduce caching: URLs for which we have recently performed DNS lookups are likely to be found in the DNS cache, avoiding the need to go to the DNS servers on the Internet. However, obeying politeness constraints limits the cache hit rate.
- There is another important difficulty in DNS resolution; the lookup implementations in standard libraries are generally synchronous. This means that once a request is made to the domain name service, other crawler threads at that node are blocked until the first request is completed.
- To avoid this, most web crawlers implement their own DNS resolver as a component of the crawler. Thread *i* executing the resolver code sends a message to the DNS server and then performs a timed wait: It resumes either when being signaled by another thread or when a set time quantum expires.
- A single, separate DNS thread listens on the standard DNS port (port 53) for incoming response packets from the name service. Upon receiving a response, it signals the appropriate crawler thread (in this case, *i*) and hands it the response packet if *i* has not yet resumed because its time quantum has expired.
- A crawler thread that resumes because its wait time quantum has expired retries for a fixed number of attempts, sending out a new message to the DNS server and performing a timed wait each time; the designers of Mercator recommend of the order of five attempts.
- The time quantum of the wait increases exponentially with each of these attempts; Mercator started with one second and ended with roughly 90 seconds, in consideration of the fact that there are host names that take tens of seconds to resolve.

The URL frontier

- Two important considerations govern the order in which URLs are returned by the frontier. First, high-quality pages that change frequently should be

prioritized for frequent crawling. Thus, the priority of a page should be a function of both its change rate and its quality.

- The second consideration is politeness: We must avoid repeated fetch requests to a host within a short time span. The likelihood of this is exacerbated because of a form of locality of reference; many URLs link to other URLs at the same host. As a result, a URL frontier implemented as a simple priority queue might result in a burst of fetch requests to a host. This might occur even if we were to constrain the crawler so that at most one thread could fetch from any single host at any time.
- A common heuristic is to insert a gap between successive fetch requests to a host that is an order of magnitude larger than the time taken for the most recent fetch from that host.
- Figure shows a polite and prioritizing implementation of a URL frontier.Its goals are to ensure that
    - (i) only one connection is open at a time to any host,
    - (ii) a waiting time of a few seconds occurs between successive requests to a host
    - (iii) High-priority pages are crawled preferentially.
- The two major sub modules are a set of *F front queues* in the upper portion of the figure and a set of *B back queues* in the lower part; all of these are FIFO queues.

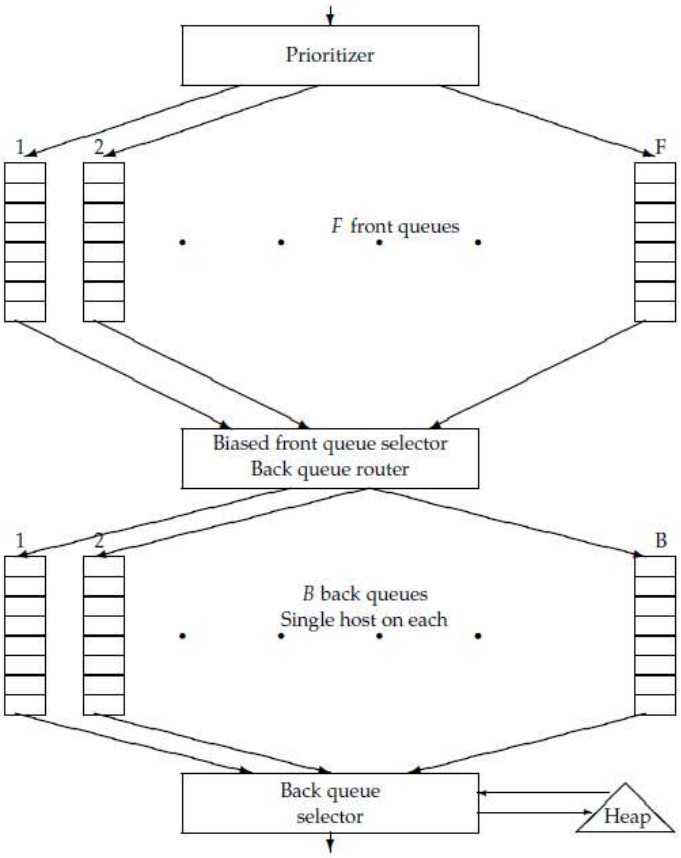


Figure . The URL frontier. URLs extracted from already crawled pages flow in at the top of the figure. A crawl thread requesting a URL extracts it from the bottom of the figure. En route, a URL flows through one of several *front queues* that manage its priority for crawling, followed by one of several *back queues* that manage the crawler's politeness.

- The front queues implement the prioritization, and the back queues implement politeness. In the flow of a URL added to the frontier as it makes its way through the front and back queues, a *prioritizer* first assigns to the URL an integer priority *i* between 1 and *F* based on its fetch history.
- For instance, a document that has exhibited frequent change would be assigned a higher priority. Other heuristics could be application dependent and explicit; for instance, URLs from news services may always be assigned the highest priority.

host	back queue
stanford.edu	23
microsoft.com	47
acm.org	12

Figure . Example of an auxiliary hosts-to-back queues table.

- Now that it has been assigned priority *i*, the URL is now appended to the *i*th of the front queues.
- Each of the *B* back queues maintains the following invariants:
  (i) It is nonempty while the crawl is in progress and
  (ii) It only contains URLs from a single host.
- An auxiliary table *T* is used to maintain the mapping from hosts to back queues. Whenever a back queue is empty and is being refilled from a front queue, table *T* must be updated accordingly.
- In addition, we maintain a heap with one entry for each back queue, the entry being the earliest time t_e at which the host corresponding to that queue can be contacted again.
- A crawler thread requesting a URL from the frontier extracts the root of this heap and (if necessary) waits until the corresponding time entry t_e. It then takes the URL *u* at the head of the back queue *j* corresponding to the extracted heap root, and proceeds to fetch the URL *u*. After fetching *u*, the calling thread checks whether *j* is empty. If so, it picks a front queue and extracts from its head a URL *v*. The choice of front queue is biased (usually by a random process) toward queues of higher priority, ensuring that URLs of high priority flow more quickly into the back queues.
- We examine *v* to check whether there is already a back queue holding URLs from its host. If so, *v* is added to that queue and we reach back to the front queues to find another candidate URL for insertion into the now-empty queue *j*. This process continues until *j* is nonempty again. In any case, the thread inserts a heap entry for *j* with a new earliest time *te* based on the properties of the URL in *j* that was last fetched (such as when its host was last contacted as well as the time taken for the last fetch), then continues with its processing. For instance, the new entry *te* could be the current time plus ten times the last fetch time.
- The number of front queues, together with the policy of assigning priorities and picking queues, determines the priority properties we wish to build into the system. The number of back queues governs the extent to which we can keep all crawl threads busy while respecting politeness. The designers of Mercator recommend a rough rule of three times as many back queues as crawler threads.
- On a web-scale crawl, the URL frontier may grow to the point where it demands more memory at a node than is available. The solution is to let most of the URL frontier reside on disk. A portion of each queue is kept in memory, with more brought in from disk as it is drained in memory.

DISTRIBUTING INDEXES:

- We now consider the distribution of the index across a large computer cluster that supports querying.
- Two obvious alternative index implementations :
  i) Partitioning by terms, also known as global index organization
  ii) Partitioning by document , also known as local index organization.

i) Global index organization:

- In the former, the dictionary of index terms is partitioned into subsets, each subset residing at a node.
- Along with the terms at a node, we keep the postings for those terms. A query is routed to the nodes corresponding to its query terms. In principle, this allows greater concurrency because a stream of queries with different query terms would hit different sets of machines.
- In practice, partitioning indexes by vocabulary terms turns out to be nontrivial.
- Multiword queries require the sending of long postings lists between sets of nodes for merging, and the cost of this can outweigh the greater concurrency.
- Load balancing the partition is governed not by an a priori analysis of relative term frequencies, but rather by the distribution of query terms and their co-occurrences, which can drift with time or exhibit sudden bursts.
- Achieving good partitions is a function of the co-occurrences of query terms and entails the clustering of terms to optimize objectives that are not easy to quantify. Finally, this strategy makes implementation of dynamic indexing more difficult.

ii) Local index organization:

- A more common implementation is to partition by documents: Each node contains the index for a subset of all documents. Each query is distributed to all nodes, with the results from various nodes being merged before presentation to the user. This strategy trades more local disk seeks for less internode communication. One difficulty in this approach is that global statistics used in scoring – such as idf – must be computed across the entire document collection even though the index at any single node only contains a subset of the documents. These are computed by distributed "background" processes that periodically refresh the node indexes with fresh global statistics.
- one simple approach would be to assign all pages from a host to a single node. This partitioning could follow the partitioning of hosts to crawler nodes. A danger of such partitioning is that, on many queries, a preponderance of the

results would come from documents at a small number of hosts (and, hence, a small number of index nodes).

- A hash of each URL into the space of index nodes results in a more uniform distribution of query time computation across nodes. At query time, the query is broadcast to each of the nodes, with the top *k* results from each node being merged to find the top *k* documents for the query. A common implementation heuristic is to partition the document collection into indexes of documents that are more likely to score highly on most queries and low-scoring indexes with the remaining documents. We only search the low-scoring indexes when there are too few matches in the high-scoring indexes.

3.9. FOCUSED CRAWLING

- Some users would like a search engine that focuses on a specific topic of information. For instance, at a website about movies, users might want access to a search engine that leads to more information about movies.
- If built correctly, this type of *vertical search* can provide higher accuracy than general search because of the lack of extraneous information in the document collection. The computational cost of running a vertical search will also be much less than a full web searches, simply because the collection will be much smaller.
- The most accurate way to get web pages for this kind of engine would be to crawl a full copy of the Web and then throw out all unrelated pages. This strategy requires a huge amount of disk space and bandwidth, and most of the web pages will be discarded at the end.
- A less expensive approach is *focused*, or *topical*, crawling. A focused crawler attempts to download only those pages that are about a particular topic.
- Focused crawlers rely on the fact that pages about a topic tend to have links to other pages on the same topic. If this were perfectly true, it would be possible to start a crawl at one on-topic page, and then crawl all pages on that topic just by following links from a single root page. In practice, a number of popular pages for a specific topic are typically used as seeds.
- Focused crawlers require some automatic means for determining whether a page is about a particular topic.
- For example, Text classifiers are tools that can make this kind of distinction. Once a page is downloaded, the crawler uses the classifier to decide whether the page is on topic. If it is, the page is kept, and links from the page are used to find other related sites. The anchor text in the outgoing links is an important clue of topicality. Also, some pages have more on-topic links than others.
- As links from a particular web page are visited, the crawler can keep track of the topicality of the downloaded pages and use this to determine whether to download other similar pages. Anchor text data and page link topicality data can be combined together in order to determine which pages should be crawled next.

Deep Web

- Not all parts of the Web are easy for a crawler to navigate. Sites that are difficult for a crawler to find are collectively referred to as the *deep Web* (also called the *hidden Web*).

Some studies have estimated that the deep Web is over a hundred times larger than the traditionally indexed Web, although it is very difficult to measure this accurately.

- Most sites that are a part of the deep Web fall into three broad categories:
    - *Private sites* are intentionally private. They may have no incoming links, or may require you to log in with a valid account before using the rest of the site. These sites generally want to block access from crawlers, although some news publishers may still want their content indexed by major search engines.
    - *Form results* are sites that can be reached only after entering some data into a form. For example, websites selling airline tickets typically ask for trip information on the site's entry page. You are shown flight information only after submitting this trip information. Even though you might want to use a search engine to find flight timetables, most crawlers will not be able to get through this form to get to the timetable information.
    - *Scripted pages* are pages that use JavaScript™, Flash®, or another client-side language in the web page. If a link is not in the raw HTML source of the web page, but is instead generated by JavaScript code running on the browser, the crawler will need to execute the JavaScript on the page in order to find the link.
- Although this is technically possible, executing JavaScript can slow down the crawler significantly and adds complexity to the system.
- Sometimes people make a distinction between *static pages* and *dynamic pages*. Static pages are files stored on a web server and displayed in a web browser unmodified, whereas dynamic pages may be the result of code executing on the web server or the client. Typically it is assumed that static pages are easy to crawl, while dynamic pages are hard. This is not quite true, however. Many websites have dynamically generated web pages that are easy to crawl; wikis are a good example of this. Other websites have static pages that are impossible to crawl because they can be accessed only through web forms.
- Web administrators of sites with form results and scripted pages often want their sites to be indexed, unlike the owners of private sites. Of these two categories, scripted pages are easiest to deal with. The site owner can usually modify the pages slightly so that links are generated by code on the server instead of by code in the browser. The crawler can also run page JavaScript, or perhaps Flash as well, although these can take a lot of time.
- The most difficult problems come with form results. Usually these sites are repositories of changing data, and the form submits a query to a database system.
- In the case where the database contains millions of records, the site would need to expose millions of links to a search engine's crawler. Adding a million links to the front page of such a site is clearly infeasible. Another option is to let the crawler guess what to enter into forms, but it is difficult to choose good form input. Even with good guesses, this approach is unlikely to expose all of the hidden data.

3.10. NEAR-DUPLICATES AND SHINGLING

- The Web contains multiple copies of the same content. By some estimates, as many as 40% of the pages on the Web are duplicates of other pages. Many of these are legal copies; for instance, certain information repositories are mirrored simply to provide redundancy and access reliability.
- Search engines try to avoid indexing multiple copies of the same content, to keep down storage and processing overheads. The simplest approach to detecting duplicates is to

compute, for each web page, a *fingerprint* that is a brief (say 64-bit) digest of the characters on that page.

- Then, whenever the fingerprints of two web pages are equal, we test whether the pages themselves are equal and if so declare one of them to be a duplicate copy of the other. This simplistic approach fails to capture a crucial and widespread phenomenon on the Web: *near duplication*.
    - In many cases, the contents of one web page are identical to those of another except for a few characters – say, a notation showing the date and time at which the page was last modified. Even in such cases, we want to be able to declare the two pages to be close enough that we only index one copy. Short of exhaustively comparing all pairs of web pages, an infeasible task at the scale of billions of pages, how can we detect and filter out such near duplicates?

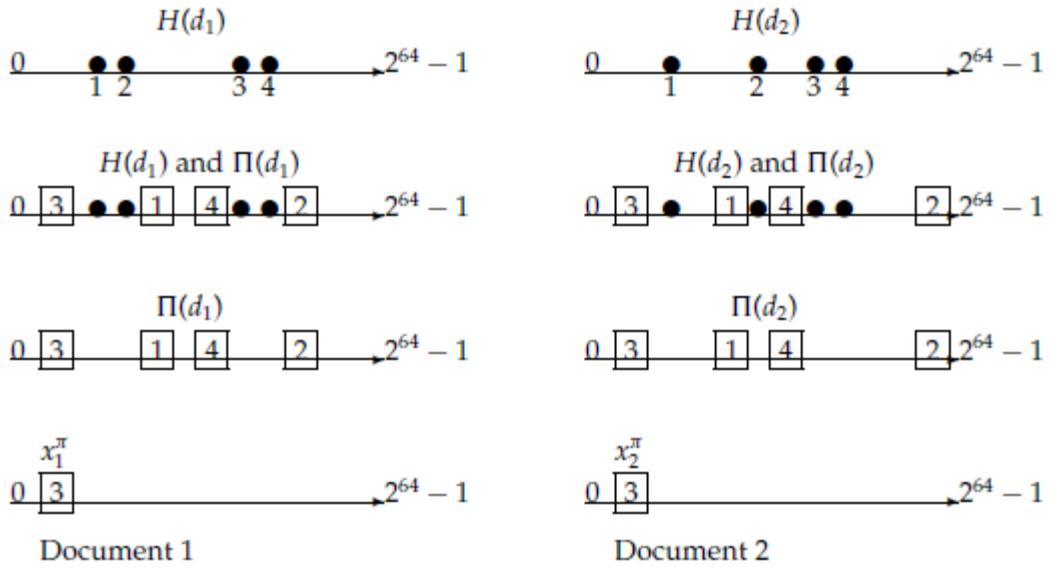


Figure . Illustration of shingle sketches. We see two documents going through four stages of shingle sketch computation. In the first step (*top row*), we apply a 64-bit hash to each shingle from each document to obtain $H(d_1)$ and $H(d_2)$ (*circles*). Next, we apply a random permutation Π to permute $H(d_1)$ and $H(d_2)$, obtaining $\Pi(d_1)$ and $\Pi(d_2)$ (*squares*). The third row shows only $\Pi(d_1)$ and $\Pi(d_2)$; the bottom row shows the minimum values x_1^π and x_2^π for each document.

- We now describe a solution to the problem of detecting near-duplicate web pages. The answer lies in a technique known as *shingling*.
- Given a positive integer k and a sequence of terms in a document d, define the k shingles of d to be the set of all consecutive sequences of k terms in d.
- As an example, consider the following text: a rose is a rose is a rose.
- The 4-shingles for this text ($k = 4$ is a typical value used in the detection of near-duplicate web pages) are a rose is a, rose is a rose, and is a rose is. The first two of these shingles each occur twice in the text. Intuitively, two documents are near duplicates if the sets of shingles generated from them are nearly the same.
- We now make this intuition precise, then develop a method for efficiently computing and comparing the sets of shingles for all web pages.

- Let $S(dj)$ denote the set of shingles of document dj . The Jaccard coefficient measures the degree of overlap between the sets $S(d1)$ and $S(d2)$ as $|S(d1) \cap S(d2)| / |S(d1) \cup S(d2)|$; denote this by $J(S(d1), S(d2))$.
- Our test for near duplication between $d1$ and $d2$ is to compute this Jaccard coefficient; if it exceeds a preset threshold (say, 0.9), we declare them near duplicates and eliminate one from indexing. However, this does not appear to have simplified matters: We still have to compute Jaccard coefficients pair wise. To avoid this, we use a form of hashing.

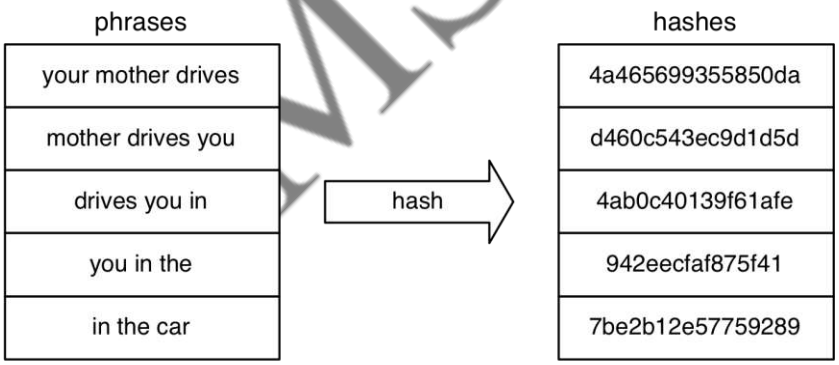
Example:

Each document is represented into a set of overlapping phrases of a few words at a time. This approach is known as shingling ,because each phrase overlaps its neighbors, much like shingles on a roof. So for, "Your mother drives you in the car," we'd get a set like this:

```
{
  'your mother drives',
  'mother drives you',
  'drives you in',
  'you in the',
  'in the car'
}
```

HASHING:

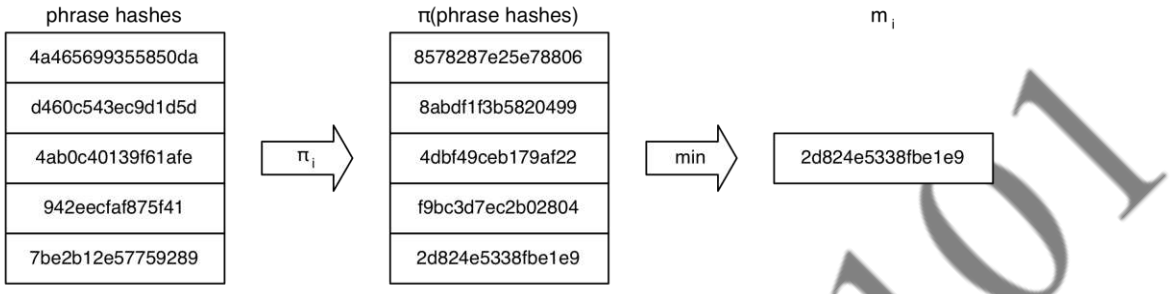
One of the problems with this bag of phrases approach is that we actually have to store the document several times over. In particular, if we use phrases of k words, then each word will appear in k phrases (except for words at the beginning and end), making the storage requirements O(nk). Turn each phrase into a number representative of the phrase using a hash function. So now instead of a bag of phrases, we have a bag of numbers on which we can still perform the same set intersection.



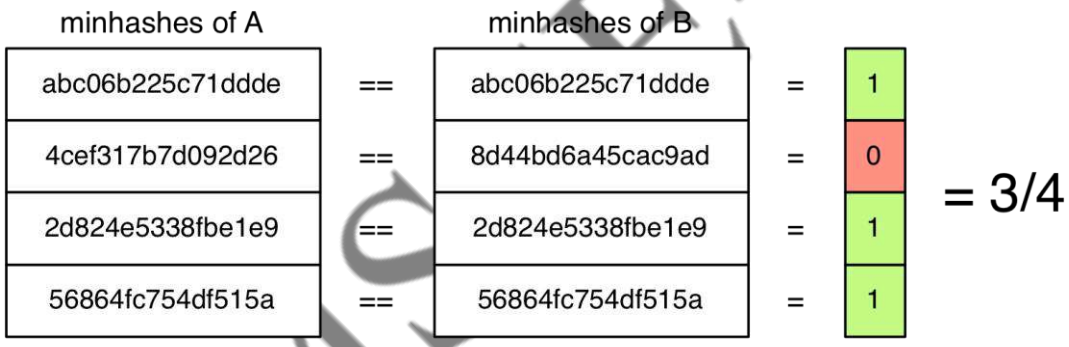
MinHash

While hashing reduces our per-document storage requirements down to O(n) in the number of documents, it's still too much. A lot of pages are really big, and when comparing two documents to determine the overlap, it must take at least O(n + m) time.

MinHash is an approach that is capable of using constant storage independent of the document length and producing a good estimate of our similarity measure. This approach distills down each document to a fixed-size set of hashes as a rough signature of this document. This feat is accomplished by using a set of k randomizing hash functions. For each randomizing hash function denoted π_i, we pass the entire document's phrase hashes through to get a minimum hash denoted m_i.



The signature of the document is now the ordered list of these minimum hashes m_0 through m_{k-1}. This method achieves an approximation to Jaccard similarity by comparing pairwise each of the document's minimum hashes and giving a portion that are identical.



This is a big win in two ways: the storage required for each document is now O(1), and our pairwise document comparison is now also O(1), an improvement over O(n) and O(m + n) respectively.

3.11.INDEX COMPRESSION

Benefits of compression:

1) Need less disk space. Compression ratios of 1:4 are easy to achieve, potentially cutting the cost of storing the index by 75%.
2) Increased use of caching. Search systems use some parts of the dictionary and the index much more than others. For example, if we cache the postings list of a frequently used

query term *t*, then the computations necessary for responding to the one-term query *t* can be entirely done in memory. With compression, we can fit a lot more information into main memory. Instead of having to expend a disk seek when processing a query with *t*, we instead access its postings list in memory and decompress it. There are simple and efficient decompression methods, so that the penalty of having to decompress the postings list is small. As a result, we are able to decrease the response time of the IR system substantially. Because memory is a more expensive resource than disk space, increased speed owing to caching – rather than decreased space requirements – is often the prime motivator for compression.

3) The second more subtle advantage of compression is faster transfer of data from disk to memory. Efficient decompression algorithms run so fast on modern hardware that the total time of transferring a compressed chunk of data from disk and then decompressing it is usually less than transferring the same chunk of data in uncompressed form. For instance, we can reduce input/output (I/O) time by loading a much smaller compressed postings list, even when you add on the cost of decompression. So, in most cases, the retrieval system runs faster on compressed postings lists than on uncompressed postings lists.

- If the main goal of compression is to conserve disk space, then the speed of compression algorithms is of no concern. But for improved cache utilization and faster disk-to-memory transfer, decompression speeds must be high. The compression algorithms we discuss in this chapter are highly efficient and can therefore serve all three purposes of index compression.
- A *posting is defined* as a docID in a postings list. For example, the postings list (6; 20, 45, 100), where 6 is the termID of the list's term, contains three postings. Postings in most search systems also contain frequency and position information; but we will only consider simple docID postings here.

1. Statistical properties of terms in information retrieval

- We use *Reuters-RCV1* collection as our model collection. RCV1 (Reuters Corbus Volume1) is a collection with roughly 1 GB of text. It consists of about 800,000 documents that were sent over the Reuters newswire during a 1-year period between August 20, 1996, and August 19, 1997.
- We give some term and postings statistics for the collection in Table. "Δ% " indicates the reduction in size from the previous line.
- "T%" is the cumulative reduction from unfiltered. The table shows the number of terms for different levels of preprocessing (column 2). The number of terms is the main factor in determining the size of the dictionary. The number of nonpositional postings (column 3) is an indicator of the expected size of the nonpositional index of the collection. The expected size of a positional index is related to the number of positions it must encode (column 4).
- In general, the statistics in Table show that preprocessing affects the size of the dictionary and the number of nonpositional postings greatly. Stemming and case folding reduce the number of (distinct) terms by 17% each and the number of nonpositional postings by 4% and 3%, respectively.
- The treatments of the most frequent words is also important. The *rule of 30* states that the 30 most common words account for 30% of the tokens in written text (31% in the table).

Eliminating the 150 most common words from indexing (as stop words; cf. Section 2.2.2, page 25) cuts 25% to 30% of the nonpositional postings. But, although a stop list of 150 words reduces the number of postings by a quarter or more, this size reduction does not carry over to the size of the compressed index.

	(distinct) terms			nonpositional postings			tokens (= number of position entries in postings)		
	number	Δ%	T%	number	Δ%	T%	number	Δ%	T%
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	−2	−2	100,680,242	−8	−8	179,158,204	−9	−9
case folding	391,523	−17	−19	96,969,056	−3	−12	179,158,204	−0	−9
30 stop words	391,493	−0	−19	83,390,443	−14	−24	121,857,825	−31	−38
150 stop words	391,373	−0	−19	67,001,847	−30	−39	94,516,599	−47	−52
stemming	322,383	−17	−33	63,812,300	−4	−42	94,516,599	−0	−52

Table. The effect of preprocessing on the number of terms, nonpositional postings, and tokens for Reuters-RCV1. "_%" indicates the reduction in size from the previous line, except that "30 stop words" and "150 stop words" both use "case folding" as their reference line. "T%" is the cumulative ("total") reduction from unfiltered. We performed stemming with the Porter stemmer.

- The postings lists of frequent words require only a few bits per posting after compression. The deltas in the table are in a range typical of large collections. Note, however, that the percentage reductions can be very different for some text collections.
- For example, for a collection of web pages with a high proportion of French text, a lemmatizer for French reduces vocabulary size much more than the Porter stemmer does for an English-only collection because French is a morphologically richer language than English.
- The compression techniques are lossless if all information is preserved. Better compression ratios can be achieved with *lossy compression*, which discards some information. Case folding, stemming, and stop word elimination are forms of lossy compression.
- Similarly, the vector space model and dimensionality reduction techniques like latent semantic indexing create compact representations from which we cannot fully restore the original collection.
- Lossy compression makes sense when the "lost" information is unlikely ever to be used by the search system. For example, web search is characterized by a large number of documents, short queries, and users who only look at the first few pages of results. As a consequence, we can discard postings of documents that would only be used for hits far down the list. Thus, there are retrieval scenarios where lossy methods can be used for compression without any reduction in effectiveness.
- Before introducing techniques for compressing the dictionary, we want to estimate the number of distinct terms M in a collection. It is sometimes said that languages have a vocabulary of a certain size. The second edition of the *Oxford English Dictionary* (OED) defines more than 600,000 words. But the vocabulary of most large collections is much

larger than the OED. The OED does not include most names of people, locations, products, or scientific entities like genes. These names need to be included in the inverted index, so our users can search for them.

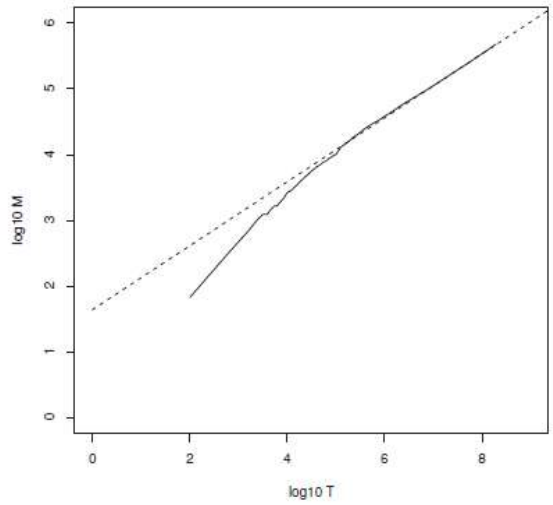


Figure . Heaps' law. Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line log10 M = 0.49 * $\log_{10} T$ + 1.64 is the best least squares fit. Thus, $k = 101.64 \approx 44$ and $b = 0.49$.

Heaps' law: Estimating the number of terms:

- A better way of getting a handle on M is *Heaps' law*, which estimates vocabulary size as a function of collection size:

$$M = kT^b$$

- Where T is the number of tokens in the collection. Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$. The motivation for Heaps' law is that the simplest possible relationship between collection size and vocabulary size is linear in log–log space and the assumption of linearity is usually born out in practice as shown in Figure for Reuters-RCV1. In this case, the fit is excellent for $T > 10^5 = 100{,}000$, for the parameter values $b = 0.49$ and $k = 44$. For example, for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1{,}000{,}020^{0.49} \approx 38{,}323.$$

- The actual number is 38,365 terms, very close to the prediction.
- The parameter k is quite variable because vocabulary growth depends a lot on the nature of the collection and how it is processed. Case-folding and stemming reduce the growth rate of the vocabulary, whereas including numbers and spelling errors increase it.
- Regardless of the values of the parameters for a particular collection, Heaps' law suggests that

(i) the dictionary size continues to increase with more documents in the collection, rather than a maximum vocabulary size being reached, and

(ii) The size of the dictionary is quite large for large collections. These two hypotheses have been empirically shown to be true of large text collections. So dictionary compression is important for an effective information retrieval system.

Zipf's law: Modeling the distribution of terms:

- We also want to understand how terms are distributed across documents. This helps us to characterize the properties of the algorithms for compressing postings lists.
- A commonly usedmodel of the distribution of terms in a collection is *Zipf's law*. It states that, if t_1 is the most common term in the collection, t_2 is the next most common, and so on, then the collection frequency cf_i of the i^{th} most common term is proportional to $1/i$:

$$cf_i \propto \frac{1}{i}.$$

So if the most frequent term occurs cf1 times, then the second most frequent term has half as many occurrences, the third most frequent term a third as any occurrences, and so on. The intuition is that frequency decreases very rapidly with rank. Equation (5.2) is one of the simplest ways of formalizing such a rapid decrease and it has been found to be a reasonably good model. Equivalently, we can write Zipf's law as $cf_i = ci_k$ or as $\log cf_i = \log c + k \log i$ where $k = -1$ and c is a constant. It is therefore a *power law* with exponent $k = -1$.

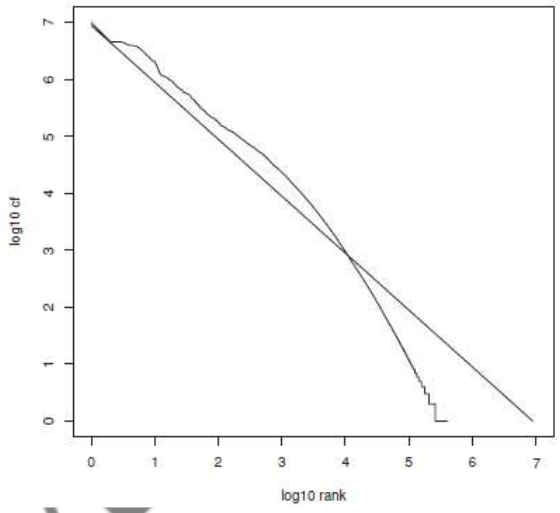


Figure. Zipf's law for Reuters-RCV1. Frequency is plotted as a function of frequency rank for the terms in the collection. The line is the distribution predicted by Zipf's law (weighted least-squares fit; intercept is 6.95).

- The log–log graph in Figure plots the collection frequency of a term as a function of its rank for Reuters-RCV1. A line with slope –1, corresponding to the Zipf function $\log cf_i = \log c - \log i$, is also shown.

2. Dictionary compression:

- One of the primary factors in determining the response time of an IR system is the number of disk seeks necessary to process a query. If parts of the dictionary are on disk, then many more disk seeks are necessary in query evaluation.
- Thus, the main goal of compressing the dictionary is to fit it in main memory, or at least a large portion of it, to support high query throughput. Although dictionaries of very large

collections fit into the memory of a standard desktop machine, this is not true of many other application scenarios.

- For example, an enterprise search server for a large corporation may have to index a multi terabyte collection with a comparatively large vocabulary because of the presence of documents in many different languages. We also want to be able to design search systems for limited hardware such as mobile phones and onboard computers. Other reasons for wanting to conserve memory are fast startup time and having to share resources with other applications.

Dictionary as a string:

- The simplest data structure for the dictionary is to sort the vocabulary lexicographically and store it in an array of fixed-width entries as shown in Figure. We allocate 20 bytes for the term itself (because few terms have more than twenty characters in English), 4 bytes for its document frequency, and 4 bytes for the pointer to its postings list.
- Four-byte pointers resolve a 4 gigabytes (GB) address space. For large collections like the web, we need to allocate more bytes per pointer. We look up terms in the array by binary search. For Reuters-RCV1, we need $M\times$ (20 + 4 + 4) = 400,000 × 28 = 11.2 megabytes (MB) for storing the dictionary in this scheme.

	term	document frequency	pointer to postings list
	a	656,265	⟶
	aachen	65	⟶
	…	…	…
	zulu	221	⟶
space needed:	20 bytes	4 bytes	4 bytes

Figure. Storing the dictionary as an array of fixed-width entries.

- Using fixed-width entries for terms is clearly wasteful. The average length of a term in English is about eight characters, so on average we are wasting twelve characters in the fixed-width scheme.
- Also, we have no way of storing terms with more than twenty characters like hydrochlorofluorocarbons and supercalifragilisticexpialidocious. We can overcome these shortcomings by storing the dictionary terms as one long string of characters, as shown in Figure.
- The pointer to the next term is also used to demarcate the end of the current term. As before, we locate terms in the data structure by way of binary search in the (now smaller) table. This scheme saves us 60% compared to fixed-width storage – 12 bytes on average of the 20 bytes we allocated for terms before. However, we now also need to store term pointers. The term pointers resolve 400,000 × 8 = 3.2×10^6 positions, so they need to be $\log_2 3.2\times10^6 \approx 22$ bits or 3 bytes long.

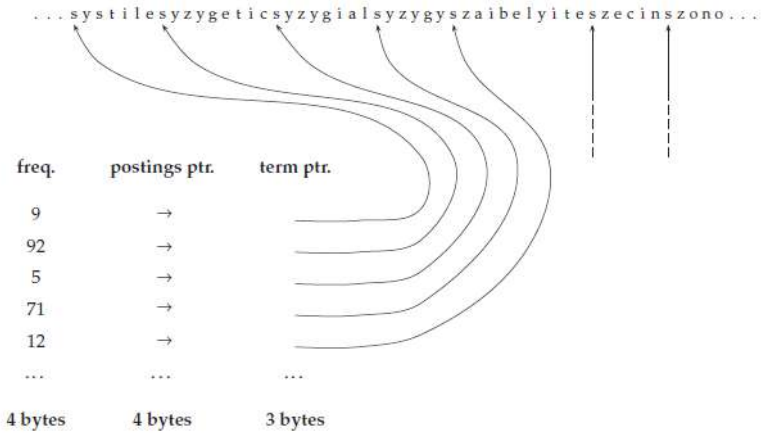


Figure. Dictionary-as-a-string storage. Pointers mark the end of the preceding term and the beginning of the next. For example, the first three terms in this example are systile, syzygetic, and syzygial.

- In this new scheme, we need 400,000 × (4 + 4 + 3 + 8) = 7.6 MB for the Reuters-RCV1 dictionary: 4 bytes each for frequency and postings pointer, 3 bytes for the term pointer, and 8 bytes on average for the term. So we have reduced the space requirements by one third from 11.2 to 7.6 MB.

Blocked storage:

- We can further compress the dictionary by grouping terms in the string into blocks of size *k* and keeping a term pointer only for the first term of each block . We store the length of the term in the string as an additional byte at the beginning of the term. We thus eliminate $k - 1$ term pointers, but need an additional *k* bytes for storing the length of each term.
- For $k = 4$, we save $(k - 1) \times 3 = 9$ bytes for term pointers, but need an additional $k = 4$ bytes for term lengths. So the total space requirements for the dictionary of Reuters-RCV1 are reduced by 5 bytes per four-term block, or a total of 400,000× 1/4× 5 = 0.5MB, bringing us down to 7.1 MB.

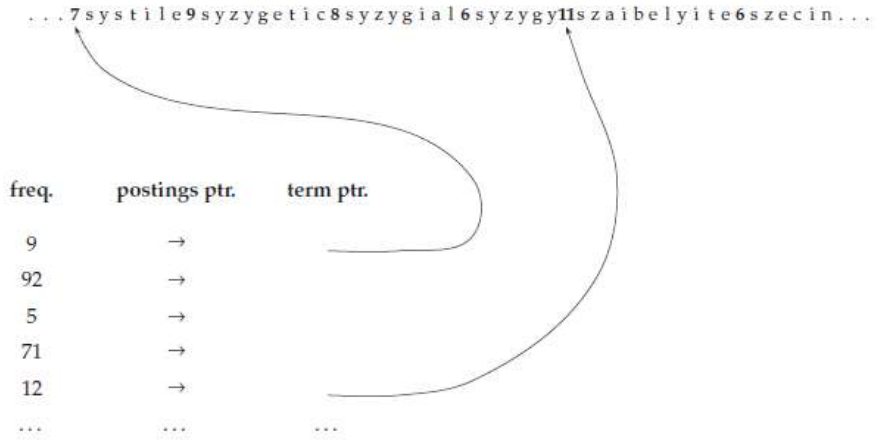


Figure . Blocked storage with four terms per block. The first block consists of systile, syzygetic, syzygial, and syzygy with lengths of seven, nine, eight, and six characters, respectively. Each term is preceded by a byte encoding its length that indicates how many bytes to skip to reach subsequent terms.

- By increasing the block size *k*, we get better compression. However, there is a tradeoff between compression and the speed of term lookup. For the eight-term dictionary in Figure , steps in binary search are shown as double lines and steps in list search as simple lines. We search for terms in the uncompressed dictionary by binary search (a).
- In the compressed dictionary, we first locate the term's block by binary search and then its position within the list by linear search through the block (b). Searching the uncompressed dictionary in (a) takes on average (0 +1 + 2+ 3+ 2 +1 +2 + 2)/8 ≈ 1.6 steps, assuming each term is equally likely to come up in a query. For example, finding the two terms, aid and box, takes three and two steps, respectively.
- With blocks of size $k = 4$ in (b), we need (0+1+2+3+4+1+2+3)/8 = 2 steps on average, ≈ 25% more. For example, finding den takes one binary search step and two steps through the block. By increasing *k*, we can get the size of the compressed dictionary arbitrarily close to the minimum of 400,000 × (4 + 4 + 1 + 8) = 6.8 MB, but term lookup becomes prohibitively slow for large values of *k*.
- One source of redundancy in the dictionary we have not exploited yet is the fact that consecutive entries in an alphabetically sorted list share common prefixes. This observation leads to *front coding*.

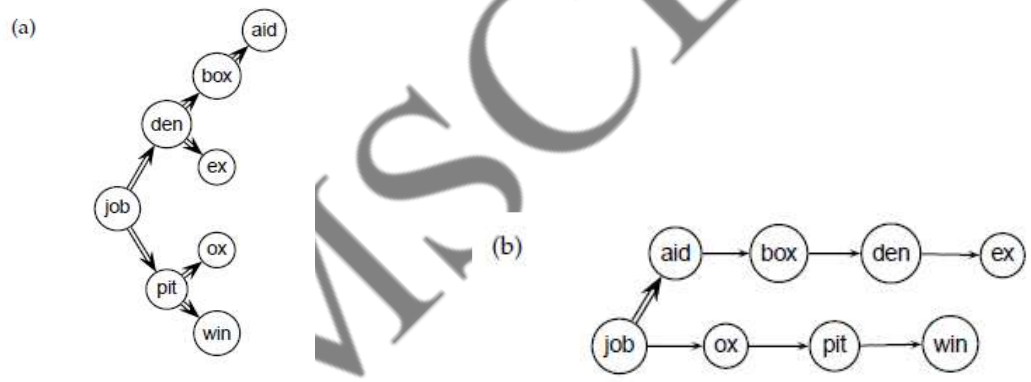


Figure. Search of the uncompressed dictionary (a) and a dictionary compressed by blocking with $k = 4$ (b).

One block in blocked compression ($k = 4$) …
8automata8automate9automatic10automation

⇓

…further compressed with front coding.
8automat∗a1⋄e2⋄ic3⋄ion

Figure. Front coding. A sequence of terms with identical prefix ("automat") is encoded by marking the end of the prefix with * and replacing it with ◇ in subsequent terms. As before, the first byte of each entry encodes the number of characters.

data structure	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
∼, with blocking, $k = 4$	7.1
∼, with blocking & front coding	5.9

Table. Dictionary compression for Reuters-RCV1.

- A common prefix is identified for a subsequence of the term list and then referred to with a special character. In the case of Reuters, front coding saves another 1.2 MB, as we found in an experiment. Other schemes with even greater compression rely on minimal perfect hashing, that is, a hash function that maps M terms onto [1, . . . ,M] without collisions. However, we cannot adapt perfect hashes incrementally because each new term causes a collision and therefore requires the creation of a new perfect hash function. Therefore, they cannot be used in a dynamic environment.
- Even with the best compression scheme, it may not be feasible to store the entire dictionary in main memory for very large text collections and for hardware with limited memory. If we have to partition the dictionary onto pages that are stored on disk, then we can index the first term of each page using a B-tree.
- For processing most queries, the search system has to go to disk anyway to fetch the postings. One additional seek for retrieving the term's dictionary page from disk is a significant, but tolerable increase in the time it takes to process a query. Table summarizes the compression achieved by the four dictionary data structures.

3. Postings file compression:

- Reuters-RCV1 has 800,000 documents, 200 tokens per document, six characters per token, and 100,000,000 postings where we define a posting in this chapter as a docID in a postings list, that is, excluding frequency and position information. These numbers correspond to line 3 ("case folding") in Table. Document identifiers are $\log_2 800{,}000 \approx 20$ bits long. Thus, the size of the collection is about 800,000× 200 × 6 bytes = 960 MB and the size of the uncompressed postings file is 100,000,000×20/8 = 250MB.

	encoding	postings list							
the	docIDs	…		283042		283043		283044	283045
	gaps				1		1	1	
computer	docIDs	…		283047		283154		283159	283202
	gaps				107		5	43	
arachnocentric	docIDs	252000		500100					
	gaps	252000	248100						

Table . Encoding gaps instead of document IDs. For example, we store gaps 107, 5, 43, . . . , instead of docIDs 283154, 283159, 283202, . . . for computer. The first docID is left unchanged (only shown for arachnocentric).

- To devise a more efficient representation of the postings file, one that uses fewer than 20 bits per document, we observe that the postings for frequent terms are close together. Imagine going through the documents of a collection one by one and looking for a frequent term like computer. We will find a document containing computer, then we skip a few documents that do not contain it, then there is again a document with the term and so on .
- The key idea is that the *gaps* between postings are short, requiring a lot less space than 20 bits to store. In fact, gaps for the most frequent terms such as the and for are mostly equal to 1. But the gaps for a rare term that occurs only once or twice in a collection (e.g., arachnocentric in Table) have the same order of magnitude as the docIDs and need 20 bits. For an economical representation of this distribution of gaps, we need a *variable encoding* method that uses fewer bits for short gaps.
- To encode small numbers in less space than large numbers, we look at two types of methods: byte wise compression and bitwise compression. As the names suggest, these methods attempt to encode gaps with the minimum number of bytes and bits, respectively.

Variable byte codes:

- *Variable byte (VB) encoding* uses an integral number of bytes to encode a gap. The last 7 bits of a byte are "payload" and encode part of the gap. The first bit of the byte is a *continuation bit*.
- It is set to 1 for the last byte of the encoded gap and to 0 otherwise. To decode a variable byte code, we read a sequence of bytes with continuation bit 0 terminated by a byte with continuation bit 1.We then extract and concatenate the 7-bit parts.
- Figure gives pseudo code for VB encoding and decoding and Table as an example of a VB-encoded postings list. With VB compression, the size of the compressed index for Reuters-RCV1 is 116MB as we verified in an experiment. This is a more than 50%reduction of the size of the uncompressed index.

```
VBENCODENUMBER(n)
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4     if n < 128
5        then BREAK
6     n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes

VBENCODE(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4     bytestream ← EXTEND(bytestream, bytes)
5  return bytestream

VBDECODE(bytestream)
1  numbers ← ⟨⟩
2  n ← 0
3  for i ← 1 to LENGTH(bytestream)
4  do if bytestream[i] < 128
5        then n ← 128 × n + bytestream[i]
6        else n ← 128 × n + (bytestream[i] − 128)
7             APPEND(numbers, n)
8             n ← 0
9  return numbers
```

Figure . VB encoding and decoding. The functions div and mod compute integer division and remainder after integer division, respectively. PREPEND adds an element to the beginning of a list, for example, PREPEND(⟨1, 2⟩, 3) = ⟨3, 1, 2⟩. EXTEND extends a list, for example, EXTEND(⟨1, 2⟩, ⟨3, 4⟩) = ⟨1, 2, 3, 4⟩.

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Table .VB encoding. Gaps are encoded using an integral number of bytes. The first bit, the continuation bit, of each byte indicates whether the code ends with this byte (1) or not (0).

number	unary code	length	offset	γ code
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110,0000000001

Table. Some examples of unary and γ codes. Unary codes are only shown for the smaller numbers. Commas in γ codes are for readability only and are not part of the actual codes.

- The idea of VB encoding can also be applied to larger or smaller units than bytes: 32-bit words, 16-bit words, and 4-bit words or *nibbles*. Larger words further decrease the amount of bit manipulation necessary at the cost of less effective (or no) compression. Word sizes smaller than bytes get even better compression ratios at the cost of more bit manipulation. In general, bytes offer a good compromise between compression ratio and speed of decompression.
- For most IR systems variable byte codes offer an excellent tradeoff between time and space. They are also simple to implement – most of the alternatives referred to in Section 5.4 are more complex. But if disk space is a scarce resource, we can achieve better compression ratios by using bit-level encodings, in particular two closely related encodings: γ codes, which we will turn to next, and δ codes.

***g* codes:**

- VB codes use an adaptive number of *bytes* depending on the size of the gap. Bit-level codes adapt the length of the code on the finer grained *bit* level. The simplest bit-level code is *unary code*. The unary code of *n* is a string of *n* 1s followed by a 0 (see the first two columns of Table). Obviously, this is not a very efficient code, but it will come in handy in a moment.
- How efficient can a code be in principle? Assuming the 2^n gaps G with $1 \leq G \leq 2^n$ are all equally likely, the optimal encoding uses n bits for each G. So some gaps ($G = 2^n$ in this case) cannot be encoded with fewer than $\log_2 G$ bits. Our goal is to get as close to this lower bound as possible.
- A method that is within a factor of optimal is γ *encoding*. γ codes implement variable-length encoding by splitting the representation of a gap G into a pair of *length* and *offset*. *Offset* is G in binary, but with the leading 1 removed. For example, for 13 (binary 1101) *offset* is 101. *Length* encodes the length of *offset* in unary code. For 13, the length of *offset* is 3 bits, which is 1110 in unary. The γ code of 13 is therefore 1110101, the concatenation of length 1110 and offset 101. The right hand column of Table 5.5 gives additional examples of γ codes.
- A γ code is decoded by first reading the unary code up to the 0 that terminates it, for example, the four bits 1110 when decoding 1110101. Now we know how long the offset is: 3 bits. The offset 101 can then be read correctly and the 1 that was chopped off in encoding is prepended: 101→1101 = 13.
- The length of *offset* is $\lfloor \log_2 G \rfloor$ bits and the length of *length* is $\lfloor \log_2 G \rfloor + 1$ bits, so the length of the entire code is $2 \times \lfloor \log_2 G \rfloor + 1$ bits. γ codes are always of odd length and they are within a factor of 2 of what we claimed to be the optimal encoding length $\log_2 G$. We derived this optimum from the assumption that the 2^n gaps between 1 and 2^n are equiprobable. But this need not be the case. In general, we do not know the probability distribution over gaps a priori.
- The characteristic of a discrete probability distribution P that determines its coding properties (including whether a code is optimal) is its *entropy* $H(P)$, which is defined as follows:

$$H(P) = -\sum_{x \in X} P(x) \log_2 P(x)$$

- Where X is the set of all possible numbers we need to be able to encode (and therefore $\sum_{x \in X} P(x) = 1.0$). Entropy is a measure of uncertainty as shown in Figure for a probability distribution P over two possible outcomes, namely, $X = \{x_1, x_2\}$. Entropy is maximized ($H(P) = 1$) for $P(x_1) = P(x_2) = 0.5$ when uncertainty about which *xi* will appear next is largest; and minimized ($H(P) = 0$) for $P(x_1) = 1$, $P(x_2) = 0$ and for $P(x_1) = 0$, $P(x_2) = 1$ when there is absolute certainty.

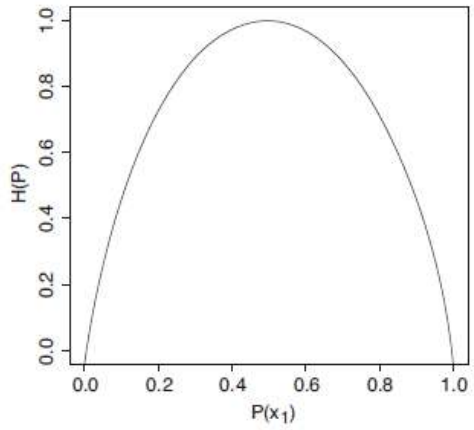


Figure . Entropy $H(P)$ as a function of $P(x1)$ for a sample space with two outcomes x_1 and x_2.

- It can be shown that the lower bound for the expected length $E(L)$ of a code L is $H(P)$ if certain conditions hold. It can further be shown that for $1 < H(P) < ¥$, γ encoding is within a factor of 3 of this optimal encoding, approaching 2 for large $H(P)$:

$$\frac{E(L_\gamma)}{H(P)} \leq 2 + \frac{1}{H(P)} \leq 3.$$

- What is remarkable about this result is that it holds for any probability distribution P. So without knowing anything about the properties of the distribution of gaps, we can apply γ codes and be certain that they are within a factor of ≈ 2 of the optimal code for distributions of large entropy. A code like γ code with the property of being within a factor of optimal for an arbitrary distribution P is called *universal*.
- In addition to universality, γ codes have two other properties that are useful for index compression. First, they are *prefix free*, namely, no γ code is the prefix of another. This means that there is always a unique decoding of a sequence of γ codes – and we do not need delimiters between them, which would decrease the efficiency of the code.
- The second property is that γ codes are *parameter free*. For many other efficient codes, we have to fit the parameters of a model (e.g., the binomial distribution) to the distribution of gaps in the index. This complicates the implementation of compression and decompression. For instance, the parameters need to be stored and retrieved.

- And in dynamic indexing, the distribution of gaps can change, so that the original parameters are no longer appropriate. These problems are avoided with a parameter-free code.
- How much compression of the inverted index do γ codes achieve? To answer this question we use Zipf's law, the term distribution model .According to Zipf's law, the collection frequency cf$i$ is proportional to the inverse of the rank i, that is, there is a constant c' such that:

$$\mathrm{cf}_i = \frac{c'}{i}.$$

- We can choose a different constant c such that the fractions c/i are relative frequencies and sum to 1 (that is, c/i = cf$i$/T):

$$1 = \sum_{i=1}^{M} \frac{c}{i} = c \sum_{i=1}^{M} \frac{1}{i} = c\, H_M$$

$$c = \frac{1}{H_M}$$

- Where M is the number of distinct terms and HM is the Mth harmonic number.
- Reuters-RCV1 has M = 400,000 distinct terms and $HM \approx \ln M$, so we have

$$c = \frac{1}{H_M} \approx \frac{1}{\ln M} = \frac{1}{\ln 400{,}000} \approx \frac{1}{13}.$$

- Thus the i^{th} term has a relative frequency of roughly 1/(13i), and the expected average number of occurrences of term i in a document of length L is:

$$L\frac{c}{i} \approx \frac{200 \times \frac{1}{13}}{i} \approx \frac{15}{i}$$

- Where we interpret the relative frequency as a term occurrence probability. Recall that 200 is the average number of tokens per document in Reuters- RCV1.
- Now we have derived term statistics that characterize the distribution of terms in the collection and, by extension, the distribution of gaps in the postings lists. From these statistics, we can calculate the space requirements for an inverted index compressed with γ encoding. We first stratify the vocabulary into blocks of size Lc = 15.
- On average, term i occurs 15/i times per document. So the average number of occurrences f per document is $1 \leq f$ for terms in the first block, corresponding to a total number of N gaps per term.

	N documents
Lc most frequent terms	N gaps of 1 each
Lc next most frequent terms	$N/2$ gaps of 2 each
Lc next most frequent terms	$N/3$ gaps of 3 each
…	…

Figure. Stratification of terms for estimating the size of a γ encoded inverted index.

- The average is $1/2 \leq f < 1$ for terms in the second block, corresponding to $N/2$ gaps per term, and $1/3 \leq f < 1/2$ for terms in the third block, corresponding to $N/3$ gaps per term, and so on. (We take the lower bound because it simplifies subsequent calculations.
- The final estimate is too pessimistic, even with this assumption.) We will make the somewhat unrealistic assumption that all gaps for a given term have the same size as shown in Figure. Assuming such a uniform distribution of gaps, we then have gaps of size 1 in block 1, gaps of size 2 in block 2, and so on.
- Encoding the N/j gaps of size j with γ codes, the number of bits needed for the postings list of a term in the jth block (corresponding to one row in the figure) is:

$$\begin{aligned} \textit{bits-per-row} &= \frac{N}{j} \times (2 \times \lfloor \log_2 j \rfloor + 1) \\ &\approx \frac{2N \log_2 j}{j}. \end{aligned}$$

- To encode the entire block,we need $(Lc) \cdot (2N\log 2\ j)/j$ bits. There are$M/(Lc)$ blocks, so the postings file as a whole will take up:

$$\sum_{j=1}^{\frac{M}{Lc}} \frac{2NLc \log_2 j}{j}.$$

data structure	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
∼, with blocking, $k = 4$	7.1
∼, with blocking & front coding	5.9
collection (text, xml markup etc)	3600.0
collection (text)	960.0
term incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ encoded	101.0

Table . Index and dictionary compression for Reuters-RCV1. The compression ratio depends on the proportion of actual text in the collection. Reuters-RCV1 contains a large amount of XML markup. Using the two best compression schemes, γ encoding and blocking with front coding, the ratio compressed index to collection size is therefore especially small for Reuters-RCV1: $(101 + 5.9)/3600 \approx 0.03$.

For Reuters-RCV1, $\frac{M}{Lc} \approx 400{,}000/15 \approx 27{,}000$ and

$$\sum_{j=1}^{27{,}000} \frac{2 \times 10^6 \times 15 \log_2 j}{j} \approx 224 \text{ MB}.$$

- So the postings file of the compressed inverted index for our 960 MB collection has a size of 224MB, one fourth the size of the original collection.
- When we run γ compression on Reuters-RCV1, the actual size of the compressed index is even lower: 101 MB, a bit more than one tenth of the size of the collection. The reason for the discrepancy between predicted and actual value is that (i) Zipf's law is not a very good approximation of the actual distribution of term frequencies for Reuters-RCV1 and (ii) gaps are not uniform.
- The Zipf model predicts an index size of 251MB for the unrounded numbers . If term frequencies are generated from the Zipf model and a compressed index is created for these artificial terms, then the compressed size is 254 MB. So to the extent that the assumptions about the distribution of term frequencies are accurate, the predictions of the model are correct. Table summarizes the compression techniques covered in this chapter.
- The term incidence matrix for Reuters-RCV1 has size 400,000×800,000 = 40 ×8 ×109 bits or 40 GB.
- γ codes achieve great compression ratios – about 15% better than variable byte codes for Reuters-RCV1. But they are expensive to decode. This is because many bit-level operations – shifts and masks – are necessary to decode a sequence of γ codes as the boundaries between codes will usually be somewhere in the middle of a machine word. As a result, query processing is more expensive for γ codes than for variable byte codes. Whether we choose variable byte or γ encoding depends on the characteristics of an application, for example, on the relative weights we give to conserving disk space versus maximizing query response time.

- The compression ratio for the index in Table is about 25%: 400MB (uncompressed, each posting stored as a 32-bitword) versus 101MB (γ) and 116 MB (VB). This shows that both γ and VB codes meet the objectives we stated in the beginning of the chapter. Index compression substantially improves time and space efficiency of indexes by reducing the amount of disk space needed, increasing the amount of information that can be kept in the cache, and speeding up data transfers from disk to memory.

3.12. XML RETRIEVAL

- Information retrieval systems are often contrasted with relational databases. Traditionally, IR systems have retrieved information from *unstructured text* – by which we mean "raw" text without markup.
- Databases are designed for querying *relational data*: sets of records that have values for predefined attributes such as employee number, title and salary. There are fundamental differences between information retrieval and database systems in terms of retrieval model, data structures and query language as shown in Table .
- Some highly structured text search problems are most efficiently handled by a relational database, for example, if the employee table contains an attribute for short textual job descriptions and you want to find all employees who are involved with invoicing. In this case, the SQL query:
  select lastname from employees where job_desc like 'invoic%';

- May be sufficient to satisfy your information need with high precision and recall.
- However, many structured data sources containing text are best modeled as structured documents rather than relational data. We call the search over such structured documents *structured retrieval*.
- Queries in structured retrieval can be either structured or unstructured, but we will assume in this chapter that the collection consists only of structured documents.
- Applications of structured retrieval include digital libraries, patent databases, blogs, text in which entities like persons and locations have been tagged (in a process called named entity tagging) and output from office suites like OpenOffice that save documents as marked up text. In all of these applications, we want to be able to run queries that combine textual criteria with structural criteria.
- Examples of such queries are give me a full-length article on fast fourier transforms (digital libraries), give me patents whose claims mention RSA public key encryption and that cite US patent 4,405,829 (patents), or give me articles about sightseeing tours of the Vatican and the Coliseum (entity-tagged text). These three queries are structured queries that cannot be answered well by an unranked retrieval system.
- Unranked retrieval models like the Boolean model suffer from low recall. For instance, an unranked system would return a potentially large number of articles that mention the Vatican, the Coliseum and sightseeing tours without ranking the ones that are most relevant for the query first. Most users are also notoriously bad at precisely stating structural constraints. For instance, users may not know for which structured elements the search system supports search.

- In our example, the user may be unsure whether to issue the query as sightseeing AND (COUNTRY:Vatican OR LANDMARK:Coliseum) , as sightseeing AND (STATE:Vatican OR BUILDING:Coliseum) or in some other form. Users may also be completely unfamiliar with structured search and advanced search interfaces or unwilling to use them. In this chapter, we look at how ranked retrieval methods can be adapted to structured documents to address these problems.

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured documents	trees with text at leaves
model	relational model	vector space & others	?
main data structure	table	inverted index	?
queries	SQL	free text queries	?

Table . RDB (relational database) search, unstructured information retrieval and structured information retrieval. There is no consensus yet as to which methods work best for structured retrieval although many researchers believe that XQuery will become the standard for structured queries.

- We will only look at one standard for encoding structured documents: *ExtensibleMarkup Language* or *XML*, which is currently the most widely used such standard. We will not cover the specifics that distinguish XML from other types of markup such as HTML and SGML. But most of what we say in this chapter is applicable to markup languages in general.
- In the context of information retrieval, we are only interested in XML as a language for encoding text and documents. A perhaps more widespread use of XML is to encode non-text data. For example, we may want to export data in XML format from an enterprise resource planning system and then read them into an analytics program to produce graphs for a presentation.
- This type of application of XML is called *data-centric* because numerical and non-text attribute-value data dominate and text is usually a small fraction of the overall data. Most data-centric XML is stored in databases – in contrast to the inverted index-based methods for text-centric XML that we present in this chapter.
- We call XML retrieval *structured retrieval* in this chapter. Some researchers prefer the term *semistructured retrieval* to distinguish XML retrieval from database querying. We have adopted the terminology that is widespread in the XML retrieval community. For instance, the standard way of referring to XML queries is *structured queries*, not *semistructured queries*. The term *structured retrieval* is rarely used for database querying and it always refers to XML retrieval in this book.
- There is a second type of information retrieval problem that is intermediate between unstructured retrieval and querying a relational database: parametric and zone search. In the data model of parametric and zone search, there are parametric fields (relational attributes like *date* or *file-size*) and zones – text attributes that each take a chunk of unstructured text as value, e.g., *author* and *title*. The data model is flat, that is, there is no nesting of attributes.

- The number of attributes is small. In contrast, XML documents have the more complex tree structure in which attributes are nested. The number of attributes and nodes is greater than in parametric and zone search.

Basic XML concepts:

- An XML document is an ordered, labeled tree. Each node of the tree is an XML ELEMENT *XML element* and is written with an opening and closing *tag*. An element can XML ATTRIBUTE have one or more *XML attributes*. In the XML document in Figure , the *scene* element is enclosed by the two tags <scene ...> and </scene>. It has an attribute *number* with value *vii* and two child elements, *title* and *verse*.

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene number="vii">
<title>Macbeth's castle</title>
<verse>Will I with wine and wassail ...</verse>
</scene>
</act>
</play>
```

Figure . An XML document.

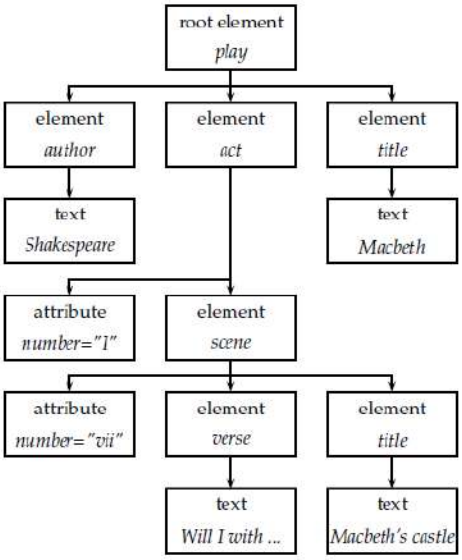


Figure . The XML document as a simplified DOM object.

- The *leaf nodes* of the tree consist of text, e.g., Shakespeare, Macbeth, and Macbeth's castle. The tree's *internal nodes* encode either the structure of the document (*title*, *act*, and *scene*) or metadata functions (*author*).

- The standard for accessing and processing XML documents is the XML Document Object Model or *DOM*. The DOM represents elements, attributes and text within elements as nodes in a tree.

```
//article
[.//yr = 2001 or .//yr = 2002]
//section
[about(.,summer holidays)]
```

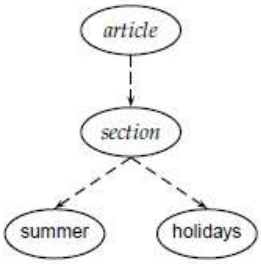


Figure . An XML query in NEXI format and its partial representation as a tree. can process an XML document by starting at the root element and then descending down the tree from parents to children.

- With a DOM API, we XPATH *XPath* is a standard for enumerating paths in an XML document collection. We will also refer to paths as *XML contexts* or simply *contexts*.
- Only a small subset of XPath is needed for our purposes. The XPath expression node selects all nodes of that name. Successive elements of a path are separated by slashes, so act/scene selects all *scene* elements whose parent is an *act* element. Double slashes indicate that an arbitrary number of elements can intervene on a path: play//scene selects all *scene* elements occurring in a *play* element.
- In Figure ,this set consists of a single *scene* element, which is accessible via the path *play, act, scene* from the top. An initial slash starts the path at the root element. /play/title selects the play's title in Figure , /play//title selects a set with two members (the play's title and the scene's title), and /scene/title selects no elements.
- For notational convenience, we allow the final element of a path to be a vocabulary term and separate it from the element path by the symbol #, even though this does not conform to the XPath standard. For example, title#"Macbeth" selects all titles containing the term Macbeth.
- A schema puts constraints on the structure of allowable XML documents for a particular application.
- A schema for Shakespeare's plays may stipulate that scenes can only occur as children of acts and that only acts and scenes have the *number* attribute. Two standards for schemas for XML documents are *XML DTD* XML SCHEMA (document type definition) and *XML Schema*. Users can only write structured queries for an XML retrieval system if they have

some minimal knowledge about the schema of the collection.

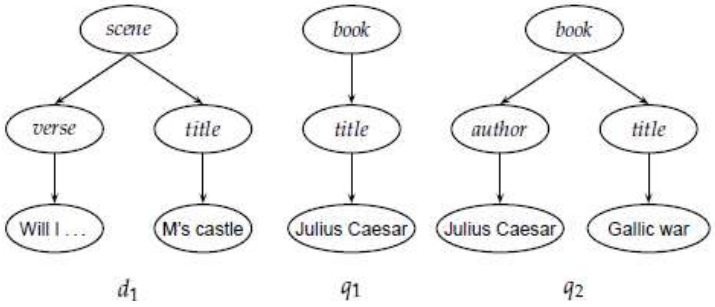


Figure. Tree representation of XML documents and queries.

- A common format for XML queries is *NEXI* (Narrowed NEXI Extended XPath I).We give an example in Figure. We display the query on four lines for typographical convenience, but it is intended to be read as one unit without line breaks. In particular, //section is embedded under //article.
- The query in Figure specifies a search for sections about the summer holidays that are part of articles from 2001 or 2002. As in XPath double slashes indicate that an arbitrary number of elements can intervene on a path. The dot in a clause in square brackets refers to the element the clause modifies. The clause [.//yr = 2001 or .//yr = 2002] modifies //article. Thus, the dot refers to //article in this case. Similarly, the dot in [about(., summer holidays)] refers to the section that the clause modifies.
- The two yr conditions are relational attribute constraints. Only articles whose yr attribute is 2001 or 2002 (or that contain an element whose yr attribute is 2001 or 2002) are to be considered. The about clause is a ranking constraint: Sections that occur in the right type of article are to be ranked according to how relevant they are to the topic summer holidays.
- We usually handle relational attribute constraints by prefiltering or postfiltering: We simply exclude all elements from the result set that do not meet the relational attribute constraints. In this chapter, we will not address how to do this efficiently and instead focus on the core information retrieval problem in XML retrieval, namely how to rank documents according to the relevance criteria expressed in the about conditions of the NEXI query.
- If we discard relational attributes, we can represent documents as trees with only one type of node: element nodes. In other words, we remove all attribute nodes from the XML document, such as the *number* attribute. Figure shows a subtree of the document as an element-node tree (labeled *d*1).
- We can represent queries as trees in the same way. This is a query-by example approach to query language design because users pose queries by creating objects that satisfy the same formal description as documents. In Figure, *q*1 is a search for books whose titles score highly for the keywords Julius Caesar. *q*2 is a search for books whose author elements score highly for Julius Caesar and whose title elements score highly for Gallic war.

Challenges in XML retrieval:

- Structured retrieval is more difficult than unstructured retrieval. Assume in structured retrieval: the collection consists of structured documents and queries are either structured or unstructured (e.g., summer holidays).
- The first challenge in structured retrieval is that users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval. If we query Shakespeare's plays for Macbeth's castle, should we return the scene, the act or the entire play in Figure ? In this case, the user is probably looking for the scene. On the other hand, an otherwise unspecified search for Macbeth should return the play of this name, not a subunit.
- One criterion for selecting the most appropriate part of a document is the *structured document retrieval principle*:
- **Structured document retrieval principle.** A system should always retrieve the most specific part of a document answering the query.
- This principle motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level. However, it can be hard to implement this principle algorithmically. Consider the query title#"Macbeth" applied to Figure. The title of the tragedy, *Macbeth*, and the title of Act I, Scene vii, *Macbeth's castle* are both good hits because they contain the matching term Macbeth. But in this case, the title of the tragedy, the higher node, is preferred. Deciding which level of the tree is right for answering a query is difficult.
- Parallel to the issue of which parts of a document to return to the user is the issue of which parts of a document to index.

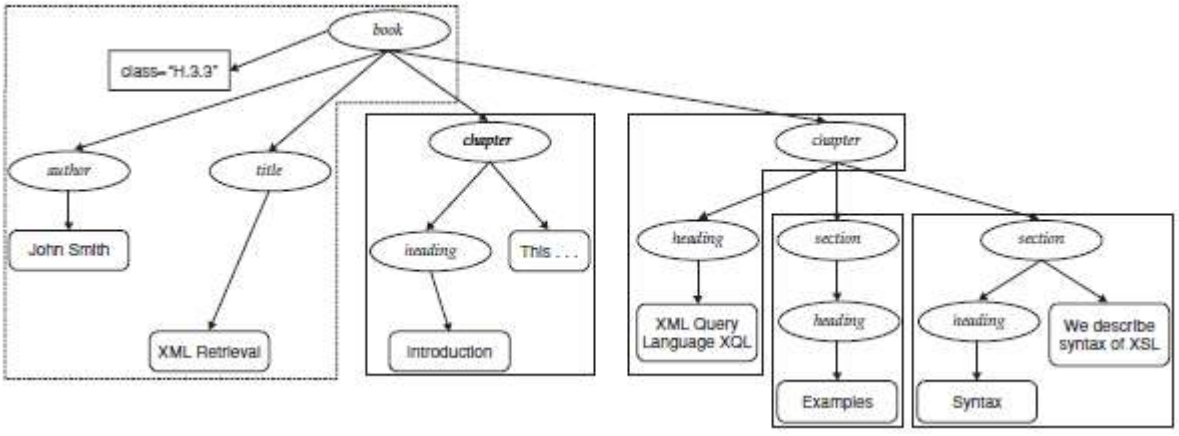


Figure . Partitioning an XML document into non-overlapping indexing units.

- In unstructured retrieval, it is usually clear what the right document unit is: files on your desktop, email messages, web pages on the web etc. In structured retrieval, there are a number of different approaches to defining the indexing unit.
- One approach is to group nodes into non-overlapping pseudo documents as shown in Figure. In the example, books, chapters and sections have been designated to be indexing units, but without overlap. For example, the leftmost dashed indexing unit contains only

those parts of the tree dominated by *book* that are not already part of other indexing units. The disadvantage of this approach is that pseudo documents may not make sense to the user because they are not coherent units. For instance, the leftmost indexing unit in Figure merges three disparate elements, the *class*, *author* and *title* elements.

- We can also use one of the largest elements as the indexing unit, for example, the *book* element in a collection of books or the *play* element for Shakespeare's works. We can then post process search results to find for each book or play the sub element that is the best hit.
- For example, the query Macbeth's castle may return the play *Macbeth*, which we can then post process to identify act I, scene vii as the best-matching sub element. Unfortunately, this two stage retrieval process fails to return the best sub element for many queries because the relevance of a whole book is often not a good predictor of the relevance of small sub elements within it.
- Instead of retrieving large units and identifying sub elements (top down), we can also search all leaves, select the most relevant ones and then extend them to larger units in post processing (bottom up). For the query Macbeth's castle in Figure 10.1, we would retrieve the title *Macbeth's castle* in the first pass and then decide in a post processing step whether to return the title, the scene, the act or the play. This approach has a similar problem as the last one:
- The relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in.

- The least restrictive approach is to index all elements. This is also problematic.
- Many XML elements are not meaningful search results, e.g., typographical elements like definitely or an ISBN number which cannot be interpreted without context. Also, indexing all elements means that search results will be highly redundant. For the query Macbeth's castle and the document in Figure, we would return all of the *play*, *act*, *scene* and *title* elements on the path between the root node and Macbeth's castle.
- The leaf node would then occur four times in the result set, once directly and three times as part of other elements. We call elements that are contained within each other *nested*. Returning redundant nested elements in a list of returned hits is not very user-friendly.
- Because of the redundancy caused by nested elements it is common to restrict the set of elements that are eligible to be returned. Restriction strategies include:

✓ Discard all small elements
✓ Discard all element types that users do not look at (this requires a working xml retrieval system that logs this information)
✓ Discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
✓ Only keep element types that a system designer or librarian has deemed to be useful search results

- In most of these approaches, result sets will still contain nested elements. Thus, we may want to remove some elements in a post processing step to reduce redundancy. Alternatively, we can collapse several nested elements in the results list and use *highlighting* of query terms to draw the user's attention to the relevant passages.
- If query terms are highlighted, then scanning a medium-sized element (e.g., a section) takes little more time than scanning a small sub element (e.g., a paragraph). Thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.

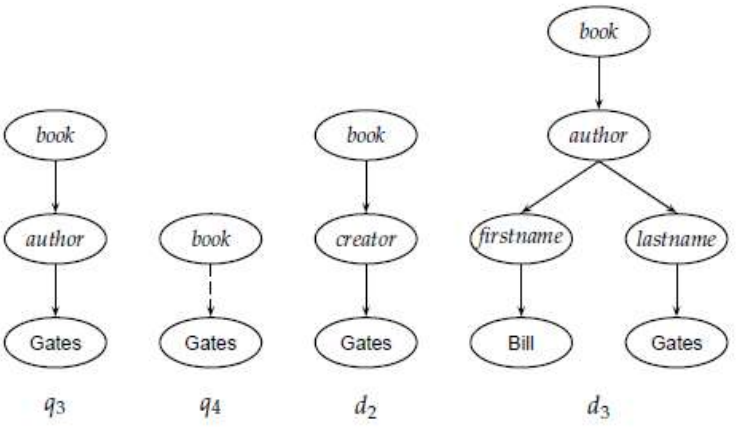
- An additional advantage of this approach is that the paragraph is presented together with its context (i.e., the embedding section). This context may be helpful in interpreting the paragraph (e.g., the source of the information reported) even if the paragraph on its own satisfies the query.
- If the user knows the schema of the collection and is able to specify the desired type of element, then the problem of redundancy is alleviated as few nested elements have the same type. But as we discussed in the introduction, users often don't know what the name of an element in the collection is (Is the Vatican a *country* or a *city*?) or they may not know how to compose structured queries at all.
- 

Figure . Schema heterogeneity: intervening nodes and mismatched names.

- A challenge in XML retrieval related to nesting is that we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf) statistics. For example, the term Gates under the node *author* is unrelated to an occurrence under a content node like *section* if used to refer to the plural of gate. It makes little sense to compute a single document frequency for Gates in this example.
- One solution is to compute idf for XML-context/term pairs, e.g., to compute different idf weights for author#"Gates" and section#"Gates".
- Unfortunately, this scheme will run into sparse data problems – that is, many XML-context pairs occur too rarely to reliably estimate df . A compromise is only to consider the parent node *x* of the term and not the rest of the path from the root to *x* to distinguish contexts. There are still conflations of contexts that are harmful in this scheme. For instance, we do not distinguish names of authors and names of corporations if both have the parent node *name*. But most important distinctions, like the example contrast author#"Gates" vs. section#"Gates", will be respected.
- In many cases, several different XML schemas occur in a collection since the XML documents in an IR application often come from more than one source. This phenomenon is called *schema heterogeneity* or *schema diversity* and presents yet another challenge. As illustrated in Figure comparable elements may have different names: *creator* in *d*2 vs. *author* in *d*3.
- In other cases, the structural organization of the schemas may be different: Author names are direct descendants of the node *author* in *q*3, but there are the intervening nodes

firstname and *lastname* in *d*3. If we employ strict matching of trees, then *q*3 will retrieve neither *d*2 nor *d*3 although both documents are relevant. Some form of approximate matching of element names in combination with semi-automatic matching of different document structures can help here. Human editing of correspondences of elements in different schemas will usually do better than automatic methods.

- Schema heterogeneity is one reason for query-document mismatches like *q*3/*d*2 and *q*3/*d*3. Another reason is that users often are not familiar with the element names and the structure of the schemas of collections they search as mentioned. This poses a challenge for interface design in XML retrieval.
- Ideally, the user interface should expose the tree structure of the collection and allow users to specify the elements they are querying. If we take this approach, then designing the query interface in structured retrieval is more complex than a search box for keyword queries in unstructured retrieval.
- We can also support the user by interpreting all parent-child relationships in queries as descendant relationships with any number of intervening nodes allowed. We call such queries *extended queries*. The tree in Figure and *q*4 in Figure are examples of extended queries. We show edges that are interpreted as descendant relationships as dashed arrows. In *q*4, a dashed arrow connects *book* and Gates. As a pseudo-XPath notation for *q*4, we adopt book//#"Gates": a book that somewhere in its structure contains the word Gates where the path from the *book* node to Gates can be arbitrarily long.
- The pseudo-XPath notation for the extended query that in addition specifies that Gates occurs in a *section* of the *book* is book//section//#"Gates". It is convenient for users to be able to issue such extended queries without having to specify the exact structural configuration in which a query term should occur – either because they do not care about the exact configuration or because they do not know enough about the schema of the collection to be able to specify it.
- In Figure, the user is looking for a chapter entitled FFT (*q*5). Suppose there is no such chapter in the collection, but that there are references to books on FFT (*d*4). A reference to a book on FFT is not exactly what the user is looking for, but it is better than returning nothing.
- Extended queries do not help here. The extended query *q*6 also returns nothing. This is a case where we may want to interpret the structural constraints specified in the query as hints as opposed to as strict conditions. Users prefer a relaxed interpretation of structural constraints: Elements that do not meet structural constraints perfectly should be ranked lower, but they should not be omitted from search results.

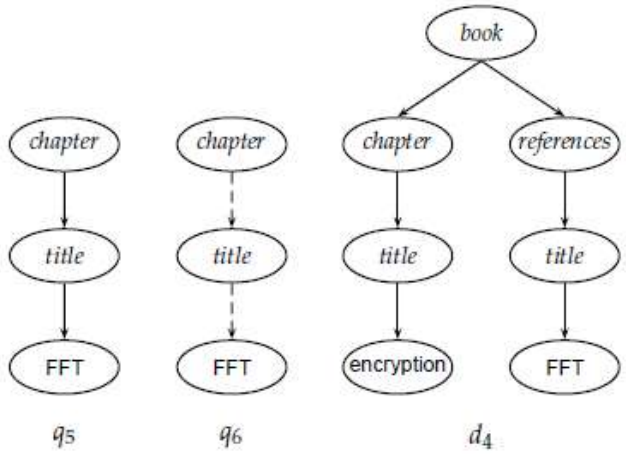


Figure . A structural mismatch between two queries and a document.

A vector space model for XML retrieval:

- In this section, we present a simple vector space model for XML retrieval. It is not intended to be a complete description of a state-of-the-art system. Instead, we want to give the reader a flavor of how documents can be represented and retrieved in XML retrieval.
- To take account of structure in retrieval, we want a book entitled *Julius Caesar* to be a match for *q*1 and no match (or a lower weighted match) for *q*2. In unstructured retrieval, there would be a single dimension of the vector space for Caesar. In XML retrieval, we must separate the title word Caesar from the author name Caesar. One way of doing this is to have each dimension of the vector space encode a word together with its position within the XML tree.

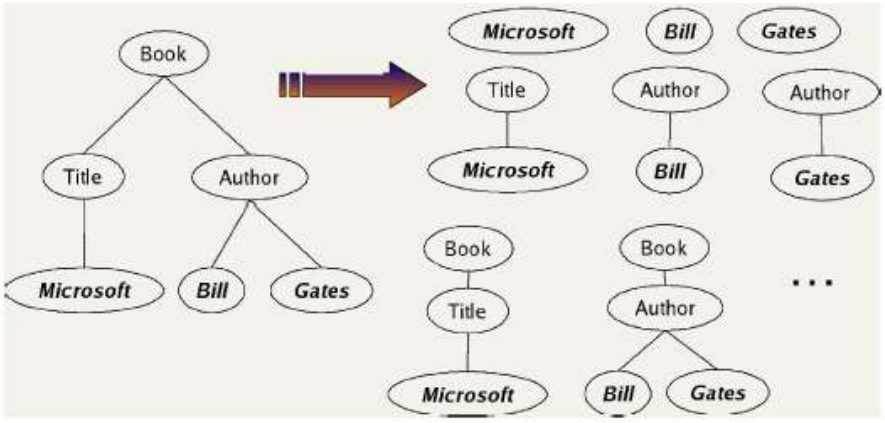


Figure . A mapping of an XML document (left) to a set of lexicalized subtrees (right).

- Figure illustrates this representation. We first take each text node (which in our setup is always a leaf) and break it into multiple nodes, one for each word. So the leaf node Bill Gates is split into two leaves Bill and Gates.

- Next we define the dimensions of the vector space to be *lexicalized subtrees* of documents – subtrees that contain at least one vocabulary term. A subset of these possible lexicalized subtrees is shown in the figure, but there are others – e.g., the subtree corresponding to the whole document with the leaf node Gates removed. We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them.
- This means that we can use the vector space formalism for XML retrieval. The main difference is that the dimensions of vector space in unstructured retrieval are vocabulary terms whereas they are lexicalized subtrees in XML retrieval.
- There is a tradeoff between the dimensionality of the space and accuracy of query results. If we trivially restrict dimensions to vocabulary terms, then we have a standard vector space retrieval system that will retrieve many documents that do not match the structure of the query (e.g., Gates in the title as opposed to the author element).
- If we create a separate dimension for each lexicalized subtree occurring in the collection, the dimensionality of the space becomes too large. A compromise is to index all paths that end in a single vocabulary term, in other words, all XML-context/term pairs.
- We call such an XML-context/term pair a *structural term* and denote it by h*c*, *t*i: a pair of XML-context *c* and vocabulary term *t*. The document in Figure has nine structural terms. Seven are shown (e.g., "Bill" and Author#"Bill") and two are not shown: /Book/Author#"Bill" and /Book/Author#"Gates". The tree with the leaves Bill and Gates is a lexicalized subtree that is not a structural term. We use the previously introduced pseudo-XPath notation for structural terms.
- Users are bad at remembering details about the schema and at constructing queries that comply with the schema. We will therefore interpret all queries as extended queries – that is, there can be an arbitrary number of intervening nodes in the document for any parent child node pair in the query. For example, we interpret *q*5 in Figure as *q*6.
- But we still prefer documents that match the query structure closely by inserting fewer additional nodes. We ensure that retrieval results respect this preference by computing a weight for each match. A simple measure of the similarity of a path *cq* in a query and a path *cd* in a document is the following *context resemblance* function CR:

$$\mathrm{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

- Where |*cq*| and |*cd*| are the number of nodes in the query path and document path, respectively, and *cq* matches *cd* iff we can transform *cq* into *cd* by inserting additional nodes. Two examples from Figure are CR(*cq*4 , *cd*2) = 3/4 = 0.75 and CR(*cq*4 , *cd*3) = 3/5 = 0.6 where *cq*4 , *cd*2 and *cd*3 are the relevant paths from top to leaf node in *q*4, *d*2 and *d*3, respectively. The value of CR(*cq*, *cd*) is 1.0 if *q* and *d* are identical.
- The final score for a document is computed as a variant of the cosine measure , which we call SIMNOMERGE for reasons that will become clear shortly. SIMNOMERGE is defined as follows:

$$\text{SIMNOMERGE}(q,d) = \sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q,t,c_k) \frac{\text{weight}(d,t,c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d,t,c)}}$$

- where V is the vocabulary of non-structural terms; B is the set of all XML contexts; and weight(q, t, c) and weight(d, t, c) are the weights of term t in XML context c in query q and document d, respectively. We compute the weights using one of the weightings such as $\text{idf}_t \cdot \text{wf}_{t,d}$. The inverse document frequency idf_t depends on which elements we use to compute df$t$. The similarity measure SIMNOMERGE(q, d) is not a true cosine measure since its value can be larger than 1.0 . We divide by $\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d,t,c)}$ to normalize for document length . We have omitted query length normalization to simplify the formula. It has no effect on ranking since, for a given query, the normalize $\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(q,t,c)}$ is the same for all documents.
- The algorithm for computing for all documents in the coll qection is shown in Figure . The array *normalizer* in Figure contains $\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d,t,c)}$ from Equation for each document.
- We give an example of how SIMNOMERGE computes query-document similarities in Figure . $\langle c_1, t \rangle$ is one of the structural terms in the query. We successively retrieve all postings lists for structural terms hc', ti with the same vocabulary term t. Three example postings lists are shown. For the first one, we have $\text{CR}(c_1, c_1) = 1.0$ since the two contexts are identical.

```
SCOREDOCUMENTSWITHSIMNOMERGE(q, B, V, N, normalizer)
 1  for n ← 1 to N
 2  do score[n] ← 0
 3  for each ⟨c_q, t⟩ ∈ q
 4  do w_q ← WEIGHT(q, t, c_q)
 5     for each c ∈ B
 6     do if CR(c_q, c) > 0
 7           then postings ← GETPOSTINGS(⟨c, t⟩)
 8                for each posting ∈ postings
 9                do x ← CR(c_q, c) * w_q * weight(posting)
10                   score[docID(posting)] += x
11  for n ← 1 to N
12  do score[n] ← score[n]/normalizer[n]
13  return score
```

Figure . The algorithm for scoring documents with SIMNOMERGE.

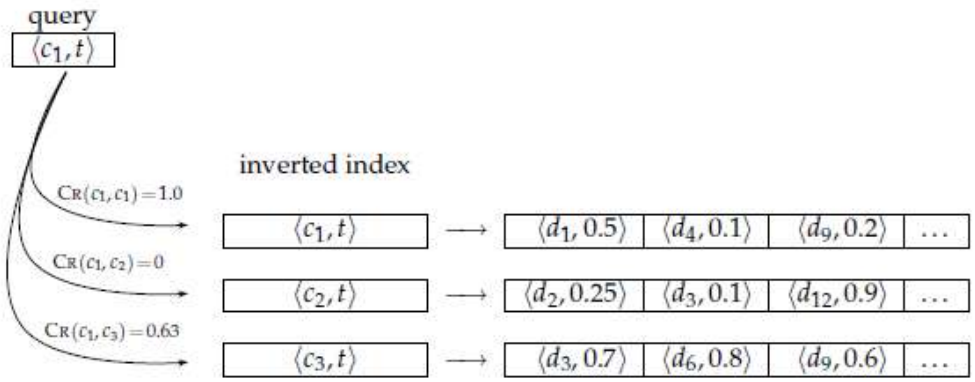


Figure . Scoring of a query with one structural term in SIMNOMERGE.

- The next context has no context resemblance with $c_1: C_R(c_1, c_2) = 0$ and the corresponding postings list is ignored. The context match of *c*1 with *c*3 is 0.63>0 and it will be processed. In this example, the highest ranking document is *d*9 with a similarity of 1.0 ×0.2 +0.63× 0.6 = 0.578. To simplify the figure, the query weight of $\langle c_1, t \rangle$ is assumed to be 1.0.
- The query-document similarity function in Figure is called SIMNOMERGE because different XML contexts are kept separate for the purpose of weighting. An alternative similarity function is SIMMERGE which relaxes the matching conditions of query and document further in the following three ways.
- We collect the statistics used for computing weight(*q*, *t*, *c*) and weight(*d*, *t*, *c*) from *all* contexts that have a non-zero resemblance to *c* (as opposed to just from *c* as in SIMNOMERGE). For instance, for computing the document frequency of the structural term atl#"recognition", we also count occurrences of recognition in XML contexts fm/atl, article//atl etc.

✓ We modify Equation by merging all structural terms in the document that have a non-zero context resemblance to a given query structural term. For example, the contexts /play/act/scene/title and /play/title in the document will be merged when matching against the query term /play/title#"Macbeth".

✓ The context resemblance function is further relaxed: Contexts have a nonzero resemblance in many cases where the definition of CR in Equation returns 0.

✓ These three changes alleviate the problem of sparse term statistics and increase the robustness of the matching function against poorly posed structural queries. The evaluation of SIMNOMERGE and SIMMERGE in the next section shows that the relaxed matching conditions of SIMMERGE increase the effectiveness of XML retrieval.

Evaluation of XML retrieval:

- The premier venue for research on XML retrieval is the INEX (**IN***itiative for the* **E***valuation of* **X***ML retrieval*) program, a collaborative effort that has produced reference collections, sets of queries, and relevance judgments. A yearly INEX meeting is held to present and discuss research results.

12,107	number of documents
494 MB	size
1995–2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

Table . INEX 2002 collection statistics.

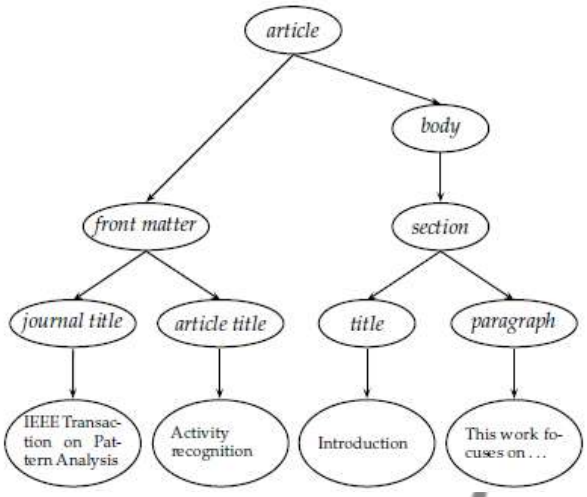


Figure . Simplified schema of the documents in the INEX collection.

- The INEX 2002 collection consisted of about 12,000 articles from IEEE journals. We give collection statistics in Table and show part of the schema of the collection in Figure . The IEEE journal collection was expanded in 2005. Since 2006 INEX uses the much larger English Wikipedia as a test collection.
- The relevance of documents is judged by human assessors using the methodology, appropriately modified for structured documents as we will discuss shortly.
- Two types of information needs or topics in INEX are content-only or CO topics and content-and-structure (CAS) topics. *CO* topics are regular key word queries as in unstructured information retrieval. *CAS topics* have structural constraints in addition to keywords. We already encountered an example of a CAS topic in Figure. The keywords in this case are summer and holidays and the structural constraints specify that the keywords occur in a section that in turn is part of an article and that this article has an embedded year attribute with value 2001 or 2002.
- Since CAS queries have both structural and content criteria, relevance assessments are more complicated than in unstructured retrieval. INEX 2002 defined component coverage and topical relevance as orthogonal dimensions of relevance. The *component coverage* dimension evaluates whether the element retrieved is "structurally" correct, i.e., neither too low nor too high in the tree. We distinguish four cases:

✓ Exact coverage (E). The information sought is the main topic of the component and the component is a meaningful unit of information.

- ✓ Too small (S). The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information.
- ✓ Too large (L). The information sought is present in the component, but is not the main topic.
- ✓ No coverage (N). The information sought is not a topic of the component.

- The *topical relevance* dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0). Components are judged on both dimensions and the judgments are then combined into a digit-letter code. 2S is a fairly relevant component that is too small and 3E is a highly relevant component that has exact coverage. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination 3N is not possible.
- The relevance-coverage combinations are quantized as follows:

$$\mathrm{Q}(rel, cov) = \begin{cases} 1.00 & \text{if} \quad (rel, cov) = \text{3E} \\ 0.75 & \text{if} \quad (rel, cov) \in \{\text{2E}, \text{3L}\} \\ 0.50 & \text{if} \quad (rel, cov) \in \{\text{1E}, \text{2L}, \text{2S}\} \\ 0.25 & \text{if} \quad (rel, cov) \in \{\text{1S}, \text{1L}\} \\ 0.00 & \text{if} \quad (rel, cov) = \text{0N} \end{cases}$$

- This evaluation scheme takes account of the fact that binary relevance judgments, which are standard in unstructured information retrieval , are not appropriate for XML retrieval. A 2S component provides incomplete information and may be difficult to interpret without more context, but it does answer the query partially. The quantization function **Q** does not impose a binary choice relevant/nonrelevant and instead allows us to grade the component as partially relevant.

algorithm	average precision
SIMNOMERGE	0.242
SIMMERGE	0.271

Table. INEX 2002 results of the vector space model in Section 10.3 for content and-structure (CAS) queries and the quantization function **Q**.

The number of relevant components in a retrieved set *A* of components can then be computed as:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} \mathrm{Q}(rel(c), cov(c))$$

- As an approximation, the standard definitions of precision, recall and F from Chapter 8 can be applied to this modified definition of relevant items retrieved, with some subtleties because we sum graded as opposed to binary relevance assessments.
- One flaw of measuring relevance this way is that overlap is not accounted for. This problem is worse in XML retrieval because of the problem of multiple nested elements occurring in a search result . Much of the recent focus at INEX has been on developing algorithms and evaluation measures that return non-redundant results lists and evaluate them properly.

- Table shows two INEX 2002 runs of the vector space system . The better run is the SIMMERGE run, which incorporates few structural constraints and mostly relies on keyword matching.
- SIMMERGE's median average precision (where the median is with respect to average precision numbers over topics) is only 0.147. Effectiveness in XML retrieval is often lower than in unstructured retrieval since XML retrieval is harder. Instead of just finding a document, we have to find the subpart of a document that is most relevant to the query.
- Also, XML retrieval effectiveness – when evaluated as described here – can be lower than unstructured retrieval effectiveness on a standard evaluation because graded judgments lower measured performance.
- Consider a system that returns a document with graded relevance 0.6 and binary relevance 1 at the top of the retrieved list. Then, interpolated precision at 0.00 recall is 1.0 on a binary evaluation, but can be as low as 0.6 on a graded evaluation.

	content only	full structure	improvement
precision at 5	0.2000	0.3265	63.3%
precision at 10	0.1820	0.2531	39.1%
precision at 20	0.1700	0.1796	5.6%
precision at 30	0.1527	0.1531	0.3%

Table . A comparison of content-only and full-structure search in INEX 2003/2004.

- Table gives us a sense of the typical performance of XML retrieval, but it does not compare structured with unstructured retrieval. Table directly shows the effect of using structure in retrieval. The results are for a language-model-based system that is evaluated on a subset of CAS topics from INEX 2003 and 2004. The evaluation metric is precision at *k*. The discretization function used for the evaluation maps highly relevant elements (roughly corresponding to the 3E elements defined for **Q**) to 1 and all other elements to 0. The content only system treats queries and documents as unstructured bags of words.
- The full-structure model ranks elements that satisfy structural constraints higher than elements that do not. For instance, for the query in Figure an element that contains the phrase summer holidays in a *section* will be rated higher than one that contains it in an *abstract*.
- The table shows that structure helps increase precision at the top of the results list. There is a large increase of precision at $k = 5$ and at $k = 10$. There is almost no improvement at $k = 30$. These results demonstrate the benefits of structured retrieval. Structured retrieval imposes additional constraints on what to return and documents that pass the structural filter are more likely to be relevant. Recall may suffer because some relevant documents will be filtered out, but for precision-oriented tasks structured retrieval is superior.

Text-centric vs. data-centric XML retrieval:

- In the type of structured retrieval we cover in this chapter, XML structure serves as a framework within which we match the text of the query with the text of the XML documents. This exemplifies a system that is optimized for *text-centric XML*. While both text and structure are important, we give higher priority to text. We do this by adapting unstructured retrieval methods to handling additional structural constraints. The premise of our approach is that XML document retrieval is characterized by

(i) Long text fields (e.g., sections of a document)
(ii) In exact matching
(iii) Relevance-ranked results.

- Relational databases do not deal well with this use case.
- In contrast, *data-centric XML* mainly encodes numerical and non-text attribute value data. When querying data-centric XML, we want to impose exact match conditions in most cases. This puts the emphasis on the structural aspects of XML documents and queries. An example is:

Find employees whose salary is the same this month as it was 12 months ago.

- This query requires no ranking. It is purely structural and an exact matching of the salaries in the two time periods is probably sufficient to meet the user's information need.
- Text-centric approaches are appropriate for data that are essentially text documents, marked up as XML to capture document structure. This is becoming a de facto standard for publishing text databases since most text documents have some form of interesting structure – paragraphs, sections, footnotes etc. Examples include assembly manuals, issues of journals, Shakespeare's collected works and newswire articles.
- Data-centric approaches are commonly used for data collections with complex structures that mainly contain non-text data. A text-centric retrieval engine will have a hard time with proteomic data in bioinformatics or with the representation of a city map that (together with street names and other textual descriptions) forms a navigational database.
- Two other types of queries that are difficult to handle in a text-centric structured retrieval model are joins and ordering constraints. The query for employees with unchanged salary requires a join. The following query imposes an ordering constraint:
- Retrieve the chapter of the book *Introduction to algorithms* that follows the chapter *Binomial heaps*.
- This query relies on the ordering of elements in XML – in this case the ordering of chapter elements underneath the book node. There are powerful query languages for XML that can handle numerical attributes, joins and ordering constraints.
- The best known of these is XQuery, a language proposed for standardization by the W3C. It is designed to be broadly applicable in all areas where XML is used. Due to its complexity, it is challenging to implement an XQuery-based ranked retrieval system with the performance characteristics that users have come to expect in information retrieval. This is currently one of the most active areas of research in XML retrieval.
- Relational databases are better equipped to handle many structural constraints, particularly joins (but ordering is also difficult in a database framework – the tuples of a relation in the relational calculus are not ordered). For this reason, most data-centric XML retrieval systems are extensions of relational databases.
- If text fields are short, exact matching meets user needs and retrieval results in form of unordered sets are acceptable, then using a relational database for XML retrieval is appropriate.

3.13. META CRAWLERS

- Unlike search engines, metacrawlers don't crawl the web themselves to build listings. Instead, they allow searches to be sent to several search engines all at once. The results are then blended together onto one page. Below are some of the major metacrawlers.

Dogpile

Popular metasearch site owned by InfoSpace that sends a search to a customizable list of search engines, directories and specialty search sites, then displays results from each search engine individually. Winner of Best Meta Search Engine award from Search Engine Watch for 2003.

Vivisimo

Enter a search term, and Vivismo will not only pull back matching responses from major search engines but also automatically organize the pages into categories. Slick and easy to use. Vivisimo won second place for Best Meta Search Engine in the 2003 Search Engine Watch awards and winner in 2002.

Kartoo

If you like the idea of seeing your web results visually, this meta search site shows the results with sites being interconnected by keywords. Honorable mention for Best Meta Search Engine award from Search Engine Watch in 2002.

Mamma

Founded in 1996, Mamma.com is one of the oldest meta search engines on the web. Mamma searches against a variety of major crawlers, directories and specialty search sites. The service also provides a paid listings option for advertisers, Mamma Classifieds. Mamma was an honorable mention for Best Meta Search Engine in the 2003 Search Engine Watch awards.

SurfWax

Searches against major engines or provides those who open free accounts the ability to chose from a list of hundreds. Using the "SiteSnaps" feature, you can preview any page in the results and see where your terms appear in the document. Allows results or documents to be saved for future use. Honorable mention for Best Meta Search Engine award from Search Engine Watch in 2002.

Clusty

Clusty, from Vivisimo, presents both standard web search results and Vivisimo's dynamic clusters that automatically categorize results. Clusty allows you to use Vivisimo's dynamic clustering technology on ten different types of web content including material from the web, image, weblog and shopping databases. You can access each type of search by simply clicking a tab directly above the search box.

CurryGuide

Meta search engine for the US and several European countries, as well as in various subject areas. Has ability to save your results for easy rerunning at a future point.

Excite

Formerly a crawled-based search engine, Excite was acquired by InfoSpace in 2002 and uses the same underlying technology as the other InfoSpace meta search engines, but maintains its own portal features.

Fazzle

Fazzle offers a highly flexible and customizable interface to a wide variety of information sources, ranging from general web results to specialized search resources in a number of subject specific categories. Formerly called SearchOnline.

Gimenei

Gimenei queries an undisclosed number of search engines and removes duplicates from results. Its most useful feature is an advanced search form that allows you to limit your search to a specific country.

IceRocket

Meta search engine with thumbnail displays. The Quick View display, similar to what WiseNut has long offered, is cool. The service queries WiseNut, Yahoo, Teoma and then somewhat repetitively also includes Yahoo-powered MSN, AltaVista and AllTheWeb. Disclosure of search sources within the actual search results is not done, sadly. Makes it hard to know exactly where the results are coming from.

Info.com

Info.com provides results from 14 search engines and pay-per-click directories, including Google, Ask Jeeves, Yahoo, Kanoodle, LookSmart, About, Overture and Open Directory. Also offers shopping, news, eBay, audio and video search, as well as a number of other interesting features.

InfoGrid

In a compact format, InfoGrid provides direct links to major search sites and topical web sites in different categories. Meta search and news searching is also offered.

Infonetware RealTerm Search

This site is primarily designed to demonstrate classification technology from Infogistics. It's a meta search engine, and it does topical classification of results, like Vivisimo. However, it is unique in that you can select several different topics, then "drill down" to see results from all of them, rather than being restricted to the results from only one topic.

Ithaki

Ithaki is probably the most "global" of all meta search engines, available in 14 languages and

offering more than 35 different categories for limiting your search. In addition, Ithaki offers country specific search, querying only local search engines rather than the regional versions of the major search engines.

Ixquick

Meta search engine that ranks results based on the number of "top 10" rankings a site receives from the various search engines.

iZito

iZito is a meta search engine with a clever feature. Click on any listing you are interested in using the P icon next to the listing title. That "parks" the listing into your to do list. Click on the P tab, and you can see all the pages you've culled. It's an easy, handy way to make a custom result set. Also interesting is the ability to show listings in up to three columns across the screen, letting you see more results at once.

Jux2

This search result comparison tool is cool. It allows you to search two major search engines at the same time, then see results that are found on both first, followed by results found on only one of them next. The small overlap visual tool displayed is great. I used to make examples like this to explain search engine overlap and why one search engine may not cover everything. Now I have an easy dynamic way to do this. The stats link at the bottom of the home page provides more visuals.

Meceoo

Meta search with the ability to create an "exclusion list" to block pages from particular web sites being included. For example, want to meta search only against .org sites? French version also offered.

MetaCrawler

One of the oldest meta search services, MetaCrawler began in July 1995 at the University of Washington. MetaCrawler was purchased by InfoSpace, an online content provider, in Feb. 97.

MetaEureka

Search against several major search engines and paid listings services. Offers a nice option to see Alexa info about pages that are listed.

CS6007 INFORMATION RETRIEVAL

UNIT-III
2-MARKS:

1. What is the use of web graph?
   The relationship between sites and pages indicated by these hyperlinks gives rise to what is called a Web graph.

2. What is the difference between static and dynamic pages?
   - The HTML for a static Web page is assumed to be generated in advance of any request, placed on disk, and transferred to the browser or Web crawler on demand. The home page of an organization is a typical example of a static page.
   - A dynamic Web page is assumed to be generated at the time the request is made, with the contents of the page partially determined by the details of the request. A search engine result page (SERP) is a typical example of a dynamic Web page for which the user's query itself helps to determine the content.

3. What is the Hidden Web mean?
   - Many pages are part of the so-called "hidden" or "invisible" or "deep" Web. This hidden Web includes pages that have no links referencing them, those that are protected by passwords, and those that are available only by querying a digital library or database. Although these pages can contain valuable content, they are difficult or impossible for a Web crawler to locate.

4. What are the categories of web queries?
   Web queries are classified into three categories reflecting users' apparent intent, as follows:

   1) Navigational query:

   - The intent behind a navigational query is to locate a specific page or site on the Web. For example, a user intending to locate the home page of the CNN news network might enter the query "CNN". A navigational query usually has a single correct result. However, this correct result may vary from user to user.

   2) Informational query:
      - A user issuing an informational query is interested in learning something about a particular topic and has a lesser regard for the source of the information, provided it is reliable.

   3) Transactional query:
      - A user issuing a transactional query intends to interact with a Web site once she finds it. This interaction may involve activities such as playing games, purchasing items, booking travel, or downloading images, music, and videos. This category may also include queries seeking services such as maps and weather, provided the user is not searching for a specific site providing that service.

5. What is meant by paid placement?
    - A far more effective and profitable form of advertising for search engines, pioneered by GoTo.com (that was renamed to Overture in 2001 and acquired by Yahoo in 2003), is called paid placement also known as sponsored search.

6. What is the difference between organic list and paid list?
    - An organic list, which contains the free unbiased results, displayed according to the search engine's ranking algorithm, and
    - A sponsored list, which is paid for by advertising managed with the aid of an online auction mechanism. This method of payment is called pay per click (PPC), also known as cost per click (CPC), since payment is made by the advertiser each time a user clicks on the link in the sponsored listing.

7. How will you measure size of web?
   If the sets A1,A2, . . . represent the sets of pages indexed by each of these search engines, a lower bound on the size of the indexable Web is the size of the union of these sets $|\cup A_i|$.

8. Define Cloaking.
    - The spammer's web server returns different pages depending on whether the http request comes from a web search engine's crawler or from a human user's browser. The former causes the web page to be indexed by the search engine under misleading keywords.
    - When the user searches for these keywords and elects to view the page, he receives a web page that has altogether different content than that indexed by the search engine. Such deception of search indexers is unknown in the traditional world of IR; it stems from the fact that the relationship between page publishers and web search engines is not completely collaborative.

9. What do you mean by Doorway page?
   A doorway page contains text and metadata carefully chosen to rank highly on selected search keywords. When a browser requests the doorway page, it is redirected to a page containing content of a more commercial nature. More complex spamming techniques involve manipulation of the metadata related to a page including the links into a web page.

10. What is the use of search engine optimizers?
    Search engine optimizers, or engine optimizers SEOs, to provide consultancy services for clients who seek to have their web pages rank highly on selected keywords.

11. What are two primary goals of a search engine?
    The two primary goals of a search engine are:
    - Effectiveness (quality): We want to be able to retrieve the most relevant set of documents possible for a query.
    - Efficiency (speed): We want to process queries from users as quickly as possible

12. What are major components of indexing process?
    - Text acquisition
    - Text transformation
    - Index creation.

13. What are major components of querying process?

    - User interaction
    - Ranking
    - Evaluation

14. Define web crawling.

    Web crawling is the process by which we gather pages from the Web to index them and support a search engine.

15. What are Features of a crawler?

    1) Robustness:
        - The Web contains servers that create spider traps, which are generators of web pages that mislead crawlers into getting stuck fetching an infinite number of pages in a particular domain. Crawlers must be designed to be resilient to such traps. Not all such traps are malicious; some are the inadvertent side effect of faulty website development.
    2) Politeness:
        - Web servers have both implicit and explicit policies regulating the rate at which a crawler can visit them. These politeness policies must be respected.
    3) Distributed:
        - The crawler should have the ability to execute in a distributed fashion across multiple machines.
    4) Scalable:
        - The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.
    5) Performance and efficiency:
        - The crawl system should make efficient use of various system resources including processor, storage, and network bandwidth.
    6) 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.
    7) Freshness:
        - In many applications, the crawler should operate in continuous mode: It should obtain fresh copies of previously fetched pages. A search engine crawler should 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.
    8) 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.

16. What are the ways of index implementations?

Two ways of index implementations are:
i) Partitioning by terms, also known as global index organization
ii) Partitioning by document, also known as local index organization.

17. What is the use of focused crawling?

A less expensive approach is focused, or topical, crawling. A focused crawler attempts to download only those pages that are about a particular topic.
Focused crawlers rely on the fact that pages about a topic tend to have links to other pages on the same topic. If this were perfectly true, it would be possible to start a crawl at one on-topic page, and then crawl all pages on that topic just by following links from a single root page. In practice, a number of popular pages for a specific topic are typically used as seeds.
Focused crawlers require some automatic means for determining whether a page is about a particular topic.

18. What are the Benefits of compression?

1) Need less disk space.
2) Increased use of caching.
3) Faster transfer of data from disk to memory.

19. State heap's law.

Heaps' law estimates vocabulary size as a function of collection size:

$$M = kT^b$$

Where T is the number of tokens in the collection. Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$.

20. What is the use of front coding?

One source of redundancy in the dictionary is the fact that consecutive entries in an alphabetically sorted list share common prefixes. This observation leads to front coding.
A sequence of terms with identical prefix is encoded by marking the end of the prefix with * and replacing it with ◇ in subsequent terms.

21. What are the challenges of XML retrieval?

1) Structured or XML retrieval: users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval.

2) Central notion for indexing and ranking in IR: documents unit or indexing unit.
   - In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages on the web etc.
   - In structured retrieval, there are four main different approaches to defining the indexing unit
     - non-overlapping pseudo documents
     - top down

- bottom up
- all

3) Because of the redundancy caused by the nested elements it is common to restrict the set of elements eligible for retrieval.

22. What is the use of meta-crawler?

Meta-crawler don't crawl the web themselves to build listings. Instead, they allow searches to be sent to several search engines all at once. The results are then blended together onto one page.

UNIT- 3

PART-B

1. Explain the web search architecture and explain about it. (12)
2. Discuss about paid placement.(4)
3. Elaborate about the various ways of doing index compression. (10)
4. Explain the structure of Web and draw the web graph. (6)
5. Elaborate the method of measuring the size of the web? (8**)**
6. Explain crawling architecture and explain about it. (12)
7. Discuss about Focused crawling. (4)
8. Discuss about search engine optimization and spam. (8)
9. Elaborate the features of crawler. (4)
10. Explain the method of near duplicate detection? (8)
11. Explain in detail about XML retrieval.(16)
12. Discuss about meta-crawler.(6)

UNIT IV WEB RETRIEVAL AND WEB CRAWLING

The Web – Search Engine Architectures – Cluster based Architecture – Distributed Architectures – Search Engine Ranking – Link based Ranking – Simple Ranking Functions – Learning to Rank – Evaluations — Search Engine Ranking – Search Engine User Interaction – Browsing – Applications of a Web Crawler – Taxonomy – Architecture and Implementation – Scheduling Algorithms – Evaluation..

4.1. LINK ANALYSIS

- Hyperlinks are used for ranking web search results. Such link analysis is one of many factors considered by web search engines in computing a composite score for a web page on any given query.
- Link analysis for web search has intellectual antecedents in the field of citation analysis, aspects of which overlap with an area known as bibliometrics. These disciplines seek to quantify the influence of scholarly articles by analyzing the pattern of citations among them. Much as citations represent the conferral of authority from a scholarly article to others, link analysis on the Web treats hyperlinks from a web page to another as a conferral of authority.
- Clearly, not every citation or hyperlink implies such authority conferral; for this reason, simply measuring the quality of a web page by the number of in-links (citations from other pages) is not robust enough. For instance, one may contrive to set up multiple web pages pointing to a target web page, with the intent of artificially boosting the latter's tally of in-links. This phenomenon is referred to as ***link spam***.
- Nevertheless, the phenomenon of citation is prevalent and dependable enough that it is feasible for web search engines to derive useful signals for ranking from more sophisticated link analysis. Link analysis also proves to be a useful indicator of what page(s) to crawl next while crawling the web.

The Web as a graph:

Link analysis builds on two intuitions.
1. The anchor text pointing to page B is a good description of page B.
2. The hyperlink from A to B represents an endorsement of page B, by the creator of page A.
    - This is not always the case; for instance, many links among pages within a single website stem from the user of a common template. For instance, most corporate websites have a pointer from every page to a page containing a copyright notice – this is clearly not an endorsement. Accordingly, implementations of link analysis algorithms typically discount such "internal" links.

Anchor text and the web graph

- The following fragment of HTML code from a web page shows a hyperlink pointing to the home page of the Journal of the ACM:

Journal of the ACM.

- In this case, the link points to the page www.acm.org/jacm/ and the anchor text is *Journal of the ACM*. Clearly, in this example the anchor is descriptive of the target page. But then the target page (B = http://www.acm.org/jacm/) itself contains the same description as well as considerable additional information on the journal. So what use is the anchor text?
- The Web is full of instances where the page B does not provide an accurate description of itself. In many cases, this is a matter of how the publishers of page B choose to present themselves; this is especially common with corporate web pages, where a web presence is a marketing statement.
- For example, at the time of the writing of this book the home page of the IBM corporation (www.ibm.com) did not contain the term computer anywhere in its HTML code, despite the fact that IBM is widely viewed as the world's largest computer maker. Similarly, the HTML code for the home page of Yahoo! (www.yahoo.com) does not at this time contain the word portal.
- Thus, there is often a gap between the terms in a web page and how web users would describe that web page. Consequently, web searchers need not use the terms in a page to query for it.
- In addition, many web pages are rich in graphics and images, and/or embed their text in these images; in such cases, the HTML parsing performed when crawling will not extract text that is useful for indexing these pages.
- The fact that the anchors of many hyperlinks pointing to www.ibm.com include the word computer can be exploited by web search engines. For instance, the anchor text terms can be included as terms under which to index the target web page. Thus, the postings for the term computer would include the document www.ibm.com and that for the term portal would include the document www.yahoo.com, using a special indicator to show that these terms occur as anchor (rather than in-page) text.
- As with in-page terms, anchor text terms are generally weighted based on frequency, with a penalty for terms that occur very often (the most common terms in anchor text across the Web are Click and here) using methods very similar to idf. The actual weighting of terms is determined by machine-learned scoring; current web search engines appear to assign a substantial weighting to anchor text terms.
- The use of anchor text has some interesting side effects. Searching for big blue on most web search engines returns the home page of the IBM corporation as the top hit; this is consistent with the popular nickname that many people use to refer to IBM.
- On the other hand, there have been (and continue to be) many instances where derogatory anchor text such as evil empire leads to somewhat unexpected results on querying for these terms on web search engines. This phenomenon has been exploited in orchestrated campaigns against specific sites. Such orchestrated anchor text may be a form of spamming; a website can create misleading anchor text pointing to itself to boost its ranking on selected query terms.
- Detecting and combating such systematic abuse of anchor text is another form of spam detection that web search engines perform. The window of text surrounding anchor text (sometimes referred to as *extended anchor text*) is often usable in the same manner as anchor text itself; consider for instance the fragment of web text there is good discussion

of vedic scripture <a>here. This has been considered in a number of settings and the useful width of this window has been studied;

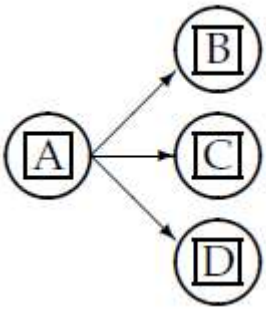


Figure . The random surfer at node A proceeds with probability 1/3 to each of B, C, and D.

4.2. PAGERANK:

- We now focus on **scoring and ranking** measures derived from the link structure alone. Our first technique for link analysis assigns to every node in the PageRank web graph a numerical score between 0 and 1, known as its *PageRank*. The PageRank of a node depends on the link structure of the web graph.
- Given a query, a web search engine computes a composite score for each web page that combines hundreds of features such as cosine **similarity** and term proximity together with the PageRank score. This composite score is used to provide a ranked list of results for the query.
- Consider a random surfer who begins at a web page (a node of the web graph) and executes a random walk on the Web as follows. At each time step, the surfer proceeds from his current page A to a randomly chosen web page that A hyperlinks to. Figure shows the surfer at a node A, out of which there are three hyperlinks to nodes B, C, and D; the surfer proceeds at the next time step to one of these three nodes, with equal probabilities 1/3.
- As the surfer proceeds in this random walk from node to node, he visits some nodes more often than others; intuitively, these are nodes with many links coming in from other frequently visited nodes. The idea behind Page- Rank is that pages visited more often in this walk are more important.
- What if the current location of the surfer, the node A, has no out-links? To address this we introduce an additional operation for our random surfer: the *teleport* operation. In the teleport operation, the surfer jumps from a node to any other node in the web graph. This could happen because he types an address into the URL bar of his browser. The destination of a teleport operation is modeled as being chosen uniformly at random from all web pages.
- In other words, if *N* is the total number of nodes in the web graph, the teleport operation takes the surfer to each node with probability 1/*N*. The surfer would also teleport to his present position with probability 1/*N*.
- In assigning a PageRank score to each node of the web graph, we use the teleport operation in two ways:

(i) When at a node with no out-links, the surfer invokes the teleport operation.

(ii) At any node that has outgoing links, the surfer invokes the teleport operation with probability $0 < \alpha < 1$ and the standard random walk (follow an out-link chosen uniformly at random as in Figure) with probability $1 - \alpha$, where α is a fixed parameter chosen in advance. Typically, α might be 0.1.

- When the surfer follows this combined process (random walk plus teleport) he visits each node v of the web graph a fixed fraction of the time $\pi(v)$ that depends on

(i) The structure of the web graph and

(ii) The value of α. We call this value $\pi(v)$ the PageRank of v .

Markov chains:

- A Markov chain is a **discrete-time stochastic process**, a process that occurs in a series of time steps in each of which a random choice is made. A Markov chain consists of *N states*. Each web page will correspond to a state in the Markov chain we will formulate.
- A Markov chain is characterized by an $N \times N$ *transition probability matrix* P each of whose entries is in the interval [0, 1]; the entries in each row of P add up to 1.
- The Markov chain can be in one of the N states at any given time-step; then, the entry $Pi\ j$ tells us the probability that the state at the next time-step is j, conditioned on the current state being i. Each entry $Pi\ j$ is known as a transition probability and depends only on the current state i; this is known as the Markov property. Thus, by the Markov property,

$$\forall i, j, P_{ij} \in [0, 1]$$

And

$$\forall i, \sum_{j=1}^{N} P_{ij} = 1.$$

- A matrix with non-negative entries that satisfies Equation is known stochastic as a *stochastic matrix*. A key property of a stochastic matrix is that it has a matrix *principal left eigenvector* corresponding to its largest eigenvalue, which is 1.
- In a Markov chain, the probability distribution of next states for a Markov chain depends only on the current state, and not on how the Markov chain arrived at the current state. Figure shows a simple Markov chain with three states. From the middle state A, we proceed with (equal) probabilities of 0.5 to either B or C. From either B or C, we proceed with probability 1 to A. The transition probability matrix of this Markov chain is then

$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

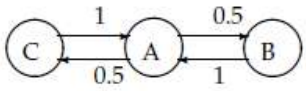


Figure . A simple Markov chain with three states; the numbers on the links indicate the transition probabilities.

- A Markov chain's probability distribution over its states may be viewed as probability a *probability vector*, a vector all of whose entries are in the interval [0, 1], and vector the entries add up to 1. An *N*-dimensional probability vector each of whose components corresponds to one of the *N* states of a Markov chain can be viewed as a probability distribution over its states. For our simple Markov chain of Figure , the probability vector would have three components that sum to 1.
- We can view a random surfer on the web graph as a Markov chain, with one state for each web page, and each transition probability representing the probability of moving from one web page to another.
- The teleport operation contributes to these transition probabilities. The adjacency matrix *A* of the web graph is defined as follows: if there is a hyperlink from page *i* to page *j*, then $A_{ij} = 1$, otherwise $A_{ij} = 0$.We can readily derive the transition probability matrix *P* for our Markov chain from the $N \times N$ matrix *A*. If a row of *A* has no 1's, then divide each element by $1/N$. For all other rows proceed as follows:

1. Divide each 1 in *A* by the number of 1s in its row. Thus, if there is a row with three 1s, then each of them is replaced by 1/3.
2. Multiply the resulting matrix by $1 - \alpha$.
3. Add α/N to every entry of the resulting matrix, to obtain *P*.

- We can depict the probability distribution of the surfer's position at any time by a probability vector $\vec{x}$. At $t = 0$ the surfer may begin at a state whose corresponding entry in $\vec{x}$ is 1 while all others are zero. By definition, the surfer's distribution at $t = 1$ is given by the probability vector $\vec{x}P$; at $t = 2$ by $(\vec{x}P)P = \vec{x}P^2$, and so on.
- We can thus compute the surfer's distribution over the states at any time, given only the initial distribution and the transition probability matrix *P*.
- If a Markov chain is allowed to run for many time steps, each state is visited at a (different) frequency that depends on the structure of the Markov chain. In our running analogy, the surfer visits certain web pages (say, popular news home pages) more often than other pages. We now make this intuition precise, establishing conditions under which such the visit frequency converges to fixed, steady-state quantity. Following this, we set the Page- Rank of each node *v* to this steady-state visit frequency and show how it can be computed.

Definition: Ergodic A Markov chain is said to be *ergodic* if there exists a positive integer T_0 such that for all pairs of states *i, j* in the Markov chain, if it is started at time 0 in state *i* then for all $t > T_0$, the probability of being in state *j* at time *t* is greater than 0.

- For a Markov chain to be ergodic, two technical conditions are required of the its states and the nonzero transition probabilities; these conditions are known as *irreducibility* and *aperiodicity*. Informally, the first ensures that there is a sequence of transitions of nonzero probability from any state to any other, while the latter ensures that the states are not partitioned into sets such that all state transitions occur cyclically from one set to another.

Theorem. *For any ergodicMarkov chain, there is a unique* steady-state *probability vector* $\vec{\pi}$ *that is the principal left eigenvector of P, such that if* $\eta(i, t)$ *is the number of visits to state i in t steps, then*

$$\lim_{t\to\infty} \frac{\eta(i,t)}{t} = \pi(i),$$

where $\pi(i) > 0$ *is the steady-state probability for state i.*

It follows from Theorem that the random walk with teleporting results in a unique distribution of steady-state probabilities over the states of the induced Markov chain. This steady-state probability for a state is the PageRank of the corresponding web page.

The PageRank computation:

- The left eigenvectors of the transition probability matrix *P* are *N*-vectors $\vec{\pi}$ such that

$$\vec{\pi} P = \lambda \vec{\pi}.$$

- The *N* entries in the principal eigenvector $\vec{\pi}$ are the steady-state probabilities of the random walk with teleporting, and thus the PageRank values for the corresponding web pages. We may interpret Equation as follows:
- If $\vec{\pi}$ is the probability distribution of the surfer across the web pages, he remains in the steady-state distribution $\vec{\pi}$. Given that $\vec{\pi}$ is the steady-state distribution, we have that $\pi P = 1\pi$, so 1 is an eigenvalue of P. Thus, if we were to compute the principal left eigenvector of the matrix *P* – the one with eigenvalue 1 – we would have computed the PageRank values.
- There are many algorithms available for computing left eigenvectors; the references at the end of Chapter 18 and the present chapter are a guide to these. We give here a rather elementary method, sometimes known as *power iteration*. If $\vec{\pi}$ is the initial distribution over the states, then the distribution at time *t* is $\vec{\pi}P^t$.
- As *t* grows large, we would expect that the distribution$\vec{\pi}P^{t\ 2}$ is very similar to the distribution $\vec{\pi}P^{\ t+1}$; for large *t* we would expect the Markov chain to attain its steady state. By Theorem, this is independent of the initial distribution$\vec{\pi}$.

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_2$	1/3	1/3	1/3
$\vec{x}_3$	1/4	1/2	1/4
$\vec{x}_4$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

Figure . The sequence of probability vectors.

- The power iteration method simulates the surfer's walk: Begin at a state and run the walk for a large number of steps *t*, keeping track of the visit frequencies for each of the states. After a large number of steps *t*, these frequencies "settle down" so that the variation in the computed frequencies is below some predetermined threshold.
- We declare these tabulated frequencies to be the PageRank values. We consider the web graph with $\alpha = 0.5$. The transition probability matrix of the surfer's walk with teleportation is then

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}$$

- Imagine that the surfer starts in state 1, corresponding to the initial probability distribution vector $\vec{x}_0 = (1\ 0\ 0)$. Then, after one step the distribution is

$$\vec{x}_0 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \vec{x}_1.$$

- After two steps it is

$$\vec{x}_1 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \vec{x}_2.$$

- Continuing in this fashion gives a sequence of probability vectors as shown in Figure .
- Continuing for several steps, we see that the distribution converges to the steady state of $\vec{x} = (5/18\ 4/9\ 5/18)$.
- In this simple example, we may directly calculate this steady-state probability distribution by observing the symmetry of the Markov chain: States 1 and 3 are symmetric, as evident from the fact that the first and third rows of the transition probability matrix in Equation are identical. Postulating, then, that they both have the same steady-state probability and denoting this probability by *p*, we know that the steady-state distribution is of the form $\vec{x} = (p\ 1 - 2p\ p)$. Now, using the identity $\vec{x} = \vec{x} P$, we solve a simple linear equation to obtain $p = 5/18$ and consequently, $\vec{x} = (5/18\ 4/9\ 5/18)$.

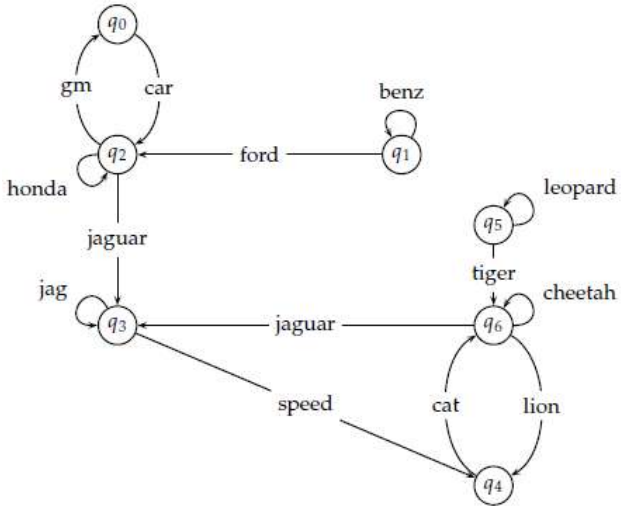


Figure. A small web graph. Arcs are annotated with the word that occurs in the anchor text of the corresponding link.

- The PageRank values of pages (and the implicit ordering among them) are independent of any query a user might pose; PageRank is thus a query independent measure of the static quality of each web page .
- On the other hand, the relative ordering of pages should, intuitively, depend on the query being served. For this reason, search engines use static quality measures such as PageRank as just one of many factors in scoring a web page on a query. Indeed, the relative contribution of PageRank to the overall score may again be determined by machine-learned scoring.

✎ **Example :** Consider the graph in Figure. For a teleportation rate of 0.14 its (stochastic) transition probability matrix is:

0.02	0.02	0.88	0.02	0.02	0.02	0.02
0.02	0.45	0.45	0.02	0.02	0.02	0.02
0.31	0.02	0.31	0.31	0.02	0.02	0.02
0.02	0.02	0.02	0.45	0.45	0.02	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.88
0.02	0.02	0.02	0.02	0.02	0.45	0.45
0.02	0.02	0.02	0.31	0.31	0.02	0.31

The PageRank vector of this matrix is:

$$\vec{x} = (0.05 \quad 0.04 \quad 0.11 \quad 0.25 \quad 0.21 \quad 0.04 \quad 0.31)$$

- Observe that in Figure, $q2$, $q3$, $q4$ and $q6$ are the nodes with at least two in-links. Of these, $q2$ has the lowest PageRank since the random walk tends to drift out of the top part of the graph – the walker can only return there through teleportation.

Topic-specific PageRank:

- Thus far, we have discussed the PageRank computation with a teleport operation in which the surfer jumps to a web page chosen uniformly at random.
- We now consider teleporting to a random web page chosen *non uniformly*. In doing so, we are able to derive PageRank values tailored to particular interests.
- For instance, a sports fan might wish that pages on sports be ranked higher than non-sports pages. Suppose that web pages on sports are "near" one another in the web graph. Then, a random surfer who frequently finds himself on random sports pages is likely (in the course of the random walk) to spend most of his time at sports pages, so that the steady-state distribution of sports pages is boosted.
- Suppose our random surfer, endowed with a teleport operation as before, teleports to *a random web page on the topic of sports* instead of teleporting to a uniformly chosen random web page.
- We will not focus on how we collect all web pages on the topic of sports; in fact, we only need a nonzero subset *S* of sports-related web pages, so that the teleport operation is feasible. This may be obtained, for instance, from a manually built directory of sports pages such as the open directory project (www.dmoz.org/) or that of Yahoo.
- Provided the set *S* of sports-related pages is nonempty, it follows that there is a nonempty set of web pages $Y \supseteq S$ over which the random walk has a steady-state distribution; let us denote this *sports PageRank* distribution by $\vec{x}_s$. For web pages not in *Y*, we set the PageRank values to 0. We call $\vec{x}_s$ the *topic-specific PageRank* for sports.
- We do not demand that teleporting takes the random surfer to a uniformly chosen sports page; the distribution over teleporting targets *S* could in fact be arbitrary.
- In like manner, we can envision topic-specific PageRank distributions for each of several topics such as science, religion, politics, and so on. Each of these distributions assigns to each web page a PageRank value in the interval [0, 1).
- For a user interested in only a single topic from among these topics, we may invoke the corresponding PageRank distribution when scoring and ranking search results. This gives us the potential of considering settings in which the search engine knows what topic a user is interested in. This may happen because users either explicitly register their interests, or because the system learns by observing each user's behavior over time.

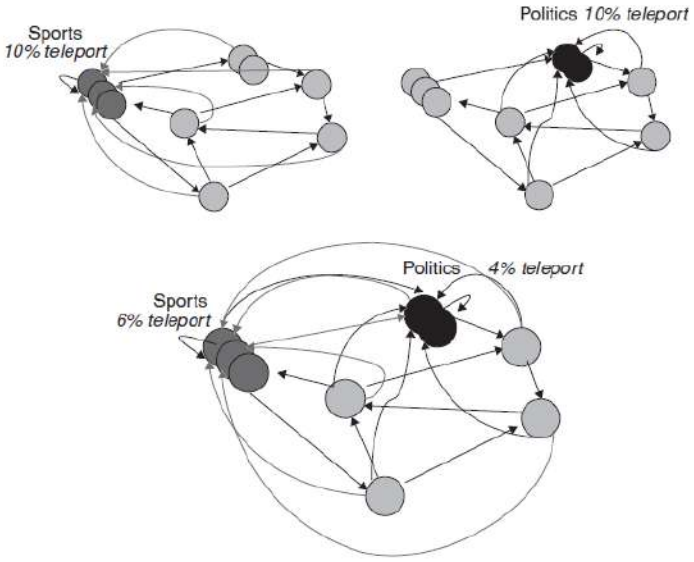


Figure. Topic-specific PageRank. In this example we consider a user whose interests are 60% sports and 40% politics. If the teleportation probability is 10%, this user is modeled as teleporting 6% to sports pages and 4% to politics pages.

- But what if a user is known to have a mixture of interests from multiple topics? For instance, a user may have an interest mixture (or *profile*) that personalized is 60% sports and 40% politics; can we compute a *personalized PageRank* for PageRank this user? At first glance, this appears daunting; how could we possibly compute a different PageRank distribution for each user profile (with, potentially, infinitely many possible profiles)? We can in fact address this provided we assume that an individual's interests can be well-approximated as a linear combination of a small number of topic page distributions.
- A user with this mixture of interests could teleport as follows: Determine first whether to teleport to the set *S* of known sports pages, or to the set of known politics pages.
- This choice is made at random, choosing sports pages 60% of the time and politics pages 40% of the time. Once we choose that a particular teleport step is to (say) a random sports page, we choose a web page in *S* uniformly at random to teleport to. This in turn leads to an ergodic Markov chain with a steady-state distribution that is personalized to this user's preferences over topics.
- Although this idea has intuitive appeal, its implementation appears cumbersome; it seems to demand that for each user, we compute a transition probability matrix and compute its steady-state distribution. We are rescued by the fact that the evolution of the probability distribution over the states of a Markov chain can be viewed as a linear system.
- It is not necessary to compute a PageRank vector for every distinct combination of user interests over topics; the personalized PageRank vector for any user can be expressed as a linear combination of the underlying topic-specific PageRanks. For instance, the personalized PageRank vector for the user whose interests are 60% sports and 40% politics can be computed as

$$0.6\vec{\pi}_s + 0.4\vec{\pi}_p,$$

- Where $\vec{x}_s$ and $\vec{x}_p$ are the topic-specific PageRank vectors for sports and for politics, respectively.

4.3. HUBS AND AUTHORITIES:

- Given a query, every web page is assigned *two* hub scores. One is called its *hub score* and the other its *authority score*.
- For any query, we compute two ranked lists of results rather than one. The ranking of one list is induced by the hub scores and that of the other by the authority scores. This approach stems from a particular insight into the creation of web pages, namely, that there are two primary kinds of web pages useful as results for *broad-topic searches*.
- By a broad topic search we mean an informational query such as "I wish to learn about leukemia." There are authoritative sources of information on the topic; in this case, the National Cancer Institute's page on leukemia would be such a page. We will call such pages *authorities*; in the computation we are about to describe, they are the pages that will emerge with high authority scores.
- On the other hand, there are many pages on the Web that are hand compiled lists of links to authoritative web pages on a specific topic. These *hub* pages are not in themselves authoritative sources of topic-specific information, but rather compilations that someone with an interest in the topic has spent time putting together. The approach is to use these hub pages to discover the authority pages. In the computation we now develop, these hub pages are the pages that will emerge with high hub scores.
- A good hub page is one that points to many good authorities; a good authority page is one that is pointed to by many good hub pages. We thus appear to have a circular definition of hubs and authorities; we will turn this into an iterative computation.
- Suppose that we have a subset of the web containing good hub and authority pages, together with the hyperlinks among them. We will iteratively compute a hub score and an authority score for every web page in this subset.
- For a web page v in our subset of the web, we use $h(v)$ to denote its hub score and $a(v)$ its authority score. Initially, we set $h(v) = a(v) = 1$ for all nodes v. We also denote by $v \mapsto y$ the existence of a hyperlink from v to y.
- The core of the iterative algorithm is a pair of updates to the hub and authority scores of all pages given by Equation, which capture the intuitive notions that good hubs point to good authorities and that good authorities are pointed to by good hubs.

$$h(v) \leftarrow \sum_{v \mapsto y} a(y)$$

$$a(v) \leftarrow \sum_{y \mapsto v} h(y).$$

- Thus, the first line of Equation sets the hub score of page v to the sum of the authority scores of the pages it links to. In other words, if v links to pages with high authority

scores, its hub score increases. The second line plays the reverse role; if page *v* is linked to by good hubs, its authority score increases.

- What happens as we perform these updates iteratively, recomputing hub scores, then new authority scores based on the recomputed hub scores, and so on? Let us recast Equation into matrix–vector form. Let $\vec{h}$ and $\vec{a}$ denote the vectors of all hub and all authority scores respectively, for the pages in our subset of the web graph. Let *A* denote the adjacency matrix of the subset of the web graph that we are dealing with: *A* is a square matrix with one row and one column for each page in the subset. The entry A_{ij} is 1 if there is a hyperlink from page *i* to page *j*, and 0 otherwise. Then, we may write Equation

$$\vec{h} \leftarrow A\vec{a}$$
$$\vec{a} \leftarrow A^T\vec{h},$$

- Where *AT* denotes the transpose of the matrix *A*. Now the right hand side of each line of Equation is a vector that is the left hand side of the other line of Equation . Substituting these into one another, we may rewrite Equation as

$$\vec{h} \leftarrow AA^T\vec{h}$$
$$\vec{a} \leftarrow A^TA\vec{a}.$$

- Now, Equation bears an uncanny resemblance to a pair of eigenvector equations; indeed, if we replace the ←symbols by = symbols and introduce the (unknown) eigenvalue, the first line of Equation becomes the equation for the eigenvectors of AA^T , and the second becomes the equation for the eigenvectors of A^TA:

$$\vec{h} = (1/\lambda_h)AA^T\vec{h}$$
$$\vec{a} = (1/\lambda_a)A^TA\vec{a}.$$

- Here we have used λh to denote the eigen value of AA^T and λa to denote the eigen value of *AT A*.This leads to some key consequences:

1. The iterative updates in Equation, if scaled by the appropriate eigen values, are equivalent to the power iteration method for computing the eigenvectors of AA^T and A^TA. Provided that the principal eigen value of AA^T is unique, the iteratively computed entries of $\vec{h}$ and $\vec{a}$ settle into unique steady-state values determined by the entries of *A* and hence the link structure of the graph.
2. In computing these eigenvector entries, we are not restricted to using the power iteration method; indeed, we could use any fast method for computing the principal eigenvector of a stochastic matrix.

- The resulting computation thus takes the following form:

1. Assemble the target subset of web pages, form the graph induced by their hyperlinks and compute AA^T and A^TA.

2. Compute the principal eigenvectors of AA^T and $A^T A$ to form the vector of hub scores $\vec{h}$ and authority scores $\vec{a}$.
3. Output the top-scoring hubs and the top-scoring authorities.

This method of link analysis is known as *HITS*, which is an acronym for *hyperlink-induced topic search*.

Example: Assuming the query jaguar and double-weighting of links whose anchors contain the query word, the matrix *A* for Figure is as follows:

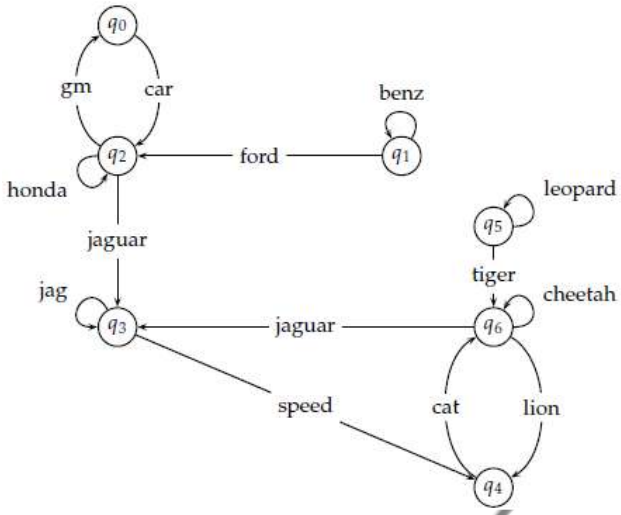


Figure .A small web graph. Arcs are annotated with the word that occurs in the anchor text of the corresponding link.

0	0	1	0	0	0	0
0	1	1	0	0	0	0
1	0	1	2	0	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	1
0	0	0	0	0	1	1
0	0	0	2	1	0	1

The hub and authority vectors are:

$$\vec{h} = (0.03 \quad 0.04 \quad 0.33 \quad 0.18 \quad 0.04 \quad 0.04 \quad 0.35)$$

$$\vec{a} = (0.10 \quad 0.01 \quad 0.12 \quad 0.47 \quad 0.16 \quad 0.01 \quad 0.13)$$

- Here, $q3$ is the main authority – two hubs ($q2$ and $q6$) are pointing to it via highly weighted jaguar links.
- Because the iterative updates captured the intuition of good hubs and good authorities, the high-scoring pages we output would give us good hubs and authorities from the target subset of web pages.

Choosing the subset of the Web:

- In assembling a subset of web pages around a topic such as leukemia, we must cope with the fact that good authority pages may not contain the specific query term leukemia. This is especially true, when an authority page uses its web presence to project a certain marketing image. For instance, many pages on the IBM website are authoritative sources of information on computer hardware, even though these pages may not contain the term computer or hardware. However, a hub compiling computer hardware resources is likely to use these terms and also link to the relevant pages on the IBM website.
- Building on these observations, the following procedure has been suggested for compiling the subset of the Web for which to compute hub and authority scores.

1. Given a query (say leukemia), use a text index to get all pages containing leukemia. Call this the *root set* of pages.
2. Build the *base set* of pages, to include the root set as well as any page that either links to a page in the root set, or is linked to by a page in the root set.

- We then use the base set for computing hub and authority scores. The base set is constructed in this manner for three reasons:

1. A good authority page may not contain the query text (such as computer hardware).
2. If the text query manages to capture a good hub page v_h in the root set, then the inclusion of all pages linked to by any page in the root set will capture all the good authorities linked to by v_h in the base set.
3. Conversely, if the text query manages to capture a good authority page v_a in the root set, then the inclusion of pages points to v_a will bring other good hubs into the base set. In other words, the "expansion" of the root set into the base set enriches the common pool of good hubs and authorities.

- Running HITS across a variety of queries reveals some interesting insights about link analysis. Frequently, the documents that emerge as top hubs and authorities include languages other than the language of the query.
- These pages were presumably drawn into the base set, following the assembly of the root set. Thus, some elements of *cross-language retrieval* (where a query in one language retrieves documents in another) are evident here; interestingly, this cross-language effect resulted purely from link analysis, with no linguistic translation taking place.
- We conclude this section with some notes on implementing this algorithm. The root set consists of all pages matching the text query; in fact, implementations suggest that it suffices to use 200 or so web pages for the root set, rather than all pages matching the text query.
- Any algorithm for computing eigenvectors may be used for computing the hub/authority score vector. In fact, we need not compute the exact values of these scores; it suffices to know the relative values of the scores so that we may identify the top hubs and authorities. To this end, it is possible that a small number of iterations of the power iteration method yields the relative ordering of the top hubs and authorities. Experiments have suggested that in practice, about five iterations of Equation

$$h(v) \leftarrow \sum_{v \mapsto y} a(y)$$

$$a(v) \leftarrow \sum_{y \mapsto v} h(y).$$

- Yield fairly good results.
- Moreover, because the link structure of the web graph is fairly sparse (the average web page links to about ten others), we do not perform these as matrix-vector products but rather as additive updates as in Equation .

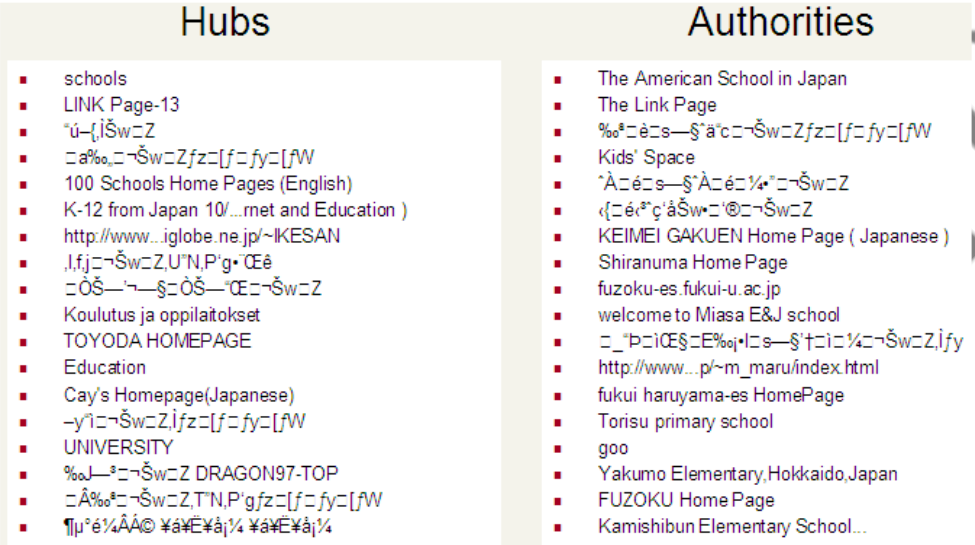


Figure. A sample run of HITS on the query japan elementary schools.

- Figure shows the results of running HITS on the query japan elementary schools. The figure shows the top hubs and authorities; each row lists the title tag from the corresponding HTML page. Because the resulting string is not necessarily in Latin characters, the resulting print is (in many cases) a string of gibberish.
- Each of these corresponds to a web page that does not use Latin characters, in this case very likely pages in Japanese. There also appear to be pages in other non-English languages, which seems surprising given that the query string is in English.
- In fact, this result is emblematic of the functioning of HITS – following the assembly of the root set, the (English) query string is ignored. The base set is likely to contain pages in other languages, for instance if an English-language hub page links to the Japanese-language home pages of Japanese elementary schools. Because the subsequent computation of the top hubs and authorities is entirely link-based, some of these non-English pages will appear among the top hubs and authorities.

4.4. HYPERLINK-INDUCED TOPIC SEARCH (HITS):

- Web pages with high PageRank are considered authoritative, since many other pages point to them. Another type of useful web page is a resource page, known as a *hub*, which has many outgoing links to informative pages. *HITS (hyperlink-induced topic search)* is based on the idea that a good authority is pointed to by good hubs, and a good hub points to good authorities.
- The first step in the implementation of HITS as a search method is to collect a *root set* of web pages for a given input query of say the top 200 hits from a search engine for the query. The root set is then expanded to the *base set* by adding all the pages pointed to by at least one page in the root set, and all the pages that point to at least one page in the root set, limiting this number to say 50, for each page in the root set. To complete the preprocessing step, links between pages within the same web site are removed from the base set, since these are often navigational rather than informational. The resulting set of pages in the base set is called a *focused subgraph*.

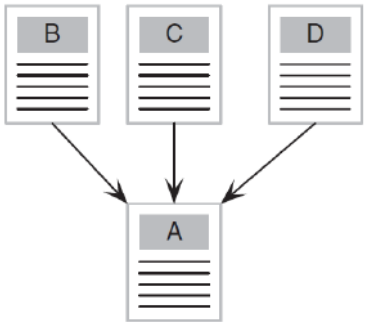


Figure .Page A is an authority.

- HITS uses this subgraph to determine the hubs and authorities for the input query; it typically contains between 1000 and 5000 pages. The algorithm then recalculates two computations several times, one for hubs and the other for authorities, in order to obtain a hub and authority weight for each page in the focused subgraph.
- Starting from an initial set of weights, at each step, the hub score of a page is the sum of the weights of the authorities it points to, and the authority score of a page is the sum of the weights of the hubs that point to it.
- The HITS algorithm normally converges within 20 or so steps. As with the computation of PageRank, the focused subgraph can be expressed as a matrix, and the HITS computation can be expressed via matrix multiplication. Linear algebra shows us that the hub and authority weights must eventually converge.
- The formal statement of the HITS algorithm is given by the two equations, which define the two operations for updating the hub and authority weights of web pages. *A(p)* and *A(q)* represent the authority weights for pages *p* and *q*, *H(p)* and *H(q)* represent the hub weights for these pages, and *F* is the focused subgraph.

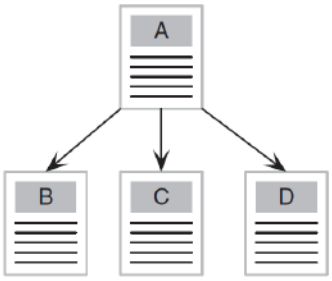


Figure . Page A is a hub.

HITS:

$$A(p) = \sum_{q:q \to p \text{ in } F} H(q) \qquad H(p) = \sum_{q:p \to q \text{ in } F} A(q)$$

- The results from HITS are encouraging. For example, for the query "search engines," it returns the major web search engines as authorities. As far as we know, a variant of HITS has to date been deployed only in a single commercial search engine, namely Teoma (www.teoma.com), whose founders from Rutgers University devised an efficient approximation of the algorithm .
- The lack of wide use of the HITS algorithm is probably due to the lack of a general efficient implementation, since unlike PageRank the hub and authority scores are computed at query time rather than offline. Another factor influencing the lack of deployment of HITS may be due to IBM holding a patent on the algorithm, as Kleinberg was working in IBM at the time.
- Another problem with HITS, called *topic drift* , is the problem that pages in the focused subgraph that were added as a result of expanding the root set may be on a different topic from the query, and as a result the hubs and authorities may also be on a different topic. Topic drift can be dealt with to a large degree by taking into account the link text, when adding pages to the root set.
- Apart from ranking the hub and authority search results, HITS can be used in the following ways:

(i) To find related pages by setting the root set to contain a single page, and up to 200 pages that point to it.

(ii) To help categorize web pages by starting with a root set of pages having a known category.

(iii) To find web communities defined as densely linked focused subgraphs.

(iv) To study citation patterns between documents.

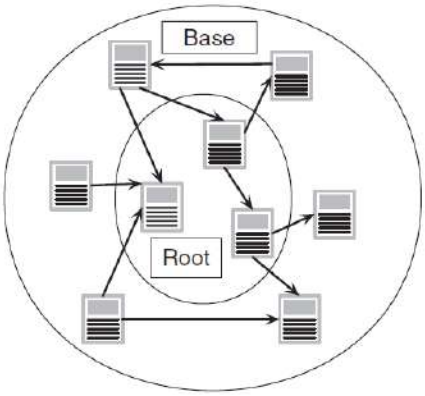


Figure . The base set for the HITS technique.

4.5. SIMILARITY:

- A *similarity measure* is used (rather than a distance or *dissimilarity* measure), so that the documents with the highest scores are the most similar to the query. A number of similarity measures have been proposed and tested for this purpose. The most successful of these is the *cosine correlation* similarity measure.
- The cosine correlation measures the cosine of the angle between the query and the document vectors. When the vectors are *normalized* so that all documents and queries are represented by vectors of equal length, the cosine of the angle between two identical vectors will be 1 (the angle is zero), and for two vectors that do not share any non-zero terms, the cosine will be 0. The cosine measure is defined as:

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

- The numerator of this measure is the sum of the products of the term weights for the matching query and document terms (known as the ***dot product* or inner product)**.
- The denominator normalizes this score by dividing by the product of the lengths of the two vectors. There is no theoretical reason why the cosine correlation should be preferred to other similarity measures, but it does perform somewhat better in evaluations of search quality.

Example:

Consider two documents
*D*1 = (0.5, 0.8, 0.3) and
*D*2 = (0.9, 0.4, 0.2) indexed by three terms, where the numbers represent term weights.
Given the query *Q* = (1.5, 1.0, 0) indexed by the same terms, the cosine measures for the two documents are:

$$Cosine(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}}$$
$$= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87$$

$$Cosine(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}}$$
$$= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97$$

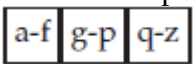
- The second document has a higher score because it has a high weight for the first term, which also has a high weight in the query. Even this simple example shows that ranking based on the vector space model is able to reflect term importance and the number of matching terms, which is not possible in Boolean retrieval.

4.6. MAP-REDUCE:

- Collections are often so large that we cannot perform index construction efficiently on a single machine. This is particularly true of the World Wide Web for which we need large computer *clusters* to construct any reasonably sized web index.
- Web search engines, therefore, use *distributed indexing* algorithms for index construction. The result of the construction process is a distributed index that is partitioned across several machines – either according to term or according to document. Distributed indexing for a term-partitioned index is explained here. Most large search engines prefer a document partitioned index (which can be easily generated from a term-partitioned index).
- The distributed index construction method is an application of *MapReduce*, a general architecture for distributed computing.
- MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on cheap commodity machines or *nodes* that are built from standard parts (processor, memory, disk) as opposed to on a supercomputer with specialized hardware.
- Although hundreds or thousands of machines are available in such clusters, individual machines can fail at any time. One requirement for robust distributed indexing is, therefore, that we divide the work up into chunks that we can easily assign and – master node in case of failure – reassign.
- A *master node* directs the process of assigning and reassigning tasks to individual worker nodes.The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time. The various steps of MapReduce are

shown in Figure and an example on a collection consisting of two documents is shown in next Figure . First, the input data, in splits our case a collection of web pages, are split into *n splits* where the size of the split is chosen to ensure that the work can be distributed evenly (chunks should not be too large) and efficiently (the total number of chunks we need to manage shouldn't be too large); 16 or 64 MB are good sizes in distributed indexing.

- Splits are not preassigned to machines, but are instead assigned by the master node on an ongoing basis: As a machine finishes processing one split, it is assigned the next one. If a machine dies or becomes a laggard due to hardware problems, the split it is working on is simply reassigned to another machine.

- In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of *key-value pairs*. For indexing, a key-value pair has the form (termID,docID). In distributed indexing, the mapping from terms to termIDs is also distributed and therefore more complex than in single-machine indexing.
- A simple solution is to maintain a (perhaps precomputed) mapping for frequent terms that is copied to all nodes and to use terms directly (instead of termIDs) for infrequent terms.We do not address this problem here and assume that all nodes share a consistent term →termID mapping.
- The *map phase* of MapReduce consists of mapping splits of the input data to key-value pairs.Call the machines that execute the map phase *parsers*. Each parser writes its output to local intermediate files, the *segment files* (shown as 

in Figure).

- For the *reduce phase*, we want all values for a given key to be stored close together, so that they can be read and processed quickly. This is achieved by partitioning the keys into *j* term partitions and having the parsers write key value pairs for each term partition into a separate segment file. In Figure, the term partitions are according to first letter: a–f, g–p, q–z, and $j = 3$. (We chose these key ranges for ease of exposition. In general, key ranges need not correspond to contiguous terms or termIDs.)
- The term partitions are defined by the person who operates the indexing system. The parsers then write corresponding segment files, one for each term partition. Each term partition thus corresponds to *r* segments files, where *r* is the number of parsers. For instance, Figure shows three a–f segment files of the a–f partition, corresponding to the three parsers shown in the figure.

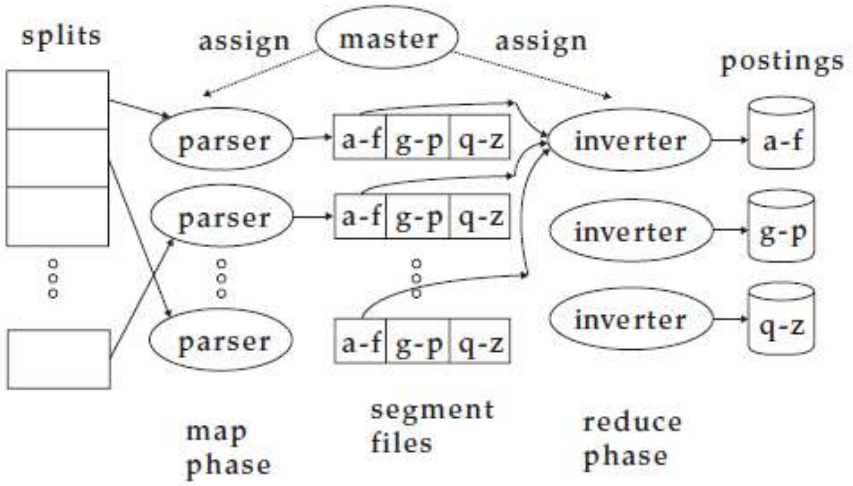


Figure . An example of distributed indexing with MapReduce.

- Collecting all values (here: docIDs) for a given key (here: termID) into one list is the task of the *inverters* in the reduce phase. The master assigns each term partition to a different inverter – and, as in the case of parsers, reassigns term partitions in case of failing or slow inverters. Each term partition (corresponding to *r* segment files, one on each parser) is processed by one inverter.
- We assume here that segment files are of a size that a single machine can handle. Finally, the list of values is sorted for each key and written to the final sorted postings list ("postings" in the figure). (Note that postings in Figure include term frequencies, whereas each posting in the other sections of this chapter is simply a docID without term frequency information.) The data flow is shown for a–f in Figure. This completes the construction of the inverted index.

Schema of map and reduce functions

map: input	→ list(k, v)
reduce: (k,list(v))	→ output

Instantiation of the schema for index construction

map: web collection	→ list(termID, docID)
reduce: (⟨termID$_1$, list(docID)⟩, ⟨termID$_2$, list(docID)⟩, …)	→ (postings_list$_1$, postings_list$_2$, …)

Example for index construction

map: d_2 : C died. d_1 : C came, C c'ed.	→ (⟨C, d_2⟩, ⟨died,d_2⟩, ⟨C,d_1⟩, ⟨came,d_1⟩, ⟨C,d_1⟩, ⟨c'ed,d_1⟩)
reduce: (⟨C,(d_2,d_1,d_1)⟩, ⟨died,(d_2)⟩, ⟨came,(d_1)⟩, ⟨c'ed,(d_1)⟩)	→ (⟨C,(d_1:2,d_2:1)⟩, ⟨died,(d_2:1)⟩, ⟨came,(d_1:1)⟩, ⟨c'ed,(d_1:1)⟩)

Figure . Map and reduce functions in MapReduce. In general, the map function produces a list of key-value pairs. All values for a key are collected into one list in the reduce phase. This list is then processed further. The instantiations of the two functions and an example are shown for index construction. Because the map phase processes documents in a distributed fashion, termID–docID pairs need not be ordered correctly initially as in this example. The example shows terms instead of termIDs for better readability. We abbreviate Caesar as C and conquered as c'ed.

- Parsers and inverters are not separate sets of machines. The master identifies idle machines and assigns tasks to them. The same machine can be a parser in the map phase and an inverter in the reduce phase. And there are often other jobs that run in parallel with index construction, so in between being a parser and an inverter a machine might do some crawling or another unrelated task.
- To minimize write times before inverters reduce the data, each parser writes its segment files to its *local disk*. In the reduce phase, the master communicates to an inverter the locations of the relevant segment files (e.g., of the *r* segment files of the a–f partition). Each segment file only requires one sequential read because all data relevant to a particular inverter were written to a single segment file by the parser. This setup minimizes the amount of network traffic needed during indexing.
- Figure shows the general schema of the MapReduce functions. Input and output are often lists of key-value pairs themselves, so that several MapReduce jobs can run in sequence. In fact, this was the design of the Google indexing system in 2004. What we describe in this section corresponds to only one of five to ten MapReduce operations in that indexing system. Another MapReduce operation transforms the term-partitioned index we just created into a document-partitioned one MapReduce offers a robust and conceptually simple framework for implementing index construction in a distributed environment. By providing a semiautomatic method for splitting index construction into smaller tasks, it can scale to almost arbitrarily large collections, given computer clusters of sufficient size.

4.7. MAP-REDUCE AND HADOOP

- How does a web search engine deliver an efficient and high-quality service to millions of surfers, hitting its servers billions of times a day, and on top of this maintain a fresh index in an environment that is in continuous flux? Well Google can do it, and in an article published in IEEE Micro several of Google's engineers revealed some of the ingredients of their system.
- In 2003, Google's computer cluster combined over 15,000 standard PCs, running in-house developed fault-tolerant software. As of late 2009 this number increased to about 500,000 which is indicator of Google's tremendous growth in computing power and its ability to index and store billions of web pages and an enormous amount of multimedia such as YouTube videos. A standard Google server has 16 GB of RAM and 2 TB of disk space.

- This architecture is much cheaper than using high performance servers and also much more efficient according to Google's engineers. Reliability of the system is attained at the software level by replicating services across many machines and automatically detecting and handling failures. With that many PCs, energy efficiency is another key factor, as power consumption and cooling are critical for this scale of operation.
- How is a Google query served? To provide capacity for the massive query traffic, Google's service consists of several clusters distributed worldwide. Each cluster has thousands of PCs, the distribution protecting Google from catastrophic failures. (As of mid-2008, Google's servers were distributed world wide in 36 data centers.)

- When a query is issued, a cluster is chosen to serve the query according to geographic proximity and once chosen, the query is processed locally on that cluster. Query execution consists of two main phases.
- In the first phase, index servers consult an inverted index to match each keyword in the query to a list of web pages. The set of relevant pages for the query is determined by intersecting these lists, one for each keyword, and then computing the score of each relevant page in order to determine its rank. The result that is returned at this stage is a list of document identifiers.
- In the second phase, the list of document identifiers is used to compile the results page that is delivered to the user's browser. This is handled by document servers and involves computing the title, URL, and summary of each relevant web page. To complete the execution, the spell checking and ad-serving systems are consulted.
- Both phases are highly parallel, as the data, which comprises many petabytes in size (a petabyte is equal to 1000 TB and a terabyte is equal to 1000 GB), is distributed across multiple servers. Overall, Google stores dozens of copies of its search index across its clusters. If part of a cluster is down for some reason, it will use machines from another cluster to keep the service operational, at the cost of reducing the overall capacity of the system. One of the axioms of Google's service is that it must be continuous and efficient at all times.
- Updates to the index are done separately offline and the clusters are then updated one at a time;
- Operating many thousands of PCs incurs significant administration and maintenance costs, but these are manageable due to the small number of applications running on each one of them. Additional costs are incurred for special cooling, which is needed for the clusters, due to the high level of power consumption.
- The central design principles of Google's architecture are purchasing the CPUs with the best price-to-performance ratio, exploiting massive parallelism by distribution and replication, and using highly reliable and scalable software for all the applications, all of which is developed in-house.

- Google's distributed architecture is based on three components:

1) The Google File System (GFS)
2) The mapreduce algorithm
3) The bigtable database system.

1) GFS:

- GFS is a very large, highly used, distributed, and fault-tolerant file system that is designed to work with Google's applications. GFS supports clusters, where each cluster has a single master and multiple chunkservers, and is accessed by multiple clients. Files are divided into chunks of fixed size (64 MB), and the chunkservers store chunks on local disks. For reliability, each chunk is replicated on multiple chunkservers.
- Metadata is maintained by the master, who controls the system-wide activities by communicating to the chunkservers. There is no caching of file data on the chunkservers or the clients. The master's involvement in reads and writes is minimal so as to avoid

bottlenecks. It is expected that the traffic is dominated by reads and appends as opposed to writes that overwrite existing data, making consistency-checking easier.

2) MapReduce:

- MapReduce is a programming model borrowed from functional programming for processing large data sets on a distributed file system such as GFS. A computation is specified in term of *map* and *reduce* functions and the computation is automatically distributed across clusters. The computation can be carried out in parallel as the order of individual map and reduce operations does not effect the output, and although new data is created, existing data is not overwritten.
- The map operation takes a collection of key/value pairs and produces one or more intermediate key/value outputs for each input pair. The reduce operation combines the intermediate key/value pairs to produce a single output value for each key. For example, the input to map could be a URL (key) and document text (value).
- The intermediate output from map could be pairs of word (key) and occurrence (value), which would be one for each word, so in this case, the map splits the document into words and returns a one for each occurrence.
- Reduce will then combine the values from the pairs for each word and return a pair for each word with its count in the document. More than 10,000 MapReduce programs have been implemented in Google, and an average of 100,000 MapReduce jobs are executed daily, processing more than 20 PB of data per day .
- Hadoop (http://hadoop.apache.org) is an open-source distributed file system for very large data sets, inspired by GFS and MapReduce. Yahoo has been a large contributor to Hadoop and has been using it in their applications. Cloudera (www.cloudera.com) is a start-up centered on support and consulting services for enterprise users of Hadoop. It has its own distribution of Hadoop, making it easy for users to install and deploy the software. It has also released a graphical browser-based interface to Hadoop allowing easier management of clusters.

3) BigTable:

- BigTable is a distributed storage system designed by Google to scale reliably for very large data sets, with peta bytes of data served over thousands of machines. BigTable is used widely in Google applications including web indexing, Google Earth, Google Finance, and Google Analytics.
- Data in a BigTable is organized along three dimensions: row keys, column keys, and time stamps, which taken together uniquely identify a cell. A particular kind of BigTable is a web table, where row keys are URLs, column keys are features of web pages, and the cells are page contents. Rows with consecutive keys are organized into tablets. For example, in a web table all the rows from the same domain may form a tablet. Columns are grouped into column families. For example, “anchor” may be a family and each single anchor will be qualified by the name of its referring site, and the cell value will be the anchor text. Time stamps allow stating different versions of the same data.
- BigTable uses GFS to store its data, and can be used with MapReduce to run large-scale parallel computations on tables. BigTable clusters, that is, a set of processes that run the BigTable software, have been in production at Google since mid-2005 and have taken about 7 years to design and develop. As of late 2006, there were more than 60 projects in

Google using BigTable . An opensource version of BigTable, called HBase, has been implemented over Hadoop.

- Another open-source distributed database storage, modeled after BigTable, is Hypertable (www.hypertable.org); Hypertable is used by the Chinese search engine Baidu.
- The BigTable concept arose from the need to store large amounts of distributed data in a wide table format with a large number of columns and sparsely populated rows, that is, where most fields are null, and where the schema, that is, the use of columns, may evolve over time. Moreover, due to the large number of columns, keyword search is the most appropriate when querying a wide table. This is the reason BigTable was designed and implemented in-house rather than over a traditional relational database management system.

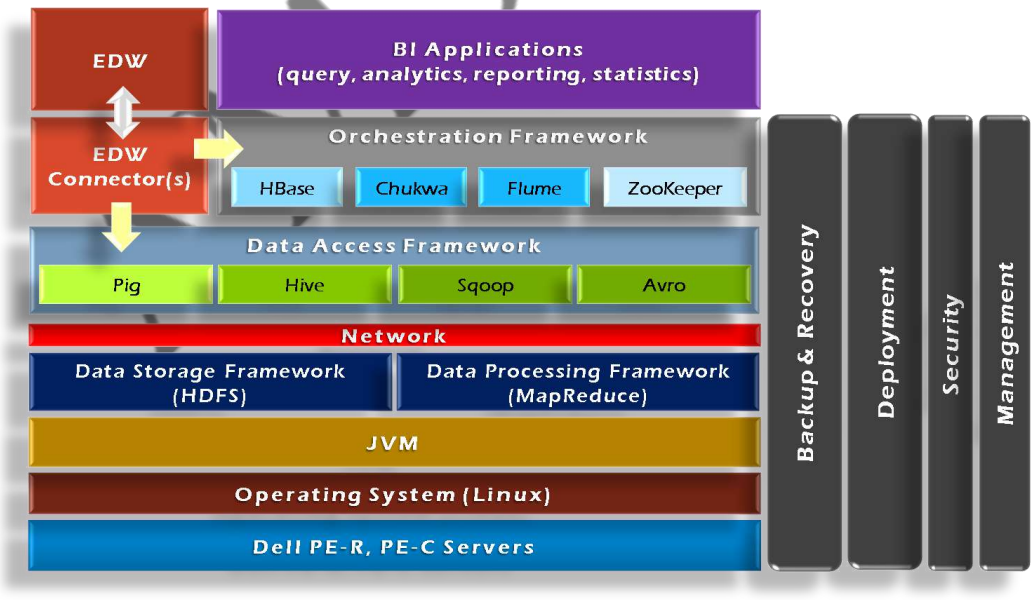
1. Explain in detail about Hadoop.

- Hadoop is an open source implementation of MapReduce coded and released in Java
- (rather than C) by Apache.
- The Hadoop implementation of MapReduce uses the *Hadoop Distributed File System (HDFS)*.
- The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS.
- The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- The following two sections cover the details of these two fundamental layers.
- **HDFS:** HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.
- **HDFS Architecture:** HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).
- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The mapping of blocks to DataNodes is determined by the NameNode.
- The NameNode (master) also manages the file system's metadata and namespace.

- **HDFS Features:**
- **HDFS Fault Tolerance:** One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception.
- **Block replication** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.
- **Replica placement** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS.
- For the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data.

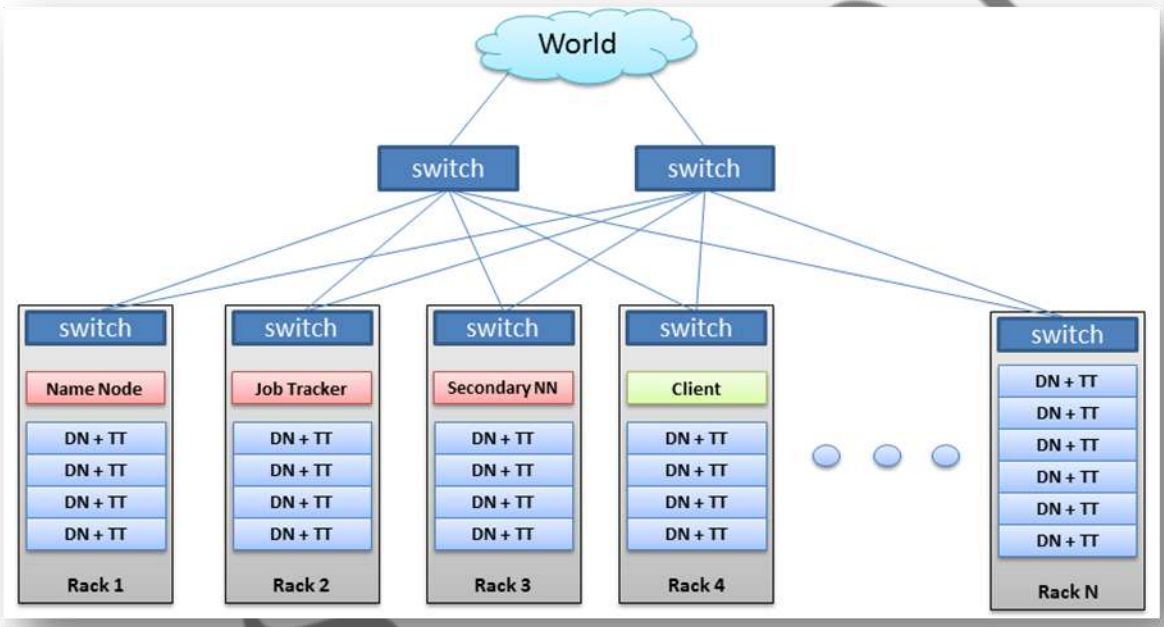
- **Heartbeat and Blockreport messages** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster.
- **HDFS High-Throughput Access to Large Data Sets (Files):** Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to decrease the amount of metadata storage required per file.
- This provides two advantages: The list of blocks per file will shrink as the size of
- individual blocks increases, and by keeping large amounts of data sequentially within
- a block, HDFS provides fast streaming reads of data.
- **HDFS Operation:** The control flow of HDFS operations such as write and read can
- properly highlight roles of the NameNode and DataNodes in the managing operations.
    - **Reading a file** To read a file in HDFS, a user sends an "open" request to the
- NameNode to get the location of file blocks.
- For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas.
- Upon receiving such information, the user calls the *read* function to connect to the
- closest DataNode containing the first block of the file.
- **Writing to a file** To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace.
- The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode.
- Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.
- The steamer then stores the block in the first allocated DataNode.

Hadoop Framework Tools



2. Hadoop's Architecture

- Distributed, with some centralization
- Main nodes of cluster are where most of the computational power and storage of the system lies
- Main nodes run TaskTracker to accept and reply to MapReduce tasks, and also DataNode to store needed blocks closely as possible
- Central control node runs NameNode to keep track of HDFS directories & files, and JobTracker to dispatch compute tasks to TaskTracker
- Written in Java, also supports Python and Ruby



- Hadoop Distributed Filesystem
- Tailored to needs of MapReduce
- Targeted towards many reads of filestreams
- Writes are more costly
- High degree of data replication (3x by default)
- No need for RAID on normal nodes
- Large blocksize (64MB)
- Location awareness of DataNodes in network

NameNode:

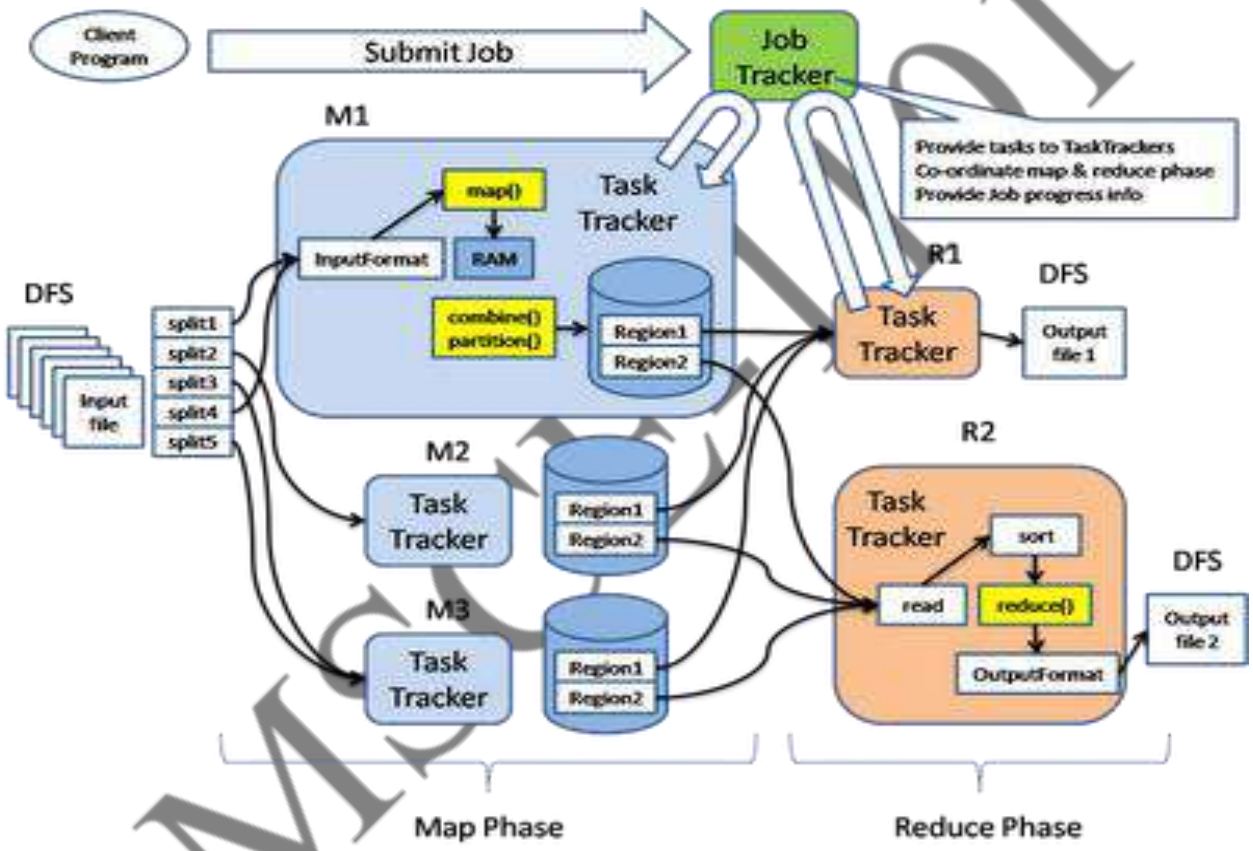
- Stores metadata for the files, like the directory structure of a typical FS.
- The server holding the NameNode instance is quite crucial, as there is only one.

- Transaction log for file deletes/adds, etc. Does not use transactions for whole blocks or file-streams, only metadata.
- Handles creation of more replica blocks when necessary after a DataNode failure

DataNode:

- Stores the actual data in HDFS
- Can run on any underlying filesystem (ext3/4, NTFS, etc)
- Notifies NameNode of what blocks it has

NameNode replicates blocks 2x in local rack, 1x elsewhere



MapReduce Engine:

- JobTracker & TaskTracker
- JobTracker splits up data into smaller tasks("Map") and sends it to the TaskTracker process in each node
- TaskTracker reports back to the JobTracker node and reports on job progress, sends data ("Reduce") or requests new jobs
- None of these components are necessarily limited to using HDFS
- Many other distributed file-systems with quite different architectures work
- Many other software packages besides Hadoop's MapReduce platform make use of HDFS

Hadoop is in use at most organizations that handle big data:

- Yahoo!
- Facebook
- Amazon
- Netflix
- Etc…

Three main applications of Hadoop

- Advertisement (Mining user behavior to generate recommendations)
- Searches (group related documents)
- Security (search for uncommon patterns)

4.8. PERSONALIZATION

- A major deficiency of current search tools is their lack of adaptation to the user's preferences. Although the quality of search has improved dramatically in the last few years and as a result user satisfaction has risen, search engines fall short of understanding an individual user's need and, accordingly, ranking the results for that individual. We believe that to further improve the quality of search results, next generation search engines must adapt to the user's personal needs.
- Search engines collect a huge amount of data from user queries. This together with the use of cookies to identify returning users, and utilities such as the search toolbar, which are installed on users' browsers, put search engines in an excellent position to provide each user with a personalized search interface tailored to the individual needs of that user.
- The first ingredient, that is, the collection of personal search data, is already present, and search engines such as Google have been hard at work to gain our trust so that they can collect this personal data without raising too many privacy concerns. We benefit by getting more powerful tools and the search engine benefits from the increased internet traffic through their site.
- Query log data can be factored into the ranking of search results. The popularity-based approach can be taken a step further by considering each user and their search habits. As part of their technology, the now defunct search engine Direct Hit , looked into personalized search as a specialization of their popularity metric.
- Some degree of personalization can be achieved by using the popularity metric to narrow down users' searches according to demographic information such as age, gender, and geographic location, or even more specifically, according to the users' individual interests and preferences.
- In order to get initial personal data into the search index, the search engine can solicit information from anonymous searchers. When surfers use the search engine, cookies can be used to store their past interaction with the search service, and the inference mechanism can then personalize their query results.
- For example, if a searcher can be identified as a man, a query such as "shoes" may be narrowed down to "men shoes". As another example, men searching on the topic of

"flowers" may show more interest in online floral services, presumably to send flowers to their dear ones, while women may be more interested in information on growing flowers, presumably for home gardening. Two approaches to search engine personalization based on search engine log data may be useful.

- In a **click-based approach** , the user's query and click pairs are used for personalization. The idea is simple. When a user repeats queries over time, he or she will prefer certain pages, that is, those that were more frequently clicked. The downside of this approach is that if a search engine presents the same old pages to the user each time a query is repeated it does not encourage the user to discover new pages. On the other hand, this type of historical information may be quite useful to the user. This approach can be refined by using content similarity to include similar queries and web pages in the personalized results.
- In **a topic-based approach, a** topical ontology is used to identify a user's interests. The ontology should include general topics that are of interest to web surfers such as the top-level topics from the Open Directory. Then a classification technique, such as naive Bayes, needs to be chosen in order to be able to classify the queries that users submit and the pages that they visit. The next step is to identify the user's preferences based on their searches, and finally these preferences can be used to personalize their results, for example, by ranking them according to the learned preferences.
- Care needs to be taken in the choice of which approach to use, since short-term interests may be inconsistent with long-term ones. For example, when issuing a navigational query a user wishes to find a web site, independently of any topical interests. Moreover, taking a long-term approach may introduce noise due to off-topic queries that were submitted and irrelevant web pages that were clicked on along the way.
- A **dynamic and adaptive approach** to personalization must be capable of monitoring the users' activity over time and to infer their interests and preferences as their behavior changes over time. To implement dynamic user profiles, machine learning techniques, such as Bayesian or neural networks, provide a sound basis for improving the machine's understanding of the human behind the machine.
- Personalization is the "next big thing" for search engines, in their quest to improve the quality of results, and that dynamic profiling of users is already on the horizon. We will soon see personalization tools emerging from various research labs. It is hard to predict exactly when personalization will become the norm, as issues such as privacy and scalability,need to be resolved, but it may be rolled out to users gradually.

1) Personalization Versus Customization

- It is important to distinguish between personalization and customization of the user interface.
- Customization involves the layout of the user interface, for example the color scheme to be used, the content displayed on the personalized web page and various other settings. A study of 60 search services published in July 2003 in the online journal First Monday, has revealed that most of the features offered are related to e-mail, business and financial information, entertainment listings, sports, news headlines, and various information tools such as a local weather report and a personal horoscope. At the time of the study, which was in May 2001, only 13% of the services included some personalization features, and My Yahoo (http://my.yahoo.com) had the most extensive list of features;

- All the personalization features offered are based on a static user profile. Information such as the user's address, age, sex, occupation, and topics of interest are recorded and used for personalization purposes. The user can change these parameters at a later date, but otherwise their value will remain constant. The problems with the static approach are that the profile is, generally, incomplete, becomes stale after a certain period of time, and does not take into account the user's continuous interaction with the system. Moreover, users are reluctant to provide the system with profile information.

2) Personalized Results Tool:

- At the Department of Computer Science, Birkbeck, University of London, we have been developing an adaptive personalization tool, which reranks search engine results according to the user's preferences. These include the search terms they have previously entered, the web pages they have been browsing and the categories they have been inspecting. Although this tool is just an experimental prototype it highlights the issues that need to be addressed in the implementation of personalization.
- The Personalized Results Tool (PResTo!) is implemented as a plug-in to the browser rather than being server based . This is a unique feature that bypasses some of the privacy and security issues, which are becoming increasingly important to users, since in the case of PResTo!, the ownership of the software and the personal data generated from searches are in the user's hands.
- A client-side approach is also more efficient for the search engine, since it does not have to manage the user profiles, and thus scalability will not be an issue.
- A downside of the client-side approach from the users' point of view is that the profile is less portable, but a partial solution to this problem may be to store the profile on a local trusted server, which would enable remote access. A downside from the search engines' point of view is that a client-side approach can be used to personalize results from any search engine that the user interacts with, using a single profile applicable to all searching.
- More importantly, not having access to the user profile is contrary to their aim of using personalization as a means of locking users into their search services and being able to provide them with additional personalized services. A compromise between client- and server based personalization, which is amenable to both parties, will have to be found.
- Personalization proceeds as follows: suppose that the user issues a query to his or her favorite search engine. The personalization plug-in detects this and sends the query results, which have been returned to the user's browser, to the personalization engine (on the user's machine), which then reranks the results according to the user's profile and makes its recommendations to the user in a separate window within the browser, alongside the results returned by the search engine.
- As a simple example of PResTo! in action, the keyword "salsa" may be associated with "recipes" or "dancing", or alternatively with "music" or some other aspect of "salsa". A search engine ranking its results for the query "salsa" without personalization, will not have access to the user's profile, and as a consequence will rank web pages about "salsa" only according to its internal criteria.
- Now suppose that the user had previously searched for "recipes" and "dancing", then PResTo! will filter web pages relating to "recipes" or "dancing" to the top of the search

engine's results list. This can be seen in the left-hand side window, generated by the PResTo! prototype when the query "salsa" is typed into Google.

- The numbers in parentheses appearing after the titles of results, show the position where the pages were originally ranked by Google. Adaptive personalization tools need to change in time, as the user's preferences change. So, if the user suddenly becomes interested in "music" and less in "dancing" or "recipes" the system should recognize this shift of interest.

4) Privacy and Scalability:

- Whenever personalization is discussed there are two issues. The first is privacy, and the second is scalability.
- Privacy will always be an issue, and it is therefore essential that all search engines have a clear and upfront privacy policy. Without gaining our trust and showing us the clear benefits of using a tool, which tracks our cyber-movements, any server-based personalization tool is doomed to failure.
- A related issue to privacy is spam. The problem of being bombarded with information we may consider as junk is a serious one, so much so, that users will not sign up to any tool that, although useful, may increase the spam coming in their direction.
- As part of the privacy policy, users will want to be assured that their information remains private, and is not circulated around the Net just to bounce back in the form of unwanted spam. Another related issue is targeted advertising. Through paid placement schemes, which are query sensitive, search engines are already delivering targeted ads that are relevant to user queries. In any case it makes it necessary for search engines to have a clear privacy policy covering these issues.
- Scalability is another problem that search engines getting into the personalization business should consider. If the service is managed through the search engine's servers, then an already stressed system will become even more loaded.
- Storing hundreds of millions of user profiles and updating them on the fly may not be viable, but a partial solution may be to shift some of the processing and data storage to the user's machine. This may also be a very good mechanism for the search engine to lock surfers into using their search service, by providing users with proprietary software that sits on their desktop.

4) Relevance Feedback:

- Personalization is closely related to *relevance feedback*, a technique that was initiated by Rocchio within the SMART retrieval system during the mid-1960s and the early 1970s . The idea behind relevance feedback is simple. When a user such as Archie is presented with a results page for a query, we give him the opportunity to mark each document in the results as being relevant or not.
- This can be done by having **two radio buttons next to each ranked document, one for specifying the document as being relevant and the other for specifying it as nonrelevant.**
- Once Archie has marked the documents of his choice, the important non common terms or keywords present in the relevant documents are used to reweight the keywords in the original query, and expand the query with new terms.

- The effect of reformulating the original query is to "move" the query toward the relevant documents and away from the nonrelevant ones. As a result we expect the reformulated query to retrieve more relevant documents and less nonrelevant ones, thus moving closer toward satisfying the user's information need.
- The relevance feedback process can be repeated for a specified number of times or until the user does not mark any more documents in the results page, that is, until no more feedback is given.
- There are many variations of relevance feedback according to how many documents to include in the process, how many keywords to include in the expanded query, and what weights to attach to these keywords. Ide's "Dec- Hi" method, originally tested in the late 1960s and the early 1970s, within the SMART retrieval system, includes all marked relevant documents but only the highest ranked marked nonrelevant one. The "Dec-Hi" method has proven to be useful over the years in several relevance feedback experiments.
- An interesting variation, called ***pseudorelevance feedback*, assumes that the top-ranked documents are marked as relevant,** and **thus** does not need any user feedback as such. Pseudorelevance feedback is also called ***blind feedback***, since once the query is submitted, no additional input from the user is needed.
- Blind feedback has produced mixed results, due to some of the top-ranked results actually being nonrelevant. It has been shown that when the precision is high for the top-ranked documents, blind feedback provides consistent improvements in the quality of the retrieved documents. It should be noted that blind feedback is not personalized, since the user is not involved in the feedback loop.
- Another way to look at relevance feedback is as a classification process, whereby the retrieval system attempts to improve its ability to discriminate between relevant and nonrelevant documents for an individual user and an initial query. When Archie marks a document as being relevant or nonrelevant, he is acting as a teacher by providing the system with training data, used to tune the classifier.
- Fast forward over 30 years ahead from the initial SMART retrieval experiments to the beginning of 21st century, where web search engines receive hundreds of millions hits a day and must respond to each search in less than a second. Relevance feedback as formulated by Rocchio and his followers, done on a single query basis with explicit user input, is not feasible.
- **Modern users would not be willing to invest the effort to give feedback, and the additional burden on the search engine's servers would be overwhelming. The way forward is to collect implicit user feedback from the users' clickstream data.**
- When Archie clicks on a link from the results page and browses the web page displayed for a given time period, he is providing feedback to the system that the page is relevant to his information need. This information can then be used to personalize Archie's queries through a special purpose tool such as PResTo!
- Spink *et al.* have examined a large query log of the Excite search engine from 1997, one of their aims being to investigate the use of relevance feedback by its users. At the time the Excite search had a "more like this" button next to each query result, so that when the user clicked on it, thus marking the result web page as relevant, the system would initiate a relevance feedback process with this information.

- The researchers found that under 5% of the logged transactions came from relevance feedback; so, only few users used the relevance feedback facility. They also measured the success rate from relevance feedback to be 63%, where a success was counted if the user quit searching after the relevance feedback. They concluded that relevance feedback on the Web merits further investigation.

5) Personalized PageRank:

- A group of researchers from Stanford have developed new algorithms that can significantly speed up the PageRank computation. These improved algorithms are particularly important when the PageRank values are personalized to the interests of an individual user, or biased toward a particular topic such as sports or business.
- The optimization step is of prime importance because each personalized PageRank vector will need to be computed separately, and for web search companies such as Google, scalability of their operation is a crucial ongoing concern.
- Recall that in the original definition of the PageRank, when the random surfer teleports himself, he can end up at any web page with equal probability. In the personalized version, once the surfer is teleported, we bias the probability of jumping to any other web page according to some preference.
- In the extreme case, when teleported, the surfer could always (i.e., with probability one) jump to his home page, or some other favorite page. We refer to this special case of personalized PageRank when the surfer is always teleported to a single page, as the ***individual*** **PageRank** for that page. A more realistic preference may be to jump to a page that the user has bookmarked or to a page from the user's history list, with the probability being proportional to the number of times the user visited the page in the past.
- An interesting fact is that personalization of PageRank had already been suggested in 1998 by the founders of Google, but at that time they could not foresee an efficient and scalable computation of personalized PageRank vectors. There are several problems in realizing personalized PageRank vectors.
- First, data has to be collected for each individual user, secondly, a personalized PageRank vector has to be efficiently computed for each user, and thirdly, the personalized PageRank has to be factored into user queries at the time the queries are submitted.
- An important result, called the *linearity theorem*, simplifies the computation of personalized PageRank vectors. It states that any personalized PageRank vector can be expressed as a linear combination of individual PageRank vectors.
- In particular, one application of this is that the global PageRank vector can be expressed as the average of the linear combination of all possible individual PageRank vectors, one for each page in the Web. This can simplify the computation of personalized PageRank vectors by precomputing individual PageRank vectors and then combining them on demand, depending on the preferred web pages in a personalization instance.
- We can compute PageRank via a Monte Carlo simulation that samples many random walks from each web page. The PageRank of a given page is then computed as the proportion of random walks that end at that page.
- Looking at it from a personalized perspective we can compute individual PageRank vectors by looking only at the samples that start at the single web page being personalized, as suggested by Fogaras *et al.*. The individual PageRank vectors can then

be combined in an arbitrary way, according to the linearity theorem, to obtain the required personalized PageRank vector.

- **An interesting variation of PageRank is *topic sensitive***. This version of PageRank is biased according to some representative set of topics, based on categories chosen, say, from the Open Directory . These could be biased toward the topics that the user prefers to explore, so if, for example, a user prefers sports over world news, this preference would translate to a higher probability of jumping to sports pages than to world news pages. A related approach, biasing the PageRank toward specific queries, is called ***query-dependent* PageRank .**
- Its motivation was to solve the topic drift problem of PageRank, when a site with a high PageRank may be ranked higher than a site which is more relevant to the query. For example, for the query "computer chess", a site having a link from an advertiser with a high PageRank may be ranked higher than a site having links from other "computer chess" sites, despite the latter being more relevant to "computer chess" as judged by its incoming links.
- Query-dependent PageRank is computed on a query basis, by adjusting the random surfer model so that only pages that are relevant to the query are followed. The relevance of a page to the query is then factored into the PageRank calculation. In order to make query-dependent PageRank practical, it has to be computed offline for a selection of query terms. The query terms selected could be topic-based as in topic-sensitive PageRank, they could be popularity-based by looking at query logs, or they could include a selected subset of the words in the search index.
- Another variation of PageRank, called ***BlockRank***, computes local PageRank values on a host basis, and then weights these local PageRank values according to the global importance of the host. BlockRank takes advantage of the fact that a majority of links (over 80% according to the researchers) are within domains rather than between them, and domains such as stanford.edu, typically contain a number of hosts.
- BlockRank could be used to create personalized PageRank vectors at the web host level rather than the web page level, so for example, a user may prefer to jump to a sports site rather than to a general news site. The other attraction of BlockRank is that it can speed up the computation of PageRank by up to 300%.
- Three members of the PageRank group at Stanford were quick to realize the importance to Google of their research on speeding up the PageRank computation and its personalization, which led them to set up a stealth start-up, called Kaltix, in June 2003.
- The next thing that happened was that Google acquired Kaltix in September 2003.This move by Google is a strong indication that personalization is high up on their agenda, and that they view PageRank as a suitable vehicle for personalizing query results. As the competition between the major search engines stiffens, Google has taken the research into its labs for dissection and further development.

6) Outride's Personalized Search:

- In fact, a couple of years earlier, in September 2001, Google acquired the assets of another company specializing in personalization of search, called Outride, which was a spin-off from Xerox Palo Alto Research Center (PARC).

- The acquisition of Outride was strategic, with Google making a claim on the intellectual property that it considered valuable within the personalization of search area. Luckily, in September 2002, the founders of Outride published a research paper in the Communications of the ACM, one of the leading computing magazines, revealing some of the ideas behind the technology they were developing .
- Together with the intellectual property from Kaltix, these acquisitions put Google in a strong position to lead the way to personalization.
- Link analysis based on the evaluation of the authority of web sites is biased against relevance, as determined by individual users. For example, when you submit the query "java" to Google, you get many pages on the programming language Java, rather than the place in Indonesia or the well-known coffee from Java.
- Popularity or usage-based ranking adds to the link-based approach, by capturing the flavor of the day and how relevance is changing over time for the user base of the search engine. In both these approaches, relevance is measured for the population of users and not for the individual user. Outride set out to build a model of the user, based on the context of the activity of the user, and individual user characteristics such as prior knowledge and history of search.
- The Outride system set out to integrate these features into the user interface as follows. Once Archie submits his query, its context is determined and the query is augmented with related terms.
- After it is processed, it is individualized, based on demographic information and the past user history. A feature called "Have Seen, Have Not Seen" allows the user to distinguish between old and new information. The Outride user interface is integrated into the browser as a side bar that can be opened and closed much like the favorites and history lists.
- According to the authors of the research paper, searches were faster and easier to complete using Outride. It remains to see what Google will do with this thought-provoking technology. Jeff Heer, who is acquainted with Outride's former employees, said in his weblog that "Their technology was quite impressive, building off a number of PARC innovations, but they were in the right place at the wrong time".
- It is worth mentioning that Google has released a tool enabling users to search the Web from any application within the Windows operating system. In releasing this desktop search tool, Google has taken a step toward integrating search within a broader context than the user's browser, and paving the way toward a personalized search tool. Another significant feature of the tool is that its query results can be displayed in a separate window rather than in the user's browser; it also allows the user to customize the tool to search within specific web sites.
- In December 2009, Google launched a personalized search service that delivers customized search results to each user based on his or her history of web searches and resulting page views. If the user is signed into Google and has enabled a feature called *web history*, then this information is used for personalization. Otherwise, when the user is not signed in or web history is disabled, personalization is based on the user's past history stored in a cookie.
- Up to 180 days of search activity is stored in the cookie including the searches and clicked results. Personalization is based on reranking the original Google results

according to previous searches and clicks, possibly constructing a personalized PageRank for each user and building on OutRide's personalization algorithms.

4.9. COLLABORATIVE FILTERING:

- Static and adaptive filtering are not social tasks, in that profiles are assumed to be independent of each other. If we now consider the complex relationships that exist between profiles, additional useful information can be obtained. For example, suppose that we have an adaptive filtering system with two profiles, which we call profile A (corresponding to user A) and profile B (corresponding to user B).
- If both user A and B judged a large number of the same documents to be relevant and/or non-relevant to their respective profiles, then we can infer that the two profiles are similar to each other. We can then use this information to improve the relevance of the matches to both user A and B. For example, if user A judged a document to be relevant to profile A, then it is likely that the document will also be relevant to profile B, and so it should probably be retrieved, even if the score assigned to it by the adaptive filtering system is below the predetermined threshold.
- Such a system is social, in the sense that a document is returned to the user based on both the document's topical relevance to the profile and any judgments or feedback that users with similar profiles have given about the document.
- Filtering that considers the relationship between profiles (or between users) and uses this information to improve how incoming items are matched to profiles (or users) is called *collaborative filtering*.
- Collaborative filtering is often used as a component of *recommender systems*. Recommender systems use collaborative filtering algorithms to recommend items (such as books or movies) to users. Many major commercial websites, such as Amazon.com and Netflix, make heavy use of recommender systems to provide users with a list of recommended products in the hopes that the user will see something she may like but may not have known about, and consequently make a purchase. Therefore, such systems can be valuable both to the end users, who are likely to see relevant products, including some that they may not have considered before, and to search engine companies, who can use such systems to increase revenue.
- Collaborative filtering algorithms for recommender systems differ from static and adaptive filtering algorithms in a number of ways.

  1) When collaborative filtering algorithms are used for making recommendations, they typically associate a single profile with each user. That is, the user is the profile.
  2) Static and adaptive filtering systems would make a binary decision (retrieve or do not retrieve) for each incoming document, but collaborative filtering algorithms for recommender systems provide *ratings* for items. These ratings may be 0 (relevant) and 1 (non-relevant) or more complex, such as ratings on a scale of 1 through 5.
  3) Collaborative filtering algorithms for recommender systems provide a rating for every incoming item, as well as every item in the database for which the current user has not explicitly provided a judgment. On the other hand, static and adaptive filtering algorithms

only decide whether or not to send incoming documents to users and never retrospectively examine older documents to determine whether they should be retrieved.

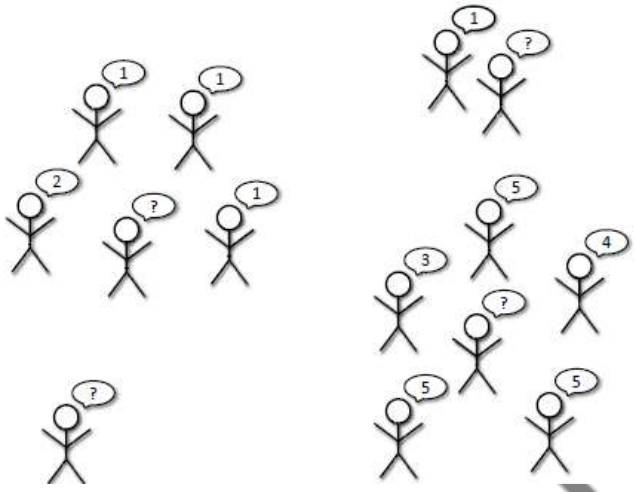


Fig. A set of users within a recommender system. Users and their ratings for some item are given. Users with question marks above their heads have not yet rated the item. It is the goal of the recommender system to fill in these question marks.

- Figure represents a virtual space of users, where users with similar preferences and tastes are close to each other. The dialog boxes drawn above each user's head denote their preference for some item, such as a movie about tropical fish. Those users who have not rated the movie have question marks in their dialog boxes. It is the job of the collaborative filtering algorithm to predict as accurately as possible what rating these users would give to the movie.
- Collaborative filtering is conceptually simple, but the details can be difficult to get correct. For example, one must decide how to represent users and how to measure the similarity between them. After similar users have been identified, the user ratings must be combined in some way. Another important issue concerns how collaborative filtering and, in particular, recommender systems should be evaluated.

Rating with user clusters

- In both of the algorithms that follow, we assume that we have a set of users *U* and a set of items *I*. Furthermore, $r_u(i)$ is user *u*'s rating of item *i*, and $\hat{r}_u(i)$ is our system's prediction for user *u*'s rating for item *i*. Note that $r_u(i)$ is typically undefined when user *u* has not provided a rating for item *i*, although, as we will describe later, this does not need to be the case. Therefore, the general collaborative filtering task is to compute $\hat{r}_u(i)$ for every user/item pair that does not have an explicit rating.
- We assume that the only input we are given is the explicit ratings $r_u(i)$, which will be used for making predictions. Furthermore, for simplicity, we will assume that ratings are integers in the range of 1 through *M*, although most of the algorithms described will work equally well for continuous ratings. One simple approach is to first apply one of the

clustering algorithms to the set of users. Typically, users are represented by their rating vectors $r_u = [r_u(i_1) \ldots r_u(i_{|\mathcal{U}|})]$.

- However, since not all users judge all items, not every entry of the vector *ru* may be defined, which makes it challenging to compute distance measures, such as cosine similarity. Therefore, the distance measures must be modified to account for the missing values. The simplest way to do this is to fill in all of the missing ratings with some value, such as 0. Another possibility is to fill in the missing values with the user's average rating, denoted by r_u, or the item's average rating.
- One of the common similarity mesures used for clustering users is the *correlation* measure, which is computed as follows:

$$\frac{\sum_{i \in I_u \cap I_{u'}} (r_u(i) - \overline{r}_u) \cdot (r_{u'}(i) - \overline{r}_{u'})}{\sqrt{\sum_{i \in I_u \cap I_{u'}} (r_u(i) - \overline{r}_u)^2 \sum_{i \in I_u \cap I_{u'}} (r_{u'}(i) - \overline{r}_{u'})^2}}$$

- Where *Iu* and *Iu′* are the sets of items that users *u* and *u′* judged, respectively, which means that the summations are only over the set of items that both user *u* and *u′* judged. Correlation takes on values in the range –1 to 1, with 1 being achieved when two users have identical ratings for the same set of items, and –1 being achieved when two users rate items exactly the opposite of each other.

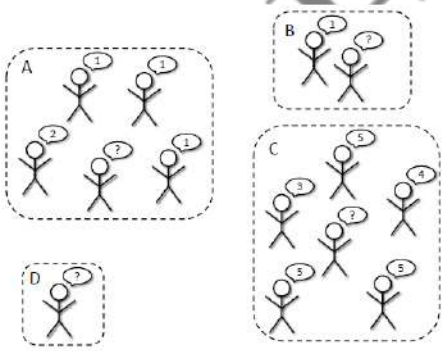


Fig. Illustration of collaborative filtering using clustering. Groups of similar users are outlined with dashed lines. Users and their ratings for some item are given. In each group, there is a single user who has not judged the item. For these users, the unjudged item is assigned an automatic rating based on the ratings of similar users.

- In Figure, we provide a hypothetical clustering of the users, denoted by dashed boundaries. After users have been clustered, any user within a cluster that has not judged some item could be assigned the average rating for the item among other users in the cluster. For example, the user who has not judged the tropical fish movie in cluster A would be assigned a rating of 1.25, which is the average of the ratings given to the movie by the four other users in cluster A. This can be stated mathematically as:

$$\hat{r}_u(i) = \frac{1}{|Cluster(u)|} \sum_{u' \in Cluster(u)} r_{u'}(i)$$

- Where *Cluster*(*u*) represents the cluster that user *u* belongs to.
- Averaging the ratings of a group of users is one simple way of aggregating the ratings within a cluster. Another possible approach is to use the expected rating of the item, given the ratings of the other users within the cluster, which is calculated as:

$$\begin{aligned}\hat{r}_u(i) &= \sum_{x=1}^{M} x \cdot P(r_u(i) = x | C = Cluster(u)) \\ &= \sum_{x=1}^{M} x \cdot \frac{|u' : r_{u'}(i) = x|}{|Cluster(u)|}\end{aligned}$$

- Where $P(r_u(i) = x | C = Cluster(u))$ is the probability that user *u* will rate the item with rating *m*, given that they are in cluster *Cluster*(*u*). This probability is estimated as $\frac{|u' : r_{u'}(i) = x|}{|Cluster(u)|}$, which is the proportion of users in *Cluster*(*u*) who have rated item *i* with value *x*. For example, if all of the users in *Cluster*(*u*) rate the item 5, then $\hat{r}_u(i)$ will equal 5. However, if five users in *Cluster*(*u*) rate the item 1 and five users rate it 5, then $\hat{r}_u(i) = 1 \cdot \frac{5}{10} + 5 \cdot \frac{5}{10} = 3$.

- One issue that arises when relying on clustering for predicting ratings is very sparse clusters, such as cluster D in Figure. What score should be assigned to a user who does not fit nicely into a cluster and has rather unique interests and tastes? This is a complex and challenging problem with no straightforward answer.
- One simple, but not very effective, solution is to assign average item ratings to every unrated item for the user. Unfortunately, this explicitly assumes that "unique" users are average, which is actually unlikely to be true.

Rating with nearest neighbors

- An alternative strategy to using clusters is to use nearest neighbors for predicting user ratings. This approach makes use of the*K* nearest neighbors clustering technique described in Chapter 9. To predict ratings for user *u*, we first find the *K* users who are closest to the user according to some similarity measure. Once we have found the nearest neighbors, we will use the ratings (and similarities) of the neighbors for prediction as follows:

$$\hat{r}_u(i) = \overline{r}_u + \frac{1}{\sum_{u' \in \mathcal{N}(u)} sim(u, u')} \sum_{u' \in \mathcal{N}(u)} sim(u, u')(r_{u'}(i) - \overline{r}_{u'})$$

- Where *sim*(*u*, *u′*) is the similarity of user *u* and *u′*, and*N*(*u*) is the set of *u*'s nearest neighbors. This algorithm predicts user *u*'s rating of item *i* by first including the user's average item rating (r_u). Then, for every user *u′* in *u*'s nearest neighborhood, $r_{u'}(i) - \overline{r}_{u'}$ is weighted by *sim*(*u*, *u′*) and added into the predicted value.
- You may wonder why $r_{u'}(i) - \overline{r}_{u'}$ is used instead of *ru′*(*i*). The difference is used because ratings are relative. Certain users may very rarely rate any item with 1, whereas other users may never rate an item below 3. Therefore, it is best to measure ratings relative to a given user's average rating for the purpose of prediction.
- Although the approaches using clusters and nearest neighbors are similar in nature, the nearest-neighbors approach tends to be more robust with respect to noise. Furthermore, when using nearest neighbors, there is no need to choose a clustering cost function, only a similarity function, thereby simplifying things.
- Empirical results suggest that predicting ratings using nearest neighbors and the correlation similarity measure tends to outperform the various clustering approaches across a range of data sets. Based on this evidence, the nearest neighbor approach, using the correlation similarity function, is a good choice for a wide range of practical collaborative filtering tasks.

Evaluation

- Collaborative filtering recommender systems can be evaluated in a number of ways. Standard information retrieval metrics can be used, including accuracy, precision, recall, and the F measure.
- However, standard information retrieval measures are very strict, since they require the system to predict exactly the correct value. Consider the case when the actual user rating is 4 and system A predicts 3 and system B predicts 1. The accuracy of both system A and B is zero, since they both failed to get exactly the right answer. However, system A is much closer to the correct answer than system B. For this reason, a number of evaluation metrics that consider the difference between the actual and predicted ratings have been used. One such measure is *absolute error*, which is computed as:

$$ABS = \frac{1}{|\mathcal{U}||\mathcal{I}|} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} |\hat{r}_u(i) - r_u(i)|$$

where the sums are over the set of user/item pairs for which predictions have been made. The other measure is called *mean squared error*, which can be calculated as:

$$MSE = \frac{1}{|\mathcal{U}||\mathcal{I}|} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (\hat{r}_u(i) - r_u(i))^2$$

- The biggest difference between absolute error and mean squared error is that mean squared error penalizes incorrect predictions more heavily, since the penalty is squared.
- These are the most commonly used evaluation metrics for collaborative filtering recommender systems. So, in the end, which measure should you use? Unfortunately, that

is not something we can easily answer. As we have repeated many times throughout the course of this book, the proper evaluation measure depends a great deal on the underlying task.

4.10.CONTENT BASED RETRIEVAL:
IMAGE SEARCH

- Web image search is important, since there is a substantial amount of visual information in web pages that users may wish to find. In some cases an image may act as a discriminator for a text-based search, for example if you are looking for a company and all you can remember is that their logo has an image of a chess piece. In other cases the result of the search could be an image, for example, if you wish to view an image of Senate House in London. Some of the images were obtained from the public web through a judicious search process, but none of the searches involved *content-based retrieval* , using features contained in the images.
- Several applications of image search are filtering offensive and illegal material (such material is not always easy to detect by the text in web pages), travel and tourism (images and maps of places we are planning to visit), education and training (images to illustrate ideas), entertainment (for fun), e-commerce (we wish to see what we plan to buy), design (such as building plans), history and art (we wish to view artifacts or paintings), fashion (we wish to see what is trendy), and domain-specific image retrieval (such as trademarks, fingerprints, and stamps).
- Compared to text-based web search, content-based image search is very much still in the research labs. In addition to image search, there are the issues pertaining to searching general multimedia content, whatever it may be, and the specific problems that need to be addressed for each specific type of media.
- Searching and retrieving 3D models is a challenging problem. It involves finding shape representations of objects that allow similarity between objects to be detected. Querying 3D models can involve text, but also a sketch of the model to be matched, for example a skeleton of a car can be drawn and the system will return car-like objects from the repository. A demonstration of a 3D search engine, under development by the Princeton Shape Retrieval and Analysis Group, can be found at http://shape.cs.princeton.edu/search.html.
- Searching audio content is an area that is developing in parallel to speech recognition. However, audio information retrieval is much wider in its scope than speech recognition. An important application of audio retrieval is being able to recognize, compare, and classify music objects on the Web. One concrete application of audio search, commercialized by Shazam Entertainment (www.shazam.com), is the recognition of songs via mobile phones .
- The user dials in to the service, and then points the phone at the source of the music for a period of about 10–15 secs. Within several additional seconds the service will identify the track from a database containing over 8 million fingerprints of tracks (as of 2009), using music recognition software developed by Avery Wang, while he was a PhD student in Electrical Engineering at Stanford University.

- Searching video content is a natural extension of web image and audio retrieval, as more video is becoming available on the Web. Video is richer in content that other multimedia as it includes images and audio. One challenge specific to video retrieval is being able to detect an image over a sequence of frames, which satisfies a *motion query* such as "a person moving in a specific direction".
- The current approach to audio and video search taken by the major search engines is mainly text based. Google (http://video.google.com) indexes the closed-captions hidden in the video signal of TV broadcasts to provide a search tool for TV programs.
- In addition, Google searches for YouTube videos and other videos found by its crawlers. Yahoo (http://video.search.yahoo.com) and Microsoft Big (http://video.bing.com) provide comparable search facilities, which include video content available on the Web, while Singingfish (http://video.aol.com), which has been acquired by AOL, is a dedicated multimedia search engine using a text-based approach for both video and audio.
- We now concentrate on image search. Although text-based web search engines are designed to find relevant web pages, they do not have in-built capabilities to find images within these pages. To remedy this situation the major web search engines now provide image search as a separate service.
- In the context of the Web, image search, is the problem of finding relevant images contained in web pages. To answer a query, an image search engine can use the textual cues within the page containing the image, and the content embedded in the image itself. The grand challenge of image search is to be able to reliably retrieve images by content . Below we will discuss how this can be done, but first we will see how text-based image search is carried out by the major search engines.

1) Text-Based Image Search:

- Google (http://images.google.com), Yahoo (http://images.yahoo.com), and Microsoft Bing (http://images.bing.com) provide image search as part of their service, while Picsearch (www.picsearch.com) is an independent image search engine, managed from Stockholm, which is devoted to video, image, and audio retrieval.
- As of early 2010, Picsearch was providing access to more than 3 billion pictures, and the main web search engines with their powerful crawling mechanisms will be indexing a much larger number.
- In all these image search engines the query is specified textually; no visual input, such as the user providing an example image, is possible. At this time (early 2010), apart from using available textual information, image search engines are already using some low-level features from the image's content in the search process.
- How can images be ranked using textual information? When the search engine's crawler downloads HTML web pages, it also downloads the images it contains, which are specified within the IMG tag. Together with the image, the crawler extracts the image filename, the text describing the image from the ALT field, the title of the image, the URL of the web page containing the image, and any text surrounding the IMG tag . The keywords obtained from these snippets of text are then stored in the image index, and used to search for the image utilizing the standard TF–IDF text retrieval method.

- Link-based metrics are independent of the image content, can be used to enhance the image retrieval algorithm, in a way similar to the PageRank and HITS (hub and authorities) algorithms used for web page search. In this context, a page is said to link to the image if either
  (i) The image is contained in the page
  (ii) The image is the source of a link to the page
  (iii) The page has a link pointing to the image.
- Using a variation of HITS, it is possible to detect *image containers*, defined as pages containing high-quality images, and *image hubs*, defined as pages pointing to good image containers.
- The results from such link analysis can be included in the image retrieval algorithm, by factoring in the score of the page the image is contained in, and also the scores of pages linking to the page it is contained in. In particular, if an image is part of an image container or pointed to by an image hub, its score will be higher. This algorithm has been implemented in an experimental image search engine called PicASHOW .
- It is interesting to note that, for many queries, there is very little overlap between the first few results returned by the three search engines, which probably indicates that there are substantial differences in the images they each store, and that the algorithms they use to rank the images may be quite different.

2) Content-Based Image Search

- Apart from the standard textual query, a user may specify an image as input to the search engine. Often it is hard to find an image, which is similar to the one we are searching, so the initial query may be textual.
- An interesting visual alternative, apart from presenting an image, that is provided by some image search engines is to enable users to sketch the main feature of the image they are looking for, or to provide a representative icon. After the initial results are returned the user may wish to refine their query through relevance feedback, as described below.
- When an image is added to the search engine's index, the image is segmented into smaller regions that are homogeneous according to some criterion. Low-level features, notably, color, shape, and texture, are extracted from the image and stored in the index. As the number of dimensions of the feature vectors may be high, dimension reduction techniques, such as clustering, will often be employed before the features are stored. In practice the information stored about the image is only a partial description of the objects represented in it.
- Moreover, low-level features cannot describe high-level ones such as the object being a car, a person, or a holiday beach. Worse still is the fact that humans tend to interpret images in terms of high-level semantic features. This, most challenging, problem is called the *semantic gap*.
- One way to address the semantic gap is to add textual annotations to images, but in the context of web search this is not a scalable solution. We have already discussed how current search engines use textual cues in web image search; these textual cues provide a form of high-level annotation.
- Organizing images by categories in a directory structure, in a similar manner to a web directory, is another method to help users identify images through semantic concepts; but as with web page directories, this is also not a scalable proposition. However, a well-built

image directory could provide many example images a user could use to initiate a content-based image search.

- To illustrate the retrieval process, assume that the query is specified as an image. This input image is segmented and its features are extracted. Then an index lookup is carried out to find the images most similar to the input, in particular, its *k* nearest neighbors are found and ranked by similarity. These are presented to the user, who can then refine the query through relevance feedback.
- In an attempt to find a suitable similarity measure, researchers have borrowed ideas from the psychological theory of similarity assessment. One common assumption is that similarity is a positive and a monotonically non-decreasing function satisfying certain distance axioms, but a different, probabilistic approach to similarity, is also possible.
- The idea behind relevance feedback is that the user, say Archie, can refine his initial query as follows, after the image retrieval system returns to him a ranked list of result images, as answers to his initial query. Archie then inspects the returned images and marks some of them as "relevant" (positive examples of what he wants to see, i.e., more like this) and others as "not relevant" (negative examples of what he does not want to see, i.e., less like this).
- The system responds to this feedback by adjusting the query in the direction of his feedback. The adjustment involves modifying the weights of features in the original query and expanding the query with new features from the marked images.
- The weights of features from positive examples are increased and the weights of features from negative examples are decreased. The adjusted query is then reissued to the image search engine, and the expected outcome is that the new set of result images have "moved" in the direction of what Archie wants to see, that is, more of the images are relevant and less are irrelevant.
- The process of relevance feedback can be iterated several times until Archie is satisfied with the results or until the set of results has stabilized. Relevance feedback is especially important in the context of image search, as often the best way to formulate a query is by giving the system example images of what you want to see, in addition or without a textual query.
- Relevance feedback can also be collaborative, in the sense that it can be stored in a log file and then aggregated across many users. The idea here is to adjust the feature weights of all users with similar queries, thus sharing feedback between different users. Experiments with iFind, a content-based image search engine developed at Microsoft Research China, have shown that this form of collective relevance feedback is effective in improving the precision of the image search engine.
- A variation, called *pseudo relevance feedback*, can be used to improve query results based only on textual cues and without any user interaction. This is how it works. Archie submits the query to the image search engine as before, and an initial set of result images is returned. Now, instead of returning these to Archie, the system reranks the initial results using a text-based search engine, as follows.
- First we consider the web pages that contain the initial results and build a vector for each page that stores the TF–IDF values of each word in the page, after omitting stop words and possibly stemming the remaining words; we call these vectors, the *image vectors* for

the query. The original query is then submitted to a text-based search engine, which returns an alternative set of result pages.

- At this stage, we construct a single *text vector*, storing the TF–IDF values of all the words in the results, by considering all the text-based results together as if they were present in a single page. (We note that in this scheme it is possible to give higher weight to words appearing in higher ranked pages.)
- The reranking of the initial results is now carried out by computing the similarity between each image vector and the text vector, and ranking them from highest to lowest similarity. One measure of similarity that can be computed between the two vectors is the dot product of the vectors. This is called *vector similarity* and is computed by multiplying the TF–IDF values in the two vectors word by word and summing up the results; if desired these could be normalized to output a similarity measure between 0 and 1.
- Experiments using pseudorelevance feedback to rerank results from Google's image search have shown substantial increases in precision of the image search system.

3) VisualRank

- Jing and Baluja from Google have proposed to build a content-based similarity graph of images and to compute a VisualRank for each image inspired by the PageRank computation for web pages.
- The way this is done is as follows. First, local features for each image as computed using Lowe's scale-invariant feature transform (SIFT) algorithm .
- The idea is to compute local descriptors for the image that are relatively stable under different transformations to the image such as scaling, rotation, or noise. The output of SIFT is a set of key point descriptors for the image describing all the features of a key point, where a key point is a local point in the image that is identified as distinct. The similarity between two images is defined as the number of key points they have in common, divided by the average number of key points in the two images.
- VisualRank is then computed on the image similarity graph with some important differences to the PageRank computation. The edges in the graph, called *visual links*, are weighted by the similarity between the images on the two sides of the edge, that is, the graph is weighted. Moreover, unlike web links, visual links are symmetric, that is, the image similarity graph is undirected.
- As in PageRank we have a teleportation factor, and the VisualRank vector can also be personalized, for example, by biasing it to the top-*m* image results from an image search engine. The VisualRank of the graph can then be computed by the power method or some optimization thereof.
- Computing VisualRank for all the Web is computationally prohibitive as it would involve generating a similarity graph for billions of images. A practical approach, which was used to compare VisualRank to the ranking generated by Google's image search, is to make the VisualRank query dependent. This is done by fixing the query and extracting the top-*n* images, say 1000, from an image search engine for this query, and reranking the results according to VisualRank.
- Jing and Baluja did exactly this for 2000 popular product queries such as "ipod", "Xbox", and "Picasso". Figure shows the similarity graph generated from the top 1000 search

results for "Mona-Lisa"; the largest two images in the center have the highest VisualRank.

- A user study of the performance of VisualRank as compared to Google Images, showed that VisualRank can significantly reduce the number of irrelevant images in search results and increase the number of clicks from the top-20 ranked images.
- The method of VisualRank could also be extended to the domains of audio and video search.

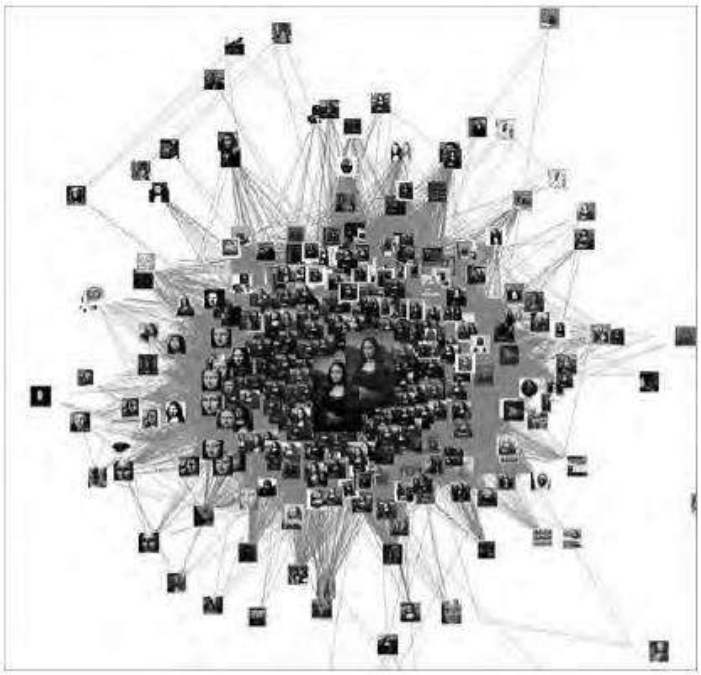


Figure .Similarity graph generated from the top 1000 search results of "Mona-Lisa".

4) CAPTCHA and reCAPTCHA

- CAPTCHA stands for completely automated public Turing test to tell computers and humans apart. It usually involves a computer asking a user to perform a task that will differentiate a human from an automated program.
- A common type of CAPTCHA is a distorted text image that needs to be typed in, but it could also be image-based requiring the user to either to identify an object in an image, to answer questions about the image, or to find the odd one out of several images.
- Ahn *et al.* estimate that more than 100 million CAPTCHAs are answered by humans every day and suggest making use of this effort to help machines read scanned text and thus complement optical character recognition (OCR), which cannot recognize about 20% of words in older faded prints.
- The suggested method, called reCAPTCHA, presents a scanned text from a newspaper or book, which OCR software cannot read, and asks the user to decipher it. When multiple users decipher the text in the same way, the text is accepted, and when multiple users reject the text, it is deemed unreadable.
- As of late 2008, the system could transcribe 4 million words per day. This is a very good example of how "human computation" can be harnessed to help solve problems that are difficult for computers to solve. The company reCAPTHA (http://recaptcha.net), which

grew out a project of the School of Computer Science at Carnegie Mellon University, was acquired by Google in September 2009.

5) Image Search for Finding Location-Based Information

- A novel application of image search in mobile computing, developed in the MIT Computer Science and Artificial Intelligence Laboratory, is the use of photographic images to specify a location. The scenario is that of a user taking a picture of a landmark with his or her mobile phone, and sending it as a query over the mobile web to an image search engine, which will find relevant location-based information and return it to the user.
- A more detailed description of the process, after an image has been sent, is as follows:
  (i) The most similar images to the snapshot are retrieved from a content-based image search engine, restricted to a chosen geographic region
  (ii) Relevant keywords are extracted from the web pages containing these images and are submitted to a web image search engine as a query, which will return the highest ranking images for the query
  (iii) Content-based image retrieval is used to choose the most similar images to the snapshot, and finally the web pages these images are contained in are returned to the user.
- In step (i) the content-based image search engine has a relatively small index, covering only few images of each landmark, within the region the user is expected to be in; this is done for efficiency purposes. To get accuracy we need step (ii), leveraging the power of web image search engines, so that the quality of the final results returned to the user is ensured. Step (iii) is needed in order to get a small number of focused results back to the user.

4.11. INVISIBLE WEB

- ✓ Search engines are, in a sense, the heartbeat of the internet; "googling" has become a part of everyday speech and is even recognized by Merriam-Webster as a grammatically correct verb. It's a common misconception, however, that googling a search term will reveal every site out there that addresses your search. In fact, typical search engines like Google, Yahoo, or Bing actually access only a tiny fraction – estimated at 0.03% – of the internet. The sites that traditional searches yield are part of what's known as the Surface Web, which is comprised of indexed pages that a search engine's web crawlers are programmed to retrieve.
- ✓ So where's the rest? The vast majority of the Internet lies in the Deep Web, sometimes referred to as the Invisible Web. The actual size of the Deep Web is impossible to measure, but many experts estimate it is about 500 times the size of the web as we know it.
- ✓ Deep Web pages operate just like any other site online, but they are constructed so that their existence is invisible to Web crawlers.

Search Engines and the Surface Web

- ✓ Understanding how surface Web pages are indexed by search engines can help you understand what the Deep Web is all about. In the early days, computing power and storage space was at such a premium that search engines indexed a minimal number of pages, often storing only partial content. The methodology behind searching reflected users' intentions; early Internet users generally sought research, so the first search engines indexed simple queries that students or other researchers were likely to make. Search results consisted of actual content that a search engine had stored.
- ✓ Over time, advancing technology made it profitable for search engines to do a more thorough job of indexing site content. Today's Web crawlers, or spiders, use sophisticated algorithms to collect page data from hyperlinked pages. These robots maneuver their way through all linked data on the Internet, earning their spidery nickname. Every surface site is indexed by metadata that crawlers collect. This metadata, consisting of elements such as page title, page location (URL) and repeated keywords used in text, takes up much less space than actual page content. Instead of the cached content dump of old, today's search engines speedily and efficiently direct users to websites that are relevant to their queries.
- ✓ To get a sense of how search engines have improved over time, Google's interactive breakdown "How Search Works" details all the factors at play in every Google search. In a similar vein, Moz.com's timeline of Google's search engine algorithm will give you an idea of how nonstop the efforts have been to refine searches. How these efforts impact the Deep Web is not exactly clear. But it's reasonable to assume that if major search engines keep improving, ordinary web users will be less likely to seek out arcane Deep Web searches.

How is the Deep Web Invisible to Search Engines?

- ✓ Search engines like Google are extremely powerful and effective at distilling up-to-the-moment Web content. What they lack, however, is the ability to index the vast amount of data that isn't hyperlinked and therefore immediately accessible to a Web crawler. This may or may not be intentional; for example, content behind a paywall or a blog post that's written but not yet published both technically reside in the Deep Web.
- ✓ Some examples of other Deep Web content include:

- Data that needs to be accessed by a search interface
- Results of database queries
- Subscription-only information and other password-protected data
- Pages that are not linked to by any other page
- Technically limited content, such as that requiring CAPTCHA technology
- Text content that exists outside of conventional *http://* or *https:// protocols*
- While the scale and diversity of the Deep Web are staggering, it's notoriety – and appeal – comes from the fact that users are anonymous on the Deep Web, and so are their Deep Web activities. Because of this, it's been an important tool for governments; the U.S. Naval research laboratory first launched intelligence tools for Deep Web use in 2003.

- Unfortunately, this anonymity has created a breeding ground for criminal elements who take advantage of the opportunity to hide illegal activities. Illegal pornography, drugs, weapons and passports are just a few of the items available for purchase on the Deep Web. However, the existence of sites like these doesn't mean that the Deep Web is inherently evil; anonymity has its value, and many users prefer to operate within an untraceable system on principle.
- Just as Deep Web content can't be traced by Web crawlers, it can't be accessed by conventional means. The same Naval research group to develop intelligence-gathering tools created The Onion Router Project, now known by its acronym TOR. Onion routing refers to the process of removing encryption layers from Internet communications, similar to peeling back the layers of an onion. TOR users' identities and network activities are concealed by this software. TOR, and other software like it, offers an anonymous connection to the Deep Web. It is, in effect, your Deep Web search engine.
- But in spite of its back-alley reputation there are plenty of legitimate reasons to use TOR. For one, TOR lets users avoid "traffic analysis" or the monitoring tools used by commercial sites, for one, to determine web users' location and the network they are connecting through. These businesses can then use this information to adjust pricing, or even what products and services they make available.
- According to the Tor Project site, the program also allows people to, "[…] Set up a website where people publish material without worrying about censorship." While this is by no means a clear good or bad thing, the tension between censorship and free speech is felt the world over; the Deep Web. The Deep Web furthers that debate by demonstrating what people can and will do to overcome political and social censorship.

Reasons a Page is Invisible

- ✓ When an ordinary search engine query comes back with no results, that doesn't necessarily mean there is nothing to be found. An "invisible" page isn't necessarily inaccessible; it's simply not indexed by a search engine. There are several reasons why a page may be invisible. Keep in mind that some pages are only temporarily invisible, possibly slated to be indexed at a later date.

- Engines have traditionally ignored any Web pages whose URLs have a long string of parameters and equal signs and question marks, on the off chance that they'll duplicate what's in their database – or worse – the spider will somehow go around in circles. Known as the **"Shallow Web,"** a number of workarounds have been developed to help you access this content.
- **Form-controlled entry** that's not password-protected. In this case, page content only gets displayed when a human applies a set of actions, mostly entering data into a form (specific query information, such as job criteria for a job search engine). This typically includes databases that generate pages on demand. Applicable content includes travel industry data (flight info, hotel availability), job listings, product databases, patents, publicly-accessible government information, dictionary definitions, laws, stock market data, phone books and professional directories.
- **Passworded access**, subscription or non subscription. This includes VPN (virtual private networks) and any website where pages require a username and password. Access may or

may not be by paid subscription. Applicable content includes academic and corporate databases, newspaper or journal content, and academic library subscriptions.

- **Timed access**. On some sites, like major news sources such as the *New York Times*, free content becomes inaccessible after a certain number of page views. Search engines retain the URL, but the page generates a sign-up form, and the content is moved to a new URL that requires a password.
- **Robots exclusion**. The robots.txt file, which usually lives in the main directory of a site, tells search robots which files and directories should not be indexed. Hence its name "robots exclusion file." If this file is set up, it will block certain pages from being indexed, which will then be invisible to searchers. Blog platforms commonly offer this feature.
- **Hidden pages**. There is simply no sequence of hyperlink clicks that could take you to such a page. The pages are accessible, but only to people who know of their existence.

Ways to Make Content More Visible

We have discussed what type of content is invisible and where we might find such information. Alternatively, the idea of making content more visible spawned the Search Engine Optimization (SEO) industry. Some ways to improve your search optimization include:

- **Categorize your database**. If you have a database of products, you could publish select information to static category and overview pages, thereby making content available without form-based or query-generated access. This works best for information that does not become outdated, like job postings.
- **Build links within your website, interlinking between your own pages**. Each hyperlink will be indexed by spiders, making your site more visible.
- **Publish a sitemap**. It is crucial to publish a serially linked, current sitemap to your site. It's no longer considered a best practice to publicize it to your viewers, but publish it and keep it up to date so that spiders can make the best assessment of your site's content.
- **Write about it elsewhere**. One of the easiest forms of Search Enging Optimization (SEO) is to find ways to publish links to your site on other webpages. This will help make it more visible.
- Use **social media to promote your site**. Link to your site on Twitter, Instagram, Facebook or any other social media platform that suits you. You'll drive traffic to your site and increase the number of links on the Internet.
- **Remove access restrictions**. Avoid login or time-limit requirements unless you are soliciting subscriptions.
- **Write clean code**. Even if you use a pre-packaged website template without customizing the code, validate your site's code so that spiders can navigate it easily.
- Match your site's page titles and link names to other text within the site, and **pay attention to keywords that are relevant to your content**.

How to Access and Search for Invisible Content

If a site is inaccessible by conventional means, there are still ways to access the content, if not the actual pages. Aside from software like TOR, there are a number of entities who do make it

possible to view Deep Web content, like universities and research facilities. For invisible content that cannot or should not be visible, there are still a number of ways to get access:

- Join a professional or research association that provides access to records, research and peer-reviewed journals.
- Access a virtual private network via an employer.
- Request access; this could be as simple as a free registration.
- Pay for a subscription.
- Use a suitable resource. Use an invisible Web directory, portal or specialized search engine such as Google Book Search, Librarian's Internet Index, or BrightPlanet's Complete Planet.

Invisible Web Search Tools

Here is a small sampling of invisible web search tools (directories, portals, engines) to help you find invisible content. To see more like these, please look at our Research Beyond Google article.

A List of Deep Web Search Engines – Purdue Owl's Resources to Search the Invisible Web

- Art – Musie du Louvre
- Books Online – The Online Books Page
- Economic and Job Data – FreeLunch.com
- Finance and Investing – Bankrate.com
- General Research – GPO's Catalog of US Government Publications
- Government Data – Copyright Records (LOCIS)
- International – International Data Base (IDB)
- Law and Politics – THOMAS (Library of Congress)
- Library of Congress – Library of Congress
- Medical and Health – PubMed
- Transportation – FAA Flight Delay Information

4.12. SUMMARIZATION AND SNIPPETS GENERATION:

Result Pages and Snippets

- Successful interactions with a search engine depend on the user understanding the results. Many different visualization techniques have been proposed for displaying search output (Hearst, 1999), but for most search engines the result pages consist of a ranked list of *document summaries* that are linked to the actual documents or web pages.
- A document summary for a web search typically contains the title and URL of the web page, links to live and cached versions of the page, and, most importantly, a short text summary, or *snippet*, that is used to convey the content of the page.

- In addition, most result pages contain advertisements consisting of short descriptions and links. Query words that occur in the title, URL, snippet, or advertisements are *highlighted* to make them easier to identify, usually by displaying them in a bold font.
- Figure gives an example of a document summary from a result page for a web search. In this case, the snippet consists of two partial sentences. Snippets are sometimes full sentences, but often text fragments, extracted from the web page. Some of the snippets do not even contain the query words.

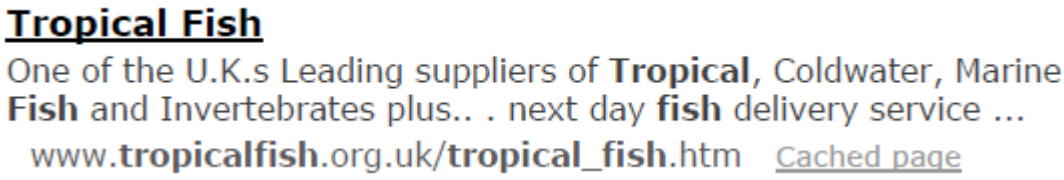


Fig. Typical document summary for a web search

- Snippet generation is an example of text summarization. Summarization techniques have been developed for a number of applications, but primarily have been tested using news stories from the TREC collections.
- A basic distinction is made between techniques that produce *query-independent* summaries and those that produce *query-dependent* summaries. Snippets in web search engine result pages are clearly query-dependent summaries, since the snippet that is generated for a page will depend on the query that retrieved it, but some query-independent features, such as the position of the text in the page and whether the text is in a heading, are also used.
- The development of text summarization techniques started with H. P. Luhn in the 1950s (Luhn, 1958). Luhn's approach was to rank each sentence in a document using a *significance factor* and to select the top sentences for the summary.
- The significance factor for a sentence is calculated based on the occurrence of significant words. Significant words are defined in his work as words of medium frequency in the document, where "medium" means that the frequency is between predefined high-frequency and low-frequency cutoff values.
- Given the significant words, portions of the sentence that are "bracketed" by these words are considered, with a limit set for the number of non-significant words that can be between two significant words (typically four). The significance factor for these bracketed text spans is computed by dividing the square of the number of significant words in the span by the total number of words. Figure 6.4 gives an example of a text span for which the significance factor is $4^2/7 = 2.3$. The significance factor for a sentence is the maximum calculated for any text span in the sentence.

w w w w w w w w w w w.
(Initial sentence)

w w s w s s w w s w w.
(Identify significant words)

w w [s w s s w w s] w w.
(Text span bracketed by significant words)

Fig. An example of a text span of words (w) bracketed by significant words (s) using Luhn's algorithm

- To be more specific about the definition of a significant word, the following is a frequency-based criterion that has been used successfully in more recent research. If *fd;w* is the frequency of word *w* in document *d*, then *w* is a significant word if it is not a stopword (which eliminates the high-frequency words), and

$$f_{d,w} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), \text{ if } s_d < 25 \\ 7, \qquad\qquad\qquad \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), \text{ otherwise,} \end{cases}$$

- Where s_d is the number of sentences in document *d*. As an example, consider the page contains less than 25 sentences (roughly 20), and so the significant words will be non-stopwords with a frequency greater than or equal to 6.5. The only words that satisfy this criterion are "information" (frequency 9), "story" (frequency 8), and "text" (frequency 7).
- Most work on summarization since Luhn has involved improvements to this basic approach, including better methods of selecting significant words and selecting sentences or sentence fragments. Snippet generation techniques can also be viewed as variations of Luhn's approach with query words being used as the significant words and different sentence selection criteria.
- Typical features that would be used in selecting sentences for snippets to summarize a text document such as a news story would include whether the sentence is a heading, whether it is the first or second line of the document, the total number of query terms occurring in the sentence, the number of unique query terms in the sentence, the longest contiguous run of query words in the sentence, and a density measure of query words, such as Luhn's significance factor.
- In this approach, a weighted combination of features would be used to rank sentences. Web pages, however, often are much less structured than a news story, and can contain a lot of text that would not be appropriate for snippets. To address this, snippet sentences are often selected from the metadata associated with the web page, such as the "description" identified by the <meta name="description" content= ...>
- HTML tags, or from external sources, such as web directories.Certain classes of web pages, such as Wikipedia entries, are more structured and have snippet sentences selected from the text.

- Although many variations are possible for snippet generation and document summaries in result pages, some basic guidelines for effective summaries have been derived from an analysis of clickthrough data . The most important is that whenever possible, all of the query terms should appear in the summary, showing their relationship to the retrieved page.
- When query terms are present in the title, however, they need not be repeated in the snippet. This allows for the possibility of using sentences from metadata or external descriptions that may not have query terms in them. Another guideline is that URLs should be selected and displayed in a manner that emphasizes their relationship to the query by, for example, highlighting the query terms present in the URL.
- Finally, search engine users appear to prefer readable prose in snippets (such as complete or near-complete sentences) rather than lists of keywords and phrases. A feature that measures readability should be included in the computation of the ranking for snippet selection.
- The efficient implementation of snippet generation will be an important part of the search engine architecture since the obvious approach of finding, opening, and scanning document files would lead to unacceptable overheads in an environment requiring high query throughput.
- Instead, documents must be fetched from a local document store or cache at query time and decompressed. The documents that are processed for snippet generation should have all HTML tags and other "noise" (such as Javascript) removed, although metadata must still be distinguished from text content. In addition, sentence boundaries should be identified and marked at indexing time, to avoid this potentially time-consuming operation when selecting snippets.

4.13. QUESTION ANSWERING (Q&A) ON THE WEB

- **Ask Jeeves'** original mission, as set out in 1996, when it was founded, was to provide natural language question answering on the Web. Its founders, venture capitalist Garrett Gruener and technologist David Warthen, came up with the idea of using P.G. Wodehouse's butler character "Jeeves" as their public face, conveying quality of service together with friendliness and integrity. This branding has been very successful, as the Ask Jeeves logo is clearly memorable in searchers' minds. There was a price to pay for the use of the Jeeves character, since they were eventually sued early in 2000 by the owners of the copyrightto Wodehouse's novels, and we assume that there was an amicable out of court settlement.
- In 2006, Jeeves retired from his services at the search engine Ask Jeeves, which was rebranded simply as Ask (www.ask.com). In 2009, Jeeves has come out of retirement to revitalize the brand on the UK site and on askjeeves. Com

1) Natural Language Annotations

- Natural language processing solutions are still very much in the research labs, especially when dealing with an open domain such as the Web. In the Q&A space Ask Jeeves has progressively moved away from an editorially driven knowledge base used to derive answers to user queries, to being able to answer queries directly by tapping into existing

structured information resources (i.e., databases compiled by human editors) or by mining information from web pages, which are unstructured information sources.

- The editorially driven approach, which is still of interest, is to build and maintain a humanly edited knowledge base containing questions that users are likely to ask, based on query logs of web searchers, in the hope of matching a searcher's question with an existing template that will link to an answer from the knowledge base.
- As already mentioned, such a knowledge base is maintained by humans rather than being automated, so its coverage in comparison to a crawler based search engine such as Google, Yahoo, or Bing is very low. We are talking about a knowledge base of a few million entries compared to a search engine index containing billions of web pages. Interestingly enough, at the end of 1999, Ask Jeeves were sued by Boris Katz and Patrick Winston from the MIT Artificial Intelligence Laboratory, for infringement of two natural language patents issued in 1994 and 1995 on generating and utilizing natural language annotations to facilitate text retrieval.
- The natural language annotations correspond to the former Ask Jeeves question templates, that pointed to answers from the knowledge base. The case was successfully settled before the trial, presumably through some licensing agreement.
- The Artificial Intelligence Lab maintains its own natural language question answering system called **START**, which can be found at http://start.csail.mit.edu.
- START was the first Q&A system to become available on the Web, as it has been operating since the end of 1993. It is based on the idea of natural language annotations, which are sentences and phrases in computer analyzable form that point to relevant documents. Thus, a question such as "who is the president of the USA?" matches an annotation in the knowledge base yielding the correct answer at this moment in time, and pointing to the source of the answer.
- START taps into a wealth of online resources on the Web by employing annotations that point to specific databases that are most likely to contain the answer. So, for example, if the question is about movies, START retrieves possible answers from the Internet Movie Database (www.imdb.com). To annotate a substantial portion of the information on the Web, much more than a handful of annotators will be required.
- One suggestion is to enlist millions of web users to help out in a collaborative effort through a simple tool that would allow users to contribute to the knowledge base by annotating web pages they stumble upon. Ask Jeeves' Q&A technology is based on extracting answers to questions through a combination of natural language processing and Teoma's algorithmic technology.
- On the one hand, "**smart answers"** extracts information from structured data sources, while the newer **"web answers"** attempts to find answers directly from web pages. This approach signifies a move by Ask Jeeves toward open Q&A.
- Ask Jeeves had an immensely successful IPO in 1999 and by early 2000, it was in the top 25 most popular destinations of web surfers. It nearly collapsed after the Internet bubble burst, but has picked up the pieces, mainly through paid placement advertising and its acquisition, in March 2004, of Interactive Search Holding, whose web properties included Excite.As of 2009, it handled about 4% of web searches, which is a respectable share for a niche player, in a very competitive yet lucrative market.

- Originally Ask Jeeves was a metasearch engine as it did not own proprietary search engine technology. Eventually it transformed itself by acquiring the search engine Teoma in September 2001 for under $4 million, which seems like a bargain compared to the $500 million price tag for acquiring Direct Hit in January 2000. In July 2005 it was acquired by the e-commerce conglomerate IAC/InterActiveCorp for a price tag of $1.85 billion.
- If Ask Jeeves can compute the answer to a question with confidence, it will present the answer. So when you type the query "who is the prime minister of the uk?" into the search box at www.ask.com the answer will appear at the top of the search results page, stating that, as of early 2010, "The Chief of State of the United Kingdom is Queen Elizabeth II, and the Head of State is Prime Minister James Gordon Brown" .
- In any case, the question to Ask Jeeves is also fired as a query to its underlying search engine, Teoma, and the search results are integrated into the results page presented to the user. Below the answer to the question, you will find a list of sponsored links, if there are any for the particular query, and below these the organic results from Teoma will be displayed. Other search engines are now also providing a limited question answering facility. Typical queries to Google such as "weather in london" or "time in london" produce the desired answer above the organic results.
- For a standard query Ask Jeeves behaves much the same as any other search engine, as can be seen when you type the query "computer chess" into its search box. First some sponsored links are displayed, following these the first batch of organic web results is displayed, and at the bottom of the screen additional sponsored links are shown.

Figure . Question "who is the prime minister of the uk?" submitted to Ask Jeeves.

- Ask Jeeves' user interface has a look and feel that is similar to that of other search engines. In addition, related searches are displayed on the righthand side of the results

page, where users can refine their search or explore related topics. These refinements use clustering technology based on Teoma's variant of the HITS algorithm, and may well incorporate information from query logs as utilized by the popularity search engine Direct Hit.

2) Factual Queries

- **Wolfram Alpha** (www.wolframalpha.com), released to the public in mid-2009, aims to make knowledge computable and accessible. The data that is used by the knowledge engine is domain specific and structured, rather than open web crawled data, and is curated, that is, collected under the supervision of one or more experts. The knowledge engine accepts natural language as input but will also accept a mathematical formula that it will then attempt to solve.
- It is implemented in Mathematica, which is the flagship product of Wolfram Research, and has capabilities in many domains, with the data coming from different sources.
- Its goal is to provide definitive answers to factual queries. So, for the query "who is the prime minister of the uk?" it will present a template with the answer and various related facts as of early 2010.
- Bing has teamed up with Wolfram Alpha to enrich its search results in select areas across nutrition, health, and advanced mathematics.One criticism of Wolfram Alpha is that it is a black box in comparison to the major search engines, where the primary source of information is made clear, and so its results are hard to verify.
- Google squared (www.google.com/squared) is a tool that collects facts from the open web and presents them in a tabular format, with a row for each fact and a column for each attribute derived by the tool. It is not in direct competition with Wolfram Alpha as its data sources are not curated and its function is to compile information into a structured table rather than to provide a computational knowledge engine that can derive and display information in different forms.

3) Open Domain Question Answering

- Question answering is not new, for example in Salton and McGill's now classical 1983 text on information retrieval, they cover Q&A concentrating on the natural language aspects. The Web as a corpus for Q&A, is an open domain without any subject boundaries. The main problem here is that although the Web is a tremendous resource of information, the quality of documents is extremely variable, it is often out of date, and not always factual.
- Despite this shortcoming the Web has a huge asset, which is data redundancy. If you ask a question such as "what is the capital of the uk?", assuming we can extract relevant phrases from web pages, then by a simple voting system we expect the correct answer to emerge as the majority answer.
- Some prototype Q&A systems, which are now surfacing out of research labs, are attempting to tackle the open domain question answering on the Web using this voting technique. This approach, which builds on the fact that in a huge corpus such as the open web, the correct answer is likely to be stated many times, in multiple ways, and in multiple documents, and is known as the *redundancy-based method*.

- One system based on this method is Microsoft's **AskMSR**, which may be deployed in a future Microsoft product. AskMSR avoids sophisticated natural language processing in favor of simple phrase analysis with question template matching, and the use of a search engine to retrieve snippets that may contain the answer.
- AskMSR first parses the input question to generate a series of queries to a backend search engine (they have used Google for this purpose), based on specific question templates and keyword matches.
- The problem of transforming the input question into a set of search engine queries that are likely to return results containing the answer, is a difficult problem, and machine learning techniques are currently being enlisted to solve this task .
- The queries output are then submitted to the search engine, and the answers returned are ranked according to how likely they are to contain the answer, which is a combination of how well they match an answer template, and the number of snippets from the web pages in which the answer occurs. This approach is proving to be successful, although at the moment it mainly deals with factual questions such as "who is the president of the USA?" rather than more complex question such as "is the usa out of recession?", which would involve advanced reasoning.

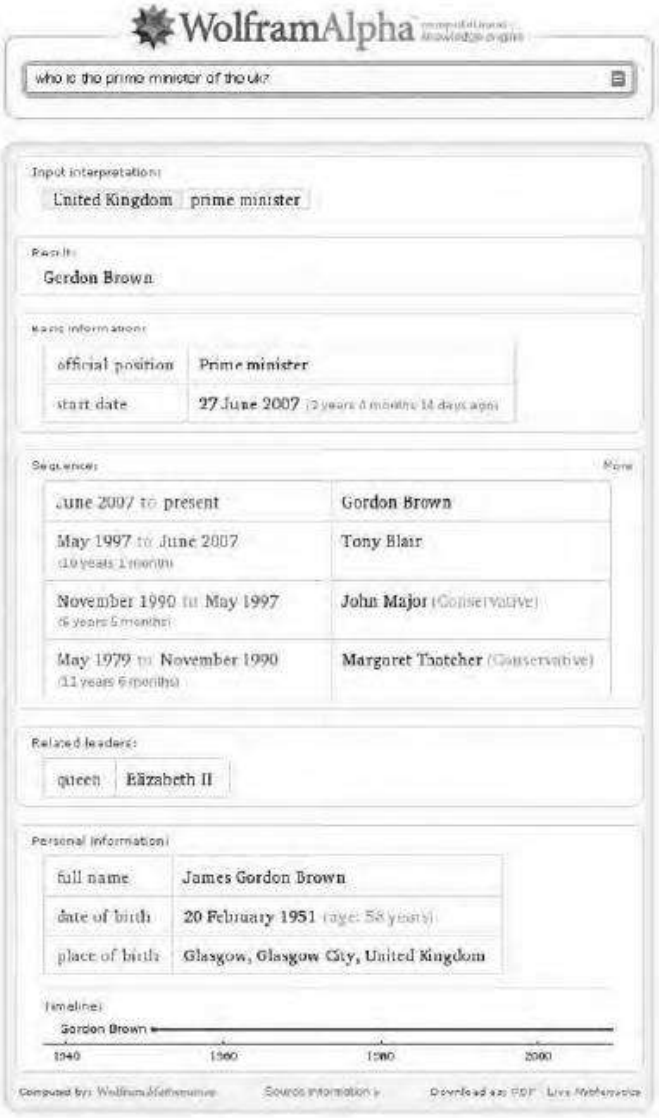


Figure . Query "who is the prime minister of the uk?" submitted to Wolfram Alpha.

- Named entity recognition, which is a subtask of information extraction, is useful in Q&A systems to pinpoint answers in text snippets. The goal of named entity extraction is to locate and classify small units of text into predefined categories such as names of people, organizations, and locations.
- In the context of Q&A, named entities are natural candidate answers to questions. As an example, for the question, "who is the prime minister of the UK?", the named entity recognizer will determine that the answer should be a person. Using this information it is possible to filter out all snippets that do not have a person entity in them. The redundancy-based method can then be used to find the majority answer as the most likely one.
- Mulder is another experimental Q&A interface to a search engine, which was developed by researchers from the University of Washington within the same research group that

developed MetaCrawler and Grouper . Mulder uses more sophisticated natural language parsing than AskMSR by modifying existing natural language processing tools, and using many other heuristics including a voting procedure to rank candidate answers.

- Mulder also uses Google as its backend search engine, to which it submits queries that it believes will contain an answer to the question at hand. Mulder was evaluated against Ask Jeeves for Q&A tasks and against Google as a baseline comparison.
- The results showed that, overall, Mulder outperformed Ask Jeeves, which was due to Ask Jeeves' method, at the time, being based on human edited annotations of a relatively small number of web pages, rather than being fully automated and using a web search engine as its corpus, in the way that Mulder and AskMSR do. It is not surprising that Mulder also outperforms Google on Q&A tasks, which again is as we would expect, since Google was not designed as a Q&A system.

4) Semantic Headers

- In 2003 a colleague of mine, Boris Galitsky, wrote a book on natural language question answering, which emphasizes a semantic approach based on logic programming, rather than a template-based approach as Ask Jeeves had employed in the past .
- The *semantic header* approach, as Boris calls it, represents potential answers to questions through relationships between objects that are relevant to the domain under consideration. So, for example in an e-commerce domain the objects may be of type product, customer, vendor, order, shopping basket, and so on, and the relationships between the objects may be of type purchase (between customer and product), and payment (between customer and vendor).
- The semantic header approach is designed to work in closed domains such as e-commerce, tax, law, or medicine, rather than in an open domain such as the Web as a whole.
- This makes the task of building the semantic headers feasible, so that the number of potential answers the system has to cater for is of the order of tens of thousands (this number was specifically discovered to be useful in financial domains).
- The technique of building the knowledge base of semantic headers is done in cooperation with domain experts, and is improved on a regular basis from feedback obtained by inspecting log data of customers using the service.
- In this sense the system is semiautomatic, much in the same way the Ask Jeeves service used to be, but it requires substantial human effort to maintain the knowledge base and keep it up to date. The payback from the semantic header approach is that it is potentially more accurate and less prone to error, than the syntactic natural language approach, which has no "understanding" of the domain.
- Boris' approach is very much in the spirit of rule-based expert systems, which are still a very important topic in artificial intelligence. The tax advisor answering system he developed was deployed in a commercial setting in 2000 on the CBS Market Watch site, which publishes business news and information.
- An analysis of the system showed that over 95% of the customers and quality assurance staff agreed that the advisor was a preferred method for nonprofessional users accessing the information.

4.14. CROSS-LINGUAL RETRIEVAL.

- By translating queries for one or more monolingual search engines covering different languages, it is possible to do *cross-language* search. A cross-language search engine receives a query in one language (e.g., English) and retrieves documents in a variety of other languages (e.g., French and Chinese).
- Users typically will not be familiar with a wide range of languages, so a cross language search system must do the query translation automatically. Since the system also retrieves documents in multiple languages, some systems also translate these for the user.

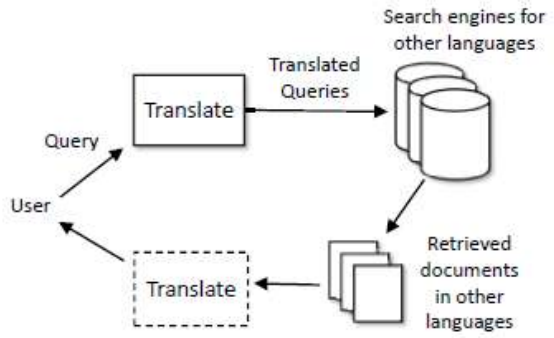


Fig. Cross-language search

- The most obvious approach to automatic translation would be to use a large bilingual dictionary that contained the translation of a word in the source language (e.g., English) to the target language (e.g., French). Sentences would then be translated by looking up each word in the dictionary.
- The main issue is how to deal with ambiguity, since many words have multiple translations. Simple dictionary based translations are generally poor, but a number of techniques have been developed, such as query expansion, that reduce ambiguity and increase the ranking effectiveness of a cross-language system to be comparable to a monolingual system.
- The most effective and general methods for automatic translation are based on ***statistical machine translation models*.** When translating a document or a web page, in contrast to a query, not only is ambiguity a problem, but the translated sentences should also be grammatically correct. Words can change order, disappear, or become multiple words when a sentence is translated.
- Statistical translation models represent each of these changes with a probability. This means that the model describes the probability that a word is translated into another word, the probability that words change order, and the probability that words disappear or become multiple words. These probabilities are used to calculate the most likely translation for a sentence.
- Although a model that is based on word-to-word translation probabilities has some similarities to a dictionary-based approach, if the translation probabilities are accurate, they can make a large difference to the quality of the translation. Unusual translations for an ambiguous word can then be easily distinguished from more typical translations.

- More recent versions of these models, called ***phrase based translation models,*** further improve the use of context in the translation by calculating the probabilities of translating sequences of words, rather than just individual words. A word such as "flight", for example, could be more accurately translated as the phrase "commercial flight", instead of being interpreted as "bird flight".
- The probabilities in statistical machine translation models are estimated primarily by using *parallel corpora*. These are collections of documents in one language together with the translations into one or more other languages. The corpora are obtained primarily from government organizations (such as the United Nations), news organizations, and by mining the Web, since there are hundreds of thousands of translated pages.
- The sentences in the parallel corpora are *aligned* either manually or automatically, which means that sentences are paired with their translations. The aligned sentences are then used for training the translation model.
- Special attention has to be paid to the translation of unusual words, especially proper nouns such as people's names. For these words in particular, the Web is a rich resource. Automatic *transliteration* techniques are also used to address the problem of people's names.
- Proper names are not usually translated into another language, but instead are transliterated, meaning that the name is written in the characters of another language according to certain rules or based on similar sounds. This can lead to many alternative spellings for the same name.
- Although they are not generally regarded as cross-language search systems, web search engines can often retrieve pages in a variety of languages. For that reason, many search engines have made translation available on the result pages.
- Figure shows an example of a page retrieved for the query "pecheur france", where the translation option is shown as a hyperlink. Clicking on this link produces a translation of the page (not the snippet), which makes it clear that the page contains links to archives of the sports magazine *Le pêcheur de France*, which is translated as "The fisherman of France". Although the translation provided is not perfect, it typically provides enough information for someone to understand the contents and relevance of the page. These translations are generated automatically using machine translation techniques, since any human intervention would be prohibitively expensive.

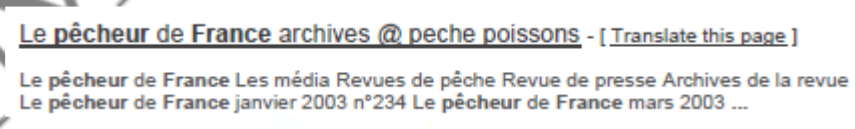


FIG. A French web page in the results list for the query "pecheur france"

CS6007 INFORMATION RETRIEVAL
UNIT-IV

PART-A

1. What is the purpose of link analysis?

Link analysis is one of many factors considered by web search engines in computing a composite score for a web page on any given query.

2. What is the use of PageRank?

Link analysis assigns to every node in the PageRank web graph a numerical score between 0 and 1, known as its PageRank. The PageRank of a node depends on the link structure of the web graph. Given a query, a web search engine computes a composite score for each web page that combines hundreds of features such as cosine similarity and term proximity together with the PageRank score. This composite score is used to provide a ranked list of results for the query.

3. What is meant by teleport operation?

- In the teleport operation, the surfer jumps from a node to any other node in the web graph. This could happen because he types an address into the URL bar of his browser. The destination of a teleport operation is modeled as being chosen uniformly at random from all web pages.

4. State the discrete-time stochastic process.

- A Markov chain is a discrete-time stochastic process, a process that occurs in a series of time steps in each of which a random choice is made. A Markov chain consists of N states. Each web page will correspond to a state in the Markov chain we will formulate.

5. What do you meant by hubs and authorities?

- Given a query, every web page is assigned two scores. One is called its hub score and the other its authority score.
- For any query, we compute two ranked lists of results rather than one. The ranking of one list is induced by the hub scores and that of the other by the authority scores. This approach stems from a particular insight into the creation of web pages, namely, that there are two primary kinds of web pages useful as results for broad-topic searches.
- By a broad topic search we mean an informational query such as "I wish to learn about leukemia." There are authoritative sources of information on the topic; in this case, the National Cancer Institute's page on leukemia would be such a page. such pages are called authorities; they are the pages that will emerge with high authority scores.

6. What is the use of HITS?

HITS (hyperlink-induced topic search) is based on the idea that a good authority is pointed to by good hubs, and a good hub points to good authorities. HITS uses subgraph to determine the hubs and authorities for the input query.

HITS can be used in the following ways:

(i) To find related pages by setting the root set to contain a single page, and up to 200 pages that point to it.

(ii) To help categorize web pages by starting with a root set of pages having a known category.
(iii) To find web communities defined as densely linked focused subgraphs.
(iv) To study citation patterns between documents.

7. What is the purpose of mapreduce?
The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time.

8. What is meant by personalization?
Search engines collect a huge amount of data from user queries. This together with the use of cookies to identify returning users, and utilities such as the search toolbar, which are installed on users' browsers, put search engines in an excellent position to provide each user with a personalized search interface tailored to the individual needs of that user.

9. How personalization is performed in click based approach?

In a click-based approach , the user's query and click pairs are used for personalization. The idea is simple. When a user repeats queries over time, he or she will prefer certain pages, that is, those that were more frequently clicked. The downside of this approach is that if a search engine presents the same old pages to the user each time a query is repeated it does not encourage the user to discover new pages. On the other hand, this type of historical information may be quite useful to the user. This approach can be refined by using content similarity to include similar queries and web pages in the personalized results.

10. How personalization is performed in topic based approach?
- In a topic-based approach, a topical ontology is used to identify a user's interests. The ontology should include general topics that are of interest to web surfers such as the top-level topics from the Open Directory. Then a classification technique, such as naive Bayes, needs to be chosen in order to be able to classify the queries that users submit and the pages that they visit. The next step is to identify the user's preferences based on their searches, and finally these preferences can be used to personalize their results, for example, by ranking them according to the learned preferences.

11. What is the use of Personalized Results Tool?
The Personalized Results Tool (PResTo!) is implemented as a plug-in to the browser rather than being server based . This is a unique feature that bypasses some of the privacy and security issues, which are becoming increasingly important to users, since in the case of PResTo!, the ownership of the software and the personal data generated from searches are in the user's hands.A client-side approach is also more efficient for the search engine, since it does not have to manage the user profiles, and thus scalability will not be an issue.

12. How can you do relevance feedback?

Relevance feedback can be done by having two radio buttons next to each ranked document, one for specifying the document as being relevant and the other for specifying it as nonrelevant.

13. What is meant by pseudorelevance feedback?

Pseudo relevance feedback, assumes that the top-ranked documents are marked as relevant, and thus does not need any user feedback as such. Pseudorelevance feedback is also called blind feedback, since once the query is submitted, no additional input from the user is needed.

14. What is meant by implicit feedback?

Modern users would not be willing to invest the effort to give feedback, and the additional burden on the search engine's servers would be overwhelming. The way forward is to collect implicit user feedback from the users' clickstream data.

15. What is meant by topic sensitive pagerank?

PageRank is topic sensitive. This version of PageRank is biased according to some representative set of topics, based on categories chosen, say, from the Open Directory . These could be biased toward the topics that the user prefers to explore, so if, for example, a user prefers sports over world news, this preference would translate to a higher probability of jumping to sports pages than to world news pages.

16. What do you meant by query-dependent PageRank

Biasing the PageRank toward specific queries, is called query-dependent PageRank .

17. What is the use of blockrank?

- ***BlockRank***, computes local PageRank values on a host basis, and then weights these local PageRank values according to the global importance of the host. BlockRank takes advantage of the fact that a majority of links (over 80% according to the researchers) are within domains rather than between them, and domains such as stanford.edu, typically contain a number of hosts.

18. Define *collaborative filtering*.

- Filtering that considers the relationship between profiles (or between users) and uses this information to improve how incoming items are matched to profiles (or users) is called *collaborative filtering*.

19. What is an invisible web?

- The vast majority of the Internet lies in the Deep Web, sometimes referred to as the Invisible Web. The actual size of the Deep Web is impossible to measure, but many experts estimate it is about 500 times the size of the web as we know it.Deep Web pages operate just like any other site online, but they are constructed so that their existence is invisible to Web crawlers.

20. What do you mean by snippets?

A document summary for a web search typically contains the title and URL of the web page, links to live and cached versions of the page, and, most importantly, a short text summary, or *snippet*, that is used to convey the content of the page.

21. What is the use of cross language search?

Cross-language search engine receives a query in one language (e.g., English) and retrieves documents in a variety of other languages (e.g., French and Chinese).

PART-B

UNIT-4

1. How will you do link analysis.(4)
2. Write about hubs and authorities.(8)
3. How the Page Rank values are calculated.(8)
4. Explain about the HITS algorithms.(6)
5. Explain the structure of Hadoop.(8)
6. Explain the operation performed in two phases of Map Reduce.(8)
7. How will you do personalized search.(16)
8. Explain the real time application of Collaborative filtering.(8)
9. Explain about the way of doing content-based retrieval.(8)
10. How will you handle "invisible" Web.(8)
11. Explain about Summarization and Snippet generation.(10)
12. What are the search engines are available for Question Answering. Explain it.(12)
13. What is the use of Cross-Lingual Retrieval and explain the architecture in detail.(10)

UNIT V RECOMMENDER SYSTEM

Recommender Systems Functions – Data and Knowledge Sources – Recommendation Techniques – Basics of Content-based Recommender Systems – High Level Architecture – Advantages and Drawbacks of Content-based Filtering – Collaborative Filtering – Matrix factorization models – Neighborhood models.

5.1. INFORMATION FILTERING

- One part of social search applications is representing individual users' interests and preferences. One of the earliest applications that focused on user profiles was document filtering. Document filtering, often simply referred to as *filtering*, is an alternative to the standard ad hoc search paradigm. In ad hoc search, users typically enter many different queries over time, while the document collection stays relatively static.
- In filtering, the user's information need stays the same, but the document collection itself is dynamic, with new documents arriving periodically. The goal of filtering, then, is to identify (filter) the relevant new documents and send them to the user. Filtering is a *push* application.
- Filtering is also an example of a supervised learning task, where the profile plays the part of the training data and the incoming documents are the test items that need to be classified as "relevant" or "not relevant." However, unlike a spam detection model, which would take thousands of labeled emails as input, a filtering system profile may only consist of a single query, making the learning task even more challenging. For this reason, filtering systems typically use more specialized techniques than general classification techniques in order to overcome the lack of training data.
- Although they are not as widely used as standard web search engines, there are many examples of real-world document filtering systems.
- For example, many news sites offer filtering services. These services include alerting users when there is breaking news, when an article is published in a certain new category (e.g., sports or politics), or when an article is published about a certain topic, which is typically specified using one or more keywords (e.g., "terrorism" or "global warming"). The alerts come in the form of emails, SMS (text messages), or even personalized newsfeeds, thereby allowing the user to keep up with topics of interest without having to continually check the news site for updates or enter numerous queries to the site's search engine.
- Therefore, filtering provides a way of personalizing the search experience by maintaining a number of long-term information needs.
- Document filtering systems have two key components.
    1) First, the user's long term information needs must be accurately represented. This is done by constructing a *profile* for every information need.
    2) Second, given a document that has just arrived in the system, a decision mechanism must be devised for identifying which are the relevant profiles for that document.

This decision mechanism must not only be efficient, especially since there are likely to be thousands of profiles, but it must also be highly accurate. The filtering system should not miss relevant documents and, perhaps even more importantly, should not be continually alerting users about non-relevant documents.

Profiles:

- In web search, users typically enter a very short query. The search engine then faces the daunting challenge of determining the user's underlying information need from this very sparse piece of information. There are numerous reasons why most search engines today expect information needs to be specified as short keyword queries. However, one of the primary reasons is that users do not want to(or do not have the time to) type in long, complex queries for each and every one of their information needs.
- Many simple, non-persistent information needs can often be satisfied using a short query to a search engine. Filtering systems, on the other hand, cater to long-term information needs. Therefore, users may be more willing to spend more time specifying their information need in greater detail in order to ensure highly relevant results over an extended period of time. The representation of a user's long-term information need is often called a *filtering profile* or just a *profile*.
- What actually makes up a filtering profile is quite general and depends on the particular domain of interest. Profiles may be as simple as a Boolean or keyword query. Profiles may also contain documents that are known to be relevant or non-relevant to the user's information need. Furthermore, they may contain other items, such as social tags and related named entities. Finally, profiles may also have one or more relational constraints, such as "published before 1990", "price in the$10–$25 range", and so on. Whereas the other constraints described act as soft filters, relational constraints of this form act as hard filters that must be satisfied in order for a document to be retrieved.
- Although there are many different ways to represent a profile, the underlying filtering model typically dictates the actual representation.
- Filtering models are very similar to the retrieval models. In fact, many of the widely used filtering models are simply retrieval models where the profile takes the place of the query.
- There are two common types of filtering models.
    - The first are *static* models. Here, static refers to the fact that the user's profile does not change over time, and therefore the same model can always be applied.
    - The second are *Adaptive* models, where the user's profile is constantly changing over time. This scenario requires the filtering model to be dynamic over time as new information is incorporated into the profile.

Static filtering models

- *Static filtering models* work under the assumption that the filtering profile remains static over time. In some ways, this makes the filtering process easier, but in other ways it makes it less robust. All of the popular static filtering models are derived from the standard retrieval models. However, unlike web search, filtering systems do not return a ranked list of documents for each profile. Instead, when a new document enters the

system, the filtering system must decide whether or not it is relevant with respect to each profile.

- Figure illustrates how a static filtering system works. As new documents arrive, they are compared to each profile. Arrows from a document to a profile indicate that the document was deemed relevant to the profile and returned to the user.

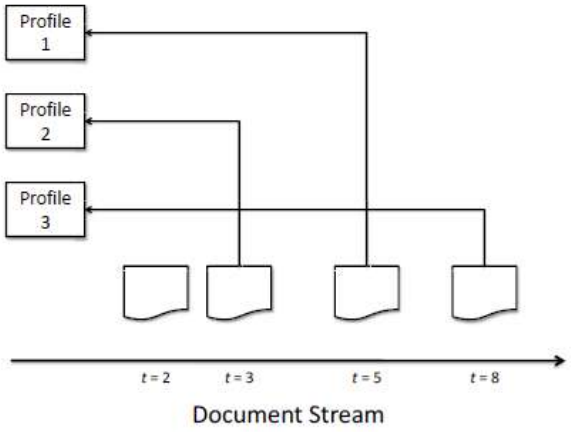


Fig. Example of a static filtering system. Documents arrive over time and are compared against each profile. Arrows from documents to profiles indicate the document matches the profile and is retrieved.

- In the most simple case, a Boolean retrieval model can be used. Here, the filtering profile would simply consist of a Boolean query, and a new document would be retrieved for the profile only if it satisfied the query.
- The Boolean model, despite its simplicity, can be used effectively for document filtering, especially where precision is important. In fact, many web-based filtering systems make use of a Boolean retrieval model.
- One of the biggest drawbacks of the Boolean model is the low level of recall. Depending on the filtering domain, users may prefer to have good coverage over very high precision results. There are various possible solutions to this problem, including using the vector space model, the probabilistic model, BM25, or language modeling.
- All of these models can be extended for use with filtering by specifying a profile using a keyword query or a set of documents. Directly applying these models to filtering, however, is not trivial, since each of them returns a score, not a "retrieve" or "do not retrieve" answer as in the case of the Boolean model.
- One of the most widely used techniques for overcoming this problem is to use a score threshold to determine whether to retrieve a document. That is, only documents with a similarity score above the threshold will be retrieved. Such a threshold would have to be tuned in order to achieve good effectiveness. Many complex issues arise when applying a global score threshold, such as ensuring that scores are comparable across profiles and over time.
- Given a static profile, which may consist of a single keyword query, multiple queries, a set of documents, or some combination of these, we must first estimate a profile language model denoted by *P*. There are many ways to do this. One possibility is:

$$P(w|P) = \frac{(1-\lambda)}{\sum_{i=1}^{K} \alpha_i} \sum_{i=1}^{K} \alpha_i \frac{f_{w,T_i}}{|T_i|} + \lambda \frac{c_w}{|C|}$$

- Where $T_1, \ldots, T_k$ are the pieces of text (e.g., queries, documents) that make up the profile, and α_i is the weight (importance) associated with text T_i. Then, given an incoming document, a document language model (D) must be estimated. Estimate D using the following:

$$P(w|D) = (1-\lambda)\frac{f_{w,D}}{|D|} + \lambda \frac{c_w}{|C|}$$

- Documents can then be ranked according to the negative KL-divergence between the profile language model (P) and the document language model (D) as follows:

$$-KL(P||D) = \sum_{w \in V} P(w|P) \log P(w|D) - \sum_{w \in V} P(w|P) \log P(w|P)$$

- Document D is then delivered to profile P if $-KL(P||D) \geq t$, where t is some relevance threshold.
- Document filtering can also be treated as a machine learning problem. At its core, filtering is a classification task that often has a very small amount of training data (i.e., the profile). The task is then to build a binary classifier that determines whether an incoming document is relevant to the profile. However, training data would be necessary in order to properly learn such a model. For this task, the training data comes in the form of binary relevance judgments over profile/document pairs. Any of the classification techniques can be used.
- Suppose that a Support Vector Machine with a linear kernel is used; the scoring function would then have the following form:

$$s(P;D) = w \cdot f(P,D) = w_1 f_1(P,D) + w_2 f_2(P,D) + \ldots + w_d f_d(P,D)$$

- where $w_1, \ldots, w_d$ are the set of weights learned during the SVM training process, and $f_1(P,D), \ldots, f_d(P,D)$ are the set of features extracted from the profile/document pair. Many of the features that have been successfully applied to text classification, such as unigrams and bigrams, can also be applied to filtering.
- Given a large amount of training data, it is likely that a machine learning approach will outperform the simple language modeling approach. However, when there is little or no training data, the language modeling approach is a good choice.

Adaptive filtering models

- Static filtering profiles are assumed not to change over time. In such a setting, a user would be able to create a profile, but could not update it to better reflect his information need. The only option would be to delete the profile and create a new one that would hopefully produce better results. This type of system is rigid and not very robust.

- *Adaptive filtering* is an alternative filtering technique that allows for dynamic profiles. This technique provides a mechanism for updating the profile over time.

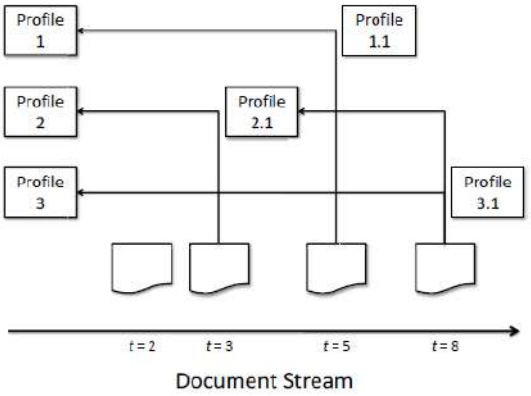


Fig. Example of an adaptive filtering system. Documents arrive over time and are compared against each profile. Arrows from documents to profiles indicate the document matches the profile and is retrieved. Unlike static filtering, where profiles are static overtime, profiles are updated dynamically (e.g., when a new match occurs).

- Profiles may be either updated using input from the user or done automatically based on user behavior, such as click or browsing patterns. There are various reasons why it may be useful to update a profile as time goes on. For example, users may want to fine-tune their information need in order to find more specific types of information. Therefore, adaptive filtering techniques are more robust than static filtering techniques and are designed to adapt to find more relevant documents over the life span of a profile. Unlike the static filtering case, when a document is delivered to a profile, the user provides feedback about the document, and the profile is then updated and used for matching future incoming documents.
- As Figure suggests, one of the most common ways to adapt a profile is in response to user feedback. User feedback may come in various forms, each of which can be used in different ways to update the user profile.
- In order to provide a concrete example of how profiles can be adapted in response to user feedback, we consider the case where users provide *relevance feedback* on documents. That is, for some set of documents, such as the set of documents retrieved for a given profile, the user explicitly states whether or not the document is relevant to the profile.
- Given the relevance feedback information, there are a number of ways to update the profile. As before, how the profile is represented and subsequently updated largely depends on the underlying retrieval model that is being used.
- The Rocchio algorithm can be used to perform relevance feedback in the vector space model. Therefore, if profiles are represented as vectors in a vector space model, Rocchio's algorithm can be applied to update the profiles when the user provides relevance feedback information. Given a profile P, a set of non-relevant feedback documents (denoted *Nonrel*), and a set of relevant feedback documents (denoted *Rel*), the adapted profile P' is computed as follows:

$$P' = \alpha.P + \beta.\frac{1}{|Rel|}\sum_{D_i \in Rel} D_i - \gamma.\frac{1}{|Nonrel|}\sum_{D_i \in Nonrel} D_i$$

- Where *Di* is the vector representing document *i*, and α, β, and γ are parameters that control how to trade off the weighting between the initial profile, the relevant documents, and the non-relevant documents.
- *Relevance models* can be used with language modeling for pseudo-relevance feedback. However, relevance models can also be used for true relevance feedback as follows:

$$P(w|P) = \frac{1}{|Rel|}\sum_{D_i \in Rel}\sum_{D \in C} P(w|D)P(D_i|D)$$
$$\approx \frac{1}{|Rel|}\sum_{D_i \in Rel} P(w|D_i)$$

- Where *C* is the set of documents in the collection, *Rel* is the set of documents that have been judged relevant, *Di* is document *i*, and *P*(*Di*|*D*) is the probability that document *Di* is generated from document *D*'s language model. The approximation (≈) can be made because *Di* is a document, and *P*(*Di*|*D*) is going to be 1 or very close to 1 when *Di*= *D* and nearly 0 for most other documents. Therefore, the probability of *w* in the profile is simply the average probability of *w* in the language models of the relevant documents. Unlike the Rocchio algorithm, then on-relevant documents are not considered.
- If a classification technique is used for filtering, then an *online learning* algorithm can be used to adapt the classification model as new user feedback arrives. Online learning algorithms update model parameters, such as the hyper plane *w* in SVMs, by considering only one new item or a batch of new items.
- These algorithms are different from standard supervised learning algorithms because they do not have a "memory." That is, once an input has been used for training, it is discarded and cannot be explicitly used in the future to update the model parameters. Only the new training inputs are used for training.

Model	*Profile Representation*	*Profile Updating*
Boolean	Boolean Expression	N/A
Vector Space	Vector	Rocchio
Language Modeling	Probability Distribution	Relevance Modeling
Classification	Model Parameters	Online Learning

Table .Summary of static and adaptive filtering models. For each, the profile representation and profile updating algorithm are given.

- Both static and adaptive filtering, therefore, can be considered special cases of many of the retrieval models and techniques. Table summarizes the various filtering models, including how profiles are represented and updated.
- In practice, the vector space model and language modeling have been shown to be effective and easy to implement, both for static and adaptive filtering. The classification models are likely to be more robust in highly dynamic environments. However, as with all classification techniques, the model requires training data to learn an effective model.

Fast filtering with millions of profiles:

- In a full-scale production system, there may be thousands or possibly even millions of profiles that must be matched against incoming documents. Fortunately, standard information retrieval indexing and query evaluation strategies can be applied to perform this matching efficiently. In most situations, profiles are represented as a set of keywords or a set of feature values, which allows each profile to be indexed.
- Scalable indexing infrastructures can easily handle millions, or possibly even billions, of profiles. Then, once the profiles are indexed, an incoming document can be transformed into a "query", which again is represented as either a set of terms or a set of features. The "query" is then run against the index of profiles, retrieving a ranked list of profiles.
- The document is then delivered to only those profiles whose score, with respect to the "query", is above the relevance threshold.

5.2. TEXT MINING

- The purpose of Text Mining is to process unstructured (textual) information, extract meaningful numeric indices from the text, and, thus, make the information contained in the text accessible to the various data mining (statistical and machine learning) algorithms.
- Information can be extracted to derive summaries for the words contained in the documents or to compute summaries for the documents based on the words contained in them. Hence, you can analyze words, clusters of words used in documents, etc., or you could analyze documents and determine similarities between them or how they are related to other variables of interest in the data mining project.
- In the most general terms, text mining will "turn text into numbers" (meaningful indices), which can then be incorporated in other analyses such as predictive data mining projects, the application of unsupervised learning methods (clustering), etc.

Typical Applications for Text Mining

- ✓ Unstructured text is very common, and in fact may represent the majority of information available to a particular research or data mining project.
- **Analyzing open-ended survey responses.:** In survey research (e.g., marketing), it is not uncommon to include various open-ended questions pertaining to the topic under investigation. The idea is to permit respondents to express their "views" or opinions without constraining them to particular dimensions or a particular response format. This may yield insights into customers' views and opinions that might otherwise not be discovered when relying solely on structured questionnaires designed by "experts." For example, you may discover a certain set of words or terms that are commonly used by respondents to describe the pro's and con's of a product or service (under investigation), suggesting common misconceptions or confusion regarding the items in the study.
- **Automatic processing of messages, emails, etc.** Another common application for text mining is to aid in the automatic classification of texts. For example, it is possible to "filter" out automatically most undesirable "junk email" based on certain terms or words that are not likely to appear in legitimate messages, but instead identify undesirable

electronic mail. In this manner, such messages can automatically be discarded. Such automatic systems for classifying electronic messages can also be useful in applications where messages need to be routed (automatically) to the most appropriate department or agency; e.g., email messages with complaints or petitions to a municipal authority are automatically routed to the appropriate departments; at the same time, the emails are screened for inappropriate or obscene messages, which are automatically returned to the sender with a request to remove the offending words or content.

- **Analyzing warranty or insurance claims, diagnostic interviews, etc.** In some business domains, the majority of information is collected in open-ended, textual form. For example, warranty claims or initial medical (patient) interviews can be summarized in brief narratives, or when you take your automobile to a service station for repairs, typically, the attendant will write some notes about the problems that you report and what you believe needs to be fixed. Increasingly, those notes are collected electronically, so those types of narratives are readily available for input into text mining algorithms. This information can then be usefully exploited to, for example, identify common clusters of problems and complaints on certain automobiles, etc. Likewise, in the medical field, open-ended descriptions by patients of their own symptoms might yield useful clues for the actual medical diagnosis.
- **Investigating competitors by crawling their web sites.** Another type of potentially very useful application is to automatically process the contents of Web pages in a particular domain. For example, you could go to a Web page, and begin "crawling" the links you find there to process all Web pages that are referenced. In this manner, you could automatically derive a list of terms and documents available at that site, and hence quickly determine the most important terms and features that are described. It is easy to see how these capabilities could efficiently deliver valuable business intelligence about the activities of competitors.

Approaches to Text Mining:

- Text mining can be summarized as a process of "numericizing" text. At the simplest level, all words found in the input documents will be indexed and counted in order to compute a table of documents and words, i.e., a matrix of frequencies that enumerates the number of times that each word occurs in each document. This basic process can be further refined to exclude certain common words such as "the" and "a" (stop word lists) and to combine different grammatical forms of the same words such as "traveling," "traveled," "travel," etc. (stemming). However, once a table of (unique) words (terms) by documents has been derived, all standard statistical and data mining techniques can be applied to derive dimensions or clusters of words or documents, or to identify "important" words or terms that best predict another outcome variable of interest.
- **Using well-tested methods and understanding the results of text mining.** Once a data matrix has been computed from the input documents and words found in those documents, various well-known analytic techniques can be used for further processing those data including methods for clustering, factoring, or predictive data mining.
- **"Black-box" approaches to text mining and extraction of concepts.** There are text mining applications which offer "black-box" methods to extract "deep meaning" from documents with little human effort (to first read and understand those documents). These text mining applications rely on proprietary algorithms for presumably extracting

"concepts" from text, and may even claim to be able to summarize large numbers of text documents automatically, retaining the core and most important meaning of those documents. While there are numerous algorithmic approaches to extracting "meaning from documents," this type of technology is very much still in its infancy, and the aspiration to provide meaningful automated summaries of large numbers of documents may forever remain elusive. We urge skepticism when using such algorithms because 1) if it is not clear to the user how those algorithms work, it cannot possibly be clear how to interpret the results of those algorithms, and 2) the methods used in those programs are not open to scrutiny, for example by the academic community and peer review and, hence, we simply don't know how well they might perform in different domains. As a final thought on this subject, you may consider this concrete example: Try the various automated translation services available via the Web that can translate entire paragraphs of text from one language into another. Then translate some text, even simple text, from your native language to some other language and back, and review the results. Almost every time, the attempt to translate even short sentences to other languages and back while retaining the original meaning of the sentence produces humorous rather than accurate results. This illustrates the difficulty of automatically interpreting the meaning of text.

- **Text mining as document search.** There is another type of application that is often described and referred to as "text mining" - the automatic search of large numbers of documents based on key words or key phrases. This is the domain of, for example, the popular internet search engines that have been developed over the last decade to provide efficient access to Web pages with certain content. While this is obviously an important type of application with many uses in any organization that needs to search very large document repositories based on varying criteria, it is very different from what has been described here.

Issues and Considerations for "Numericizing" Text

- **Large numbers of small documents vs. small numbers of large documents.** Examples of scenarios using large numbers of small or moderate sized documents were given earlier (e.g., analyzing warranty or insurance claims, diagnostic interviews, etc.). On the other hand, if your intent is to extract "concepts" from only a few documents that are very large (e.g., two lengthy books), then statistical analyses are generally less powerful because the "number of cases" (documents) in this case is very small while the "number of variables" (extracted words) is very large.
- **Excluding certain characters, short words, numbers, etc.** Excluding numbers, certain characters, or sequences of characters, or words that are shorter or longer than a certain number of letters can be done before the indexing of the input documents starts. You may also want to exclude "rare words," defined as those that only occur in a small percentage of the processed documents.
- **Include lists, exclude lists (stop-words).** Specific list of words to be indexed can be defined; this is useful when you want to search explicitly for particular words, and classify the input documents based on the frequencies with which those words occur. Also, "stop-words," i.e., terms that are to be excluded from the indexing can be defined. Typically, a default list of English stop words includes "the", "a", "of", "since," etc, i.e.,

words that are used in the respective language very frequently, but communicate very little unique information about the contents of the document.

- **Synonyms and phrases.** Synonyms, such as "sick" or "ill", or words that are used in particular phrases where they denote unique meaning can be combined for indexing. For example, "Microsoft Windows" might be such a phrase, which is a specific reference to the computer operating system, but has nothing to do with the common use of the term "Windows" as it might, for example, be used in descriptions of home improvement projects.
- **Stemming algorithms.** An important pre-processing step before indexing of input documents begins is the stemming of words. The term "stemming" refers to the reduction of words to their roots so that, for example, different grammatical forms or declinations of verbs are identified and indexed (counted) as the same word. For example, stemming will ensure that both "traveling" and "traveled" will be recognized by the text mining program as the same word.
- **Support for different languages.** Stemming, synonyms, the letters that are permitted in words, etc. are highly language dependent operations. Therefore, support for different languages is important.

Transforming Word Frequencies

- Once the input documents have been indexed and the initial word frequencies (by document) computed, a number of additional transformations can be performed to summarize and aggregate the information that was extracted.
- **Log-frequencies.** First, various transformations of the frequency counts can be performed. The raw word or term frequencies generally reflect on how salient or important a word is in each document. Specifically, words that occur with greater frequency in a document are better descriptors of the contents of that document. However, it is not reasonable to assume that the word counts themselves are proportional to their importance as descriptors of the documents. For example, if a word occurs 1 time in document A, but 3 times in document B, then it is not necessarily reasonable to conclude that this word is 3 times as important a descriptor of document B as compared to document A. Thus, a common transformation of the raw word frequency counts (wf) is to compute:

$$f(wf) = 1 + \log(wf), \text{ for } wf > 0$$

- This transformation will "dampen" the raw frequencies and how they will affect the results of subsequent computations.
- **Binary frequencies.** Likewise, an even simpler transformation can be used that enumerates whether a term is used in a document; i.e.:

$$f(wf) = 1, \text{ for } wf > 0$$

- The resulting documents-by-words matrix will contain only 1s and 0s to indicate the presence or absence of the respective words. Again, this transformation will dampen the effect of the raw frequency counts on subsequent computations and analyses.
- **Inverse document frequencies.** Another issue that you may want to consider more carefully and reflect in the indices used in further analyses are the relative document frequencies (df) of different words. For example, a term such as "guess" may occur frequently in all documents, while another term such as "software" may only occur in a few. The reason is that we might make "guesses" in various contexts, regardless of the

specific topic, while "software" is a more semantically focused term that is only likely to occur in documents that deal with computer software. A common and very useful transformation that reflects both the specificity of words (document frequencies) as well as the overall frequencies of their occurrences (word frequencies) is the so-called inverse document frequency (for the i'th word and j'th document):

$$idf(i,j) = \begin{cases} 0 & \text{if } wf_{i,j} = 0 \\ (1 + \log(wf_{i,j})) \log \frac{N}{df_i} & \text{if } wf_{i,j} \geq 1 \end{cases}$$

- In this formula, *N* is the total number of documents, and *dfi* is the document frequency for the *i*'th word (the number of documents that include this word). Hence, it can be seen that this formula includes both the dampening of the simple word frequencies via the log function , and also includes a weighting factor that evaluates to 0 if the word occurs in all documents *(log(N/N=1)=0)*, and to the maximum value when a word only occurs in a single document *(log(N/1)=log(N))*. It can easily be seen how this transformation will create indices that both reflect the relative frequencies of occurrences of words, as well as their semantic specificities over the documents included in the analysis.

Latent Semantic Indexing via Singular Value Decomposition

- The most basic result of the initial indexing of words found in the input documents is a frequency table with simple counts, i.e., the number of times that different words occur in each input document. Usually, we would transform those raw counts to indices that better reflect the (relative) "importance" of words and/or their semantic specificity in the context of the set of input documents (see the discussion of inverse document frequencies, above).
- A common analytic tool for interpreting the "meaning" or "semantic space" described by the words that were extracted, and hence by the documents that were analyzed, is to create a mapping of the word and documents into a common space, computed from the word frequencies or transformed word frequencies (e.g., inverse document frequencies). In general, here is how it works:
- Suppose you indexed a collection of customer reviews of their new automobiles (e.g., for different makes and models). You may find that every time a review includes the word "gas-mileage," it also includes the term "economy." Further, when reports include the word "reliability" they also include the term "defects" (e.g., make reference to "no defects"). However, there is no consistent pattern regarding the use of the terms "economy" and "reliability," i.e., some documents include either one or both. In other words, these four words "gas-mileage" and "economy," and "reliability" and "defects," describe two independent dimensions - the first having to do with the overall operating cost of the vehicle, the other with the quality and workmanship. The idea of latent semantic indexing is to identify such underlying dimensions (of "meaning"), into which the words and documents can be mapped. As a result, we may identify the underlying (latent) themes described or discussed in the input documents, and also identify the documents that mostly deal with economy, reliability, or both. Hence, we want to map the extracted words or terms and input documents into a common latent semantic space.

- **Singular value decomposition.** The use of singular value decomposition in order to extract a common space for the variables and cases (observations) is used in various statistical techniques, most notably in *Correspondence Analysis*. The technique is also closely related to Principal Components Analysis and *Factor Analysis*. In general, the purpose of this technique is to reduce the overall dimensionality of the input matrix (number of input documents by number of extracted words) to a lower-dimensional space, where each consecutive dimension represents the largest degree of variability (between words and documents) possible. Ideally, you might identify the two or three most salient dimensions, accounting for most of the variability (differences) between the words and documents and, hence, identify the latent semantic space that organizes the words and documents in the analysis. In some way, once such dimensions can be identified, you have extracted the underlying "meaning" of what is contained in the documents.

Incorporating Text Mining Results in Data Mining Projects

- After significant (e.g., frequent) words have been extracted from a set of input documents, and/or after singular value decomposition has been applied to extract salient semantic dimensions, typically the next and most important step is to use the extracted information in a data mining project.
- **Graphics (visual data mining methods).** Depending on the purpose of the analyses, in some instances the extraction of semantic dimensions alone can be a useful outcome if it clarifies the underlying structure of what is contained in the input documents. For example, a study of new car owners' comments about their vehicles may uncover the salient dimensions in the minds of those drivers when they think about or consider their automobile (or how they "feel" about it). For marketing research purposes, that in itself can be a useful and significant result. You can use the graphics (e.g., 2D scatter plots or 3D scatter plots) to help you visualize and identify the semantic space extracted from the input documents.
- **Clustering and factoring.** You can use cluster analysis methods to identify groups of documents (e.g., vehicle owners who described their new cars), to identify groups of similar input texts. This type of analysis also could be extremely useful in the context of market research studies, for example of new car owners. You can also use *Factor Analysis* and Principal Components and Classification Analysis (to factor analyze words or documents).
- **Predictive data mining.** Another possibility is to use the raw or transformed word counts as predictor variables in predictive data mining projects.

5.3. TEXT CLASSIFICATION AND CLUSTERING:

- ✓ In ad *hoc retrieval*, users have transient information needs that they try to address by posing one or more queries to a search engine.
- ✓ However, many users have ongoing information needs. For example, you might need to track developments in *multicore computer chips*. One way of doing this is to issue the query multicore AND computer AND chip against an index of recent newswire articles each morning. In this and the following two chapters we examine the question:

- ✓ How can this repetitive task be automated? To this end, many systems support *standing queries*.
- ✓ A standing query is like any other query except that it query is periodically executed on a collection to which new documents are incrementally added over time.
- ✓ If your standing query is just multicore chip, you will tend to miss many relevant new articles which use other terms such as *multicore processors*. To achieve good recall, standing queries thus have to be refined over time and can gradually become quite complex. In this example, using a Boolean search engine with stemming, you might end up with a query like (multicore OR multi-core) AND (chip OR processor OR microprocessor).
- ✓ To capture the generality and scope of the problem space to which standing queries belong, we now introduce the general notion of a *classification* problem.
- ✓ Given a set of *classes*, we seek to determine which class(es) a given object belongs to. In the example, the standing query serves to divide new newswire articles into the two classes: documents about multicore computer chips and documents not about multicore computer chips. This is referred as ***two-class classification***. Classification using standing queries is also called ***routing* or filtering.**
- ✓ A class need not be as narrowly focused as the standing query *multicore computer chips*. Often, a class is a more general subject area like *China* or *coffee*.
- ✓ Such more general classes are usually referred to as *topics*, and the classification task is then called *text classification, text categorization, topic classification*, or *topic spotting*.
- ✓ An example for *China* appears in Figure. Standing queries and topics differ in their degree of specificity, but the methods for solving routing, filtering, and text classification are essentially the same.
- ✓ The notion of classification is very general and has many applications within and beyond information retrieval (IR). For instance, in computer vision, a classifier may be used to divide images into classes such as *landscape*, *portrait*, and *neither*. We focus here on examples from information retrieval such as:
- ✓ Several of the preprocessing steps necessary for indexing: detecting a document's encoding (ASCII, Unicode UTF-8 etc;); word segmentation (Is the white space between two letters a word boundary or not?); true casing; and identifying the language of a document .
- ✓ The automatic detection of spam pages (which then are not included in the search engine index).
- ✓ The automatic detection of sexually explicit content (which is included in search results only if the user turns an option such as Safe Search off).

- ***Sentiment detection* o**r the automatic classification of a movie or product review as positive or negative. An example application is a user searching for negative reviews before buying a camera to make sure it has no undesirable features or quality problems.
- **Personal *email sorting*.** A user may have folders like *talk announcements*, *electronic bills*, *email from family and friends*, and so on, and may want a classifier to classify each incoming email and automatically move it to the appropriate folder. It is easier to find messages in sorted folders than in a very large inbox. The most common case of this application is a spam folder that holds all suspected spam messages.

- **Topic-specific or *vertical* search.** *Vertical search engines* restrict searches to a particular topic. For example, the query computer science on a vertical search engine for the topic *China* will return a list of Chinese computer science departments with higher precision and recall than the query computer science China on a general purpose search engine. This is because the vertical search engine does not include web pages in its index that contain the term china in a different sense (e.g., referring to a hard white ceramic), but does include relevant pages even if they do not explicitly mention the term China.
- Finally, the ranking function in ad hoc information retrieval can also be based on a document classifier.

- ✓ This list shows the general importance of classification in IR. Most retrieval systems today contain multiple components that use some form of classifier.
- ✓ The classification task we will use as an example in this book is text classification. A computer is not essential for classification. Many classification tasks have traditionally been solved manually. Books in a library are assigned Library of Congress categories by a librarian. But manual classification is expensive to scale.
- ✓ The *multicore computer chips* example illustrates one alternative approach: classification by the use of standing queries – which can be thought of as *rules* – most commonly written by hand. As in our exam ple (multicore OR multi-core) AND (chip OR processor OR microprocessor), rules are sometimes equivalent to Boolean expressions.
- ✓ A rule captures a certain combination of keywords that indicates a class.
- ✓ Hand-coded rules have good scaling properties, but creating and maintaining them over time is labor intensive. A technically skilled person (e.g., a domain expert who is good at writing regular expressions) can create rule sets that will rival or exceed the accuracy of the automatically generated classifiers we will discuss shortly; however, it can be hard to find someone with this specialized skill.
- ✓ Apart from manual classification and hand-crafted rules, there is a third approach to text classification, namely, machine learning-based text classification.
- ✓ In machine learning, the set of rules or, more generally, the decision criterion of the text classifier, is learned automatically from training data. This approach is also called ***statistical text classification*** if the learning method is statistical.
- ✓ In statistical text classification, we require a number of good example documents (or training documents) for each class. The need for manual classification is not eliminated because the training documents come from a person who has labeled them – where *labeling* refers to the process of annotating each document with its class. But labeling is arguably an easier task than writing rules. Almost anybody can look at a document and decide whether or not it is related to China.
- ✓ Sometimes such labeling is already implicitly part of an existing workflow. For instance, you may go through the news articles returned by a standing query each morning and give relevance feedback by moving the relevant articles to a special folder like *multicore-processors*.

The text classification problem:

- ✓ In text classification, we are given a description $d \in X$ of a document, where X is the *document space*; and a fixed set of *classes* $C = \{c1, c2, \ldots, c J\}$. Classes are also called *categories* or *labels*. Typically, the document space X is some type of high-dimensional space, and the classes are human defined for the needs of an application, as in the examples *China* and *documents that talk about multicore computer chips* above. We are given a *training set* D of labeled documents $\langle d, c \rangle$,where $\langle d, c \rangle \in X \times C$. For example:

  $\langle d, c \rangle$ =<Beijing joins the World Trade Organization, *China*>
- ✓ For the one-sentence document *Beijing joins the World Trade Organization* and the class (or label) *China*. Using a *learning method* or *learning algorithm*, we then wish to learn a classifier or *classification function* γ that maps documents to classes:

  $$\gamma : X \rightarrow C$$
- ✓ This type of learning is called *supervised learning* because a supervisor (the human who defines the classes and labels training documents) serves as a teacher directing the learning process. We denote the supervised learning method by Γ and write $\Gamma(D) = \gamma$. The learning method Γ takes the training set D as input and returns the learned classification function γ.
- ✓ Most names for learning methods Γ are also used for classifiers γ. We talk about the Naive Bayes (NB) *learning method* Γ when we say that "Naive Bayes is robust," meaning that it can be applied to many different learning problems and is unlikely to produce classifiers that fail catastrophically. But when we say that "Naive Bayes had an error rate of 20%," we are describing an experiment in which a particular NB *classifier* γ (which was produced by the NB learning method) had a 20% error rate in an application.
- ✓ Figure shows an example of text classification from the Reuters-RCV1 collection. There are six classes (*UK*, *China*, . . . , *sports*), each with three training documents. We show a few mnemonic words for each document's content. The training set provides some typical examples for each class, so that we can learn the classification function γ.
- ✓ Once we have learned γ, we can apply it to the *test set* (or *test data*), for example, the new document *first private Chinese airline* whose class is unknown.
- ✓ In Figure, the classification function assigns the new document to class $\gamma(d)$ = *China*, which is the correct assignment.
- ✓ The classes in text classification often have some interesting structure such as the hierarchy in Figure . There are two instances each of region categories, industry categories, and subject area categories. A hierarchy can be an important aid in solving a classification problem.

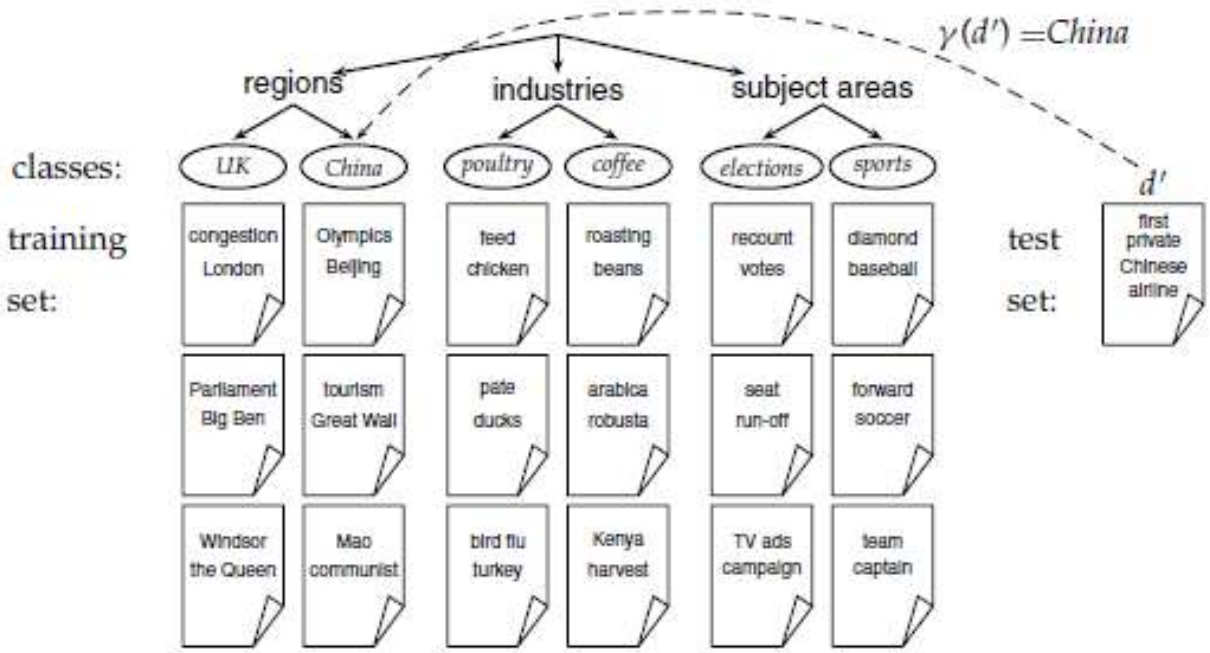


Figure . Classes, training set, and test set in text classification.

- ✓ A document is a member of exactly one class. This is not the most appropriate model for the hierarchy in Figure. For instance, a document about the 2008 Olympics should be a member of two classes: the *China* class and the *sports* class. This type of classification problem is referred to as an *any-of* problem. For the time being, we only consider *one-of* problems where a document is a member of exactly one class.
- ✓ Our goal in text classification is high accuracy on test data or *new data* – for example, the newswire articles that we will encounter tomorrow morning in the multicore chip example. It is easy to achieve high accuracy on the training set (e.g., we can simply memorize the labels). But high accuracy on the training set in general does not mean that the classifier will work well on new data in an application. When we use the training set to learn a classifier for test data, we make the assumption that training data and test data are similar or from *the same distribution*.

Difference between Classification and clustering:

- *Classification*, also referred to as categorization, is the task of automatically applying labels to data, such as emails, web pages, or images. People classify items throughout their daily lives. It would be infeasible, however, to manually label every page on the Web according to some criteria, such as "spam" or "not spam." Therefore, there is a need for automatic classification and categorization techniques.
- *Clustering* can be broadly defined as the task of grouping related items together. In classification, each item is assigned a label, such as "spam" or "not spam." In clustering, however, each item is assigned to one or more clusters, where the cluster does not necessarily correspond to a meaningful concept, such as "spam" or "not spam". items are

grouped together according to their similarity. Therefore, rather than mapping items onto a predefined set of labels, clustering allows the data to "speak for itself " by uncovering the implicit structure that relates the items.

Difference between supervised learning and unsupervised learning:

- In machine learning, the learning algorithms are typically characterized as supervised or unsupervised. In *supervised learning*, a model is learned using a set of fully labeled items, which is often called the *training set*.
- Once a model is learned, it can be applied to a set of unlabeled items, called the *test set*, in order to automatically apply labels. Classification is often cast as a supervised learning problem. For example, given a set of emails that have been labeled as "spam" or "not spam" (the training set), a classification model can be learned. The model then can be applied to incoming emails in order to classify them as "spam" or "not spam".
- *Unsupervised learning* algorithms learn entirely based on unlabeled data. Unsupervised learning tasks are often posed differently than supervised learning tasks, since the input data is not mapped to a predefined set of labels. Clustering is the most common example of unsupervised learning. Clustering algorithms take a set of unlabeled data as input and then group the items using some notion of similarity.

5.4. CATEGORIZATION ALGORITHMS

- Applying labels to observations is a very natural task, and something that most of us do, often without much thought, in our everyday lives. For example, consider a trip to the local grocery store. We often implicitly assign labels such as "ripe" or "not ripe," "healthy" or "not healthy," and "cheap" or "expensive" to the groceries that we see. These are examples of binary labels, since there are only two options for each.
- It is also possible to apply multivalued labels to foods, such as "starch," "meat," "vegetable," or "fruit." Another possible labeling scheme would arrange categories into a hierarchy, in which the "vegetable" category would be split by color into subcategories, such as "green," "red," and "yellow." Under this scheme, foods would be labeled according to their position within the hierarchy. These different labeling or categorization schemes, which include binary, multivalued, and hierarchical, are called *ontologies*.
- It is important to choose an ontology that is appropriate for the underlying task. For example, for detecting whether or not an email is spam, it is perfectly reasonable to choose a label set that consists of "spam" and "not spam". However, if one were to design a classifier to automatically detect what language a web page is written in, then the set of all possible languages would be a more reasonable ontology.
- The correct choice of ontology is dictated by the problem, but in cases when it is not, it is important to choose a set of labels that is expressive enough to be useful for the underlying task. However, since classification is a supervised learning task, it is important not to construct an overly complex ontology, since most learning algorithms will fail (i.e., not generalize well to unseen data) when there is little or no data associated with one or more of the labels.
- In the web page language classifier example, if we had only one example page for each of the Asian languages, then, rather than having separate labels for each of the languages,

such as “Chinese”, “Korean”, etc., it would be better to combine all of the languages into a single label called “Asian languages”. The classifier will then be more likely to classify things as “Asian languages” correctly, since it has more training examples.

- In order to understand how machine learning algorithms work, we must first take a look at how people classify items. Returning to the grocery store example, consider how we would classify a food as “healthy” or “not healthy.”
- In order to make this classification, we would probably look at the amount of saturated fat, cholesterol, sugar, and sodium in the food. If these values, either separately or in combination, are above some threshold, then we would label the food “healthy” or “unhealthy.” To summarize, as humans we classify items by first identifying a number of important *features* that will help us distinguish between the possible labels. We then *extract* these features from each item. We then *combine evidence* from the extracted features in some way. Finally, we *classify* the item using some decision mechanism based on the combined evidence.
- In our example, the features are things such as the amount of saturated fat and the amount of cholesterol. The features are extracted by reading the nutritional information printed on the packaging or by performing laboratory tests. There are various ways to combine the evidence in order to quantify the “healthiness” (denoted *H*) of the food, but one simple way is to weight the importance of each feature and then add the weighted feature values together, such as:

$$H(food) \approx w_{fat} fat(food) + w_{chol} chol(food) + w_{sugar} sugar(food) + w_{sodium} sodium(food)$$

- where w_{fat}, w_{chol}, etc., are the weights associated with each feature. Of course, in this case, it is likely that each of the weights would be negative.
- Once we have a healthiness score, *H*, for a given food, we must apply some decision mechanism in order to apply a “healthy” or “not healthy” label to the food. Again, there are various ways of doing this, but one of the most simple is to apply a simple threshold rule that says “a food is healthy if $H(food) \geq t$” for some threshold value *t*.
- Although this is an idealized model of how people classify items, it provides valuable insights into how a computer can be used to automatically classify items. Indeed, the two classification algorithms that we will now describe follow the same steps as we outlined earlier. The only difference between the two algorithms is in the details of how each step is actually implemented.

5.4.1. Naïve Bayes:

- One of the most straightforward yet effective classification techniques is called *Naïve Bayes*. In general, classification tasks can involve more than two labels or classes. In that situation, *Bayes' Rule*, which is the basis of a Bayes classifier, states that:

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$
$$= \frac{P(D|C)P(C)}{\sum_{c \in \mathcal{C}} P(D|C=c)P(C=c)}$$

- Where C and D are *random variables*. Random variables are commonly used when modeling uncertainty. Such variables do not have a fixed (deterministic) value. Instead, the value of the variable is random. Every random variable has a set of possible outcomes associated with it, as well as a probability distribution over the outcomes. As an example, the outcome of a coin toss can be modeled as a random variable X. The possible outcomes of the random variable are "heads" (h) and "tails" (t). Given a fair coin, the probability associated with both the heads outcome and the tails outcome is 0.5. Therefore, $P(X = h) = P(X = t) = 0.5$.
- Consider another example, where you have the algebraic expression $Y = 10 + 2X$. If X was a deterministic variable, then Y would be deterministic as well. That is, for a fixed X, Y would always evaluate to the same value. However, if X is a random variable, then Y is also a random variable. Suppose that X had possible outcomes –1 (with probability 0.1), 0 (with probability 0.25), and 1 (with probability 0.65). The possible outcomes for Y would then be 8, 10, and 12, with $P(Y = 8) = 0.1$, $P(Y = 10) = 0.25$, and $P(Y = 12) = 0.65$.
- We denote random variables with capital letters (e.g., C, D) and outcomes of random variables as lowercase letters (e.g., c, d). Furthermore, we denote the entire set of outcomes with caligraphic letters (e.g., $\mathcal{C}$, $\mathcal{D}$). Finally, for notational convenience, instead of writing $P(X = x)$, we write $P(x)$. Similarly for conditional probabilities, rather than writing $P(X = x|Y = y)$, we write $P(x|y)$.
- Bayes' Rule is important because it allows us to write a conditional probability (such as $P(C|D)$) in terms of the "reverse" conditional ($P(D|C)$). This is a very powerful theorem, because it is often easy to estimate or compute the conditional probability in one direction but not the other. For example, consider spam classification, where D represents a document's text and C represents the class label (e.g., "spam" or "not spam"). It is not immediately clear how to write a program that detects whether a document is spam; that program is represented by $P(C|D)$.
- However, it is easy to find examples of documents that are and are not spam. It is possible to come up with estimates for $P(D|C)$ given examples or training data. The magic of Bayes' Rule is that it tells us how to get what we want ($P(C|D)$), but may not immediately know how to estimate, from something we do know how to estimate ($P(D|C)$).
- It is straightforward to use this rule to classify items if we let C be the random variable associated with observing a class label and let D be the random variable associated with observing a document, as in our spam example. Given a document d (an outcome of random variable D) and a set of classes $C = c_1, \ldots, c_N$ (outcomes of the random variable C), we can use Bayes' Rule to compute $P(c1|d), \ldots, P(c_N|d)$, which computes the likelihood of observing class label c_i given that document d was observed. Document d can then be labeled with the class with the highest probability of being observed given the document. That is, Naïve Bayes classifies a document d as follows:

$$\text{Class}(d) = \arg\max_{c \in \mathcal{C}} P(c|d)$$
$$= \arg\max_{c \in \mathcal{C}} \frac{P(d|c)P(c)}{\sum_{c \in C} P(d|c)P(c)}$$

- Where $\arg\max_{c \in \mathcal{C}} P(c|d)$ means "return the class *c*, out of the set of all possible classes *C*, that maximizes *P*(*c*|*d*)." This is a mathematical way of saying that we are trying to find the most likely class *c* given the document *d*.
- Instead of computing *P*(*c*|*d*) directly, we can compute *P*(*d*|*c*) and *P*(*c*) instead and then apply Bayes' Rule to obtain *P*(*c*|*d*). As we explained before, one reason for using Bayes' Rule is when it is easier to estimate the probabilities of one conditional, but not the other. We now explain how these values are typically estimated in practice.
- We first describe how to estimate the class prior, *P*(*c*). The estimation is straightforward. It is estimated according to:

$$P(c) = \frac{N_c}{N}$$

- Where N_c is the number of training instances that have label *c*, and *N* is the total number of training instances. Therefore, *P*(*c*) is simply the proportion of training instances that have label *c*.
- Estimating *P*(*d*|*c*) is a little more complicated because the same "counting" estimate that we were able to use for estimating *P*(*c*) would not work. In order to make the estimation feasible, we must impose the simplifying assumption that *d* can be represented as $d = w_1, \ldots, w_n$ and that w_i is independent of w_j for every i ≠ j.
- Simply stated, this says that document *d* can be factored into a set of elements (terms) and that the elements (terms) are independent of each other. This assumption is the reason for calling the classifier *naïve*, because it requires documents to be represented in an overly simplified way.
- In reality, terms are not independent of each other. However properly modeling term dependencies is possible, but typically more difficult. Despite the independence assumption, the Naïve Bayes classifier has been shown to be robust and highly effective for various classification tasks.
- This naïve independence assumption allows us to invoke a classic result from probability that states that the joint probability of a set of (conditionally) independent random variables can be written as the product of the individual conditional probabilities. That means that *P*(*d*|*c*) can be written as:

$$P(d|c) = \prod_{i=1}^{n} P(w_i|c)$$

- Therefore, we must estimate *P*(*w*|*c*) for every possible term *w* in the vocabulary *V* and class *c* in the ontology *C*. It turns out that this is a much easier task than estimating *P*(*d*|*c*)

since there is a finite number of terms in the vocabulary and a finite number of classes, but an infinite number of possible documents. The independence assumption allows us to write the probability $P(c|d)$ as:

$$P(c|d) = \frac{P(d|c)P(c)}{\sum_{c \in \mathcal{C}} P(d|c)P(c)} = \frac{\prod_{i=1}^{\mathcal{V}} P(w_i|c)P(c)}{\sum_{c \in \mathcal{C}} \prod_{i=1}^{\mathcal{V}} P(w_i|c)P(c)}$$

- The only thing left to describe is how to estimate $P(w|c)$. Before we can estimate the probability, we must first decide on what the probability actually means. For example, $P(w|c)$ could be interpreted as "the probability that term w is related to class c," "the probability that w has nothing to do with class c," or any number of other things. In order to make the meaning concrete, we must explicitly define the event space that the probability is defined over.
- An ***event space*** is the set of possible events (or outcomes) from some process. A probability is assigned to each event in the event space, and the sum of the probabilities over all of the events in the event space must equal one.
- The probability estimates and the resulting classification will vary depending on the choice of event space. We will now briefly describe two of the more popular event spaces and show how $P(w|c)$ is estimated in each.

Multiple-Bernoulli model

- Given a class c, we define a *binary* random variable w_i for every term in the vocabulary. The outcome for the binary event is either 0 or 1. The probability $P(wi = 1|c)$ can then be interpreted as "the probability that term w_i is generated by class c." Conversely, $P(wi = 0|c)$ can be interpreted as "the probability that term w_i is not generated by class c." This is exactly the event space used by the binary independence model and is known as the ***multiple-Bernoulli* event space**.
- Under this event space, for each term in some class c, we estimate the probability that the term is generated by the class. For example, in a spam classifier, $P(cheap = 1|spam)$ is likely to have a high probability, whereas $P(dinner = 1|spam)$ is going to have a much lower probability.

document *id*	cheap	buy	banking	dinner	the	*class*
1	0	0	0	0	1	not spam
2	1	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	1	0	1	0	1	spam
5	1	1	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	0	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	1	0	1	1	not spam

Fig. Illustration of how documents are represented in the multiple-Bernoulli event space. In this example, there are 10 documents (each with a unique *id*), two classes (spam and not spam), and a vocabulary that consists of the terms "cheap", "buy", "banking", "dinner", and "the".

- Figure shows how a set of training documents can be represented in this event space. In the example, there are 10 documents, two classes (spam and not spam), and a vocabulary that consists of the terms "cheap", "buy", "banking", "dinner", and "the". In this example, P(spam) = 3/ 10 and P(not spam) = 7 /10 . Next, we must estimate $P(w|c)$ for every pair of terms and classes. The most straightforward way is to estimate the probabilities using what is called the maximum likelihood estimate, which is:

$$P(w|c) = \frac{df_{w,c}}{N_c}$$

- where $df_{w;c}$ is the number of training documents with class label c in which term w occurs, and Nc is the total number of training documents with class label c. As we see, the maximum likelihood estimate is nothing more than the proportion of documents in class c that contain term w. Using the maximum likelihood estimate, we can easily compute P(*the*|spam) = 1, P(*the*|not spam) = 1, P(*dinner*|spam) = 0, P(*dinner*|not spam) = 1/ 7 , and so on.
- Using the multiple-Bernoulli model, the document likelihood, $P(d|c)$, can be written as:

$$P(d|c) = \prod_{w \in \mathcal{V}} P(w|c)^{\delta(w,d)} \left(1 - P(w|c)\right)^{1-\delta(w,d)}$$

- Where $\delta(w,D)$ is 1 if and only if term w occurs in document d.
- In practice, it is not possible to use the maximum likelihood estimate because of the *zero probability problem*. In order to illustrate the zero probability problem, let us return to the spam classification example from Figure. Suppose that we receive a spam email that happens to contain the term "dinner". No matter what other terms the email does or does not contain, the probability$P(d|c)$ will always be zero because P(*dinner*|spam) = 0 and the

term occurs in the document (i.e., δ *dinner;d* = 1). Therefore, any document that contains the term "dinner" will automatically have zero probability of being spam.

- This problem is more general, since a zero probability will result whenever a document contains a term that never occurs in one or more classes. The problem here is that the maximum likelihood estimate is based on counting occurrences in the training set. However, the training set is finite, so not every possible event is observed. This is known as data sparseness.
- Sparseness is often a problem with small training sets, but it can also happen with relatively large data sets. Therefore, we must alter the estimates in such a way that all terms, including those that have not been observed for a given class, are given some probability mass. That is, we must ensure that $P(w|c)$ is nonzero for all terms in V. By doing so, we will avoid all of the problems associated with the zero probability problem.
- **Smoothing** is a useful technique for overcoming the zero probability problem. One popular smoothing technique is often called *Bayesian smoothing*, which assumes some prior probability over models and uses a *maximum a posteriori* estimate. The resulting smoothed estimate for the multiple- Bernoulli model has the form:

$$P(w|c) = \frac{df_{w,c} + \alpha_w}{N_c + \alpha_w + \beta_w}$$

- Where α_w and β_w are parameters that depend on w. Different settings of these parameters result in different estimates. One popular choice is to set $\alpha_w = 1$ and $\beta_w = 0$ for all w, which results in the following estimate:

$$P(w|c) = \frac{df_{w,c} + 1}{N_c + 1}$$

- Another choice is to set $\alpha_w = \mu\frac{N_w}{N}$ and $\beta_w = \mu(1 - \frac{N_w}{N})$ for all w, whereNw is the total number of training documents in which term w occurs, and μ is a single tunable parameter. This results in the following estimate:

$$P(w|c) = \frac{df_{w,c} + \mu\frac{N_w}{N}}{N_c + \mu}$$

- This event space only captures whether or not the term is generated; it fails to capture *how many* times the term occurs, which can be an important piece of information.

Multinomial model

- The binary event space of the multiple-Bernoulli model is overly simplistic, as it does not model the number of times that a term occurs in a document. Term frequency has been shown to be an important feature for retrieval and classification, especially when used on long documents. When documents are very short, it is unlikely that many terms will occur more than one time, and therefore the multiple-Bernoulli model will be an accurate model. However, more often than not, real collections contain documents that are both short and long, and therefore it is important to take term frequency and, subsequently, document length into account.

- The *multinomial* event space is very similar to the multiple-Bernoulli event space, except rather than assuming that term occurrences are binary ("term occurs" or "term does not occur"), it assumes that terms occur zero or more times ("term occurs zero times", "term occurs one time", etc.).

document *id*	cheap	buy	banking	dinner	the	*class*
1	0	0	0	0	2	not spam
2	3	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	2	0	3	0	2	spam
5	5	2	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	0	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	1	0	1	2	not spam

Fig. Illustration of how documents are represented in the multinomial event space.

- In this example, there are 10 documents (each with a unique *id*), two classes (spam and not spam), and a vocabulary that consists of the terms "cheap", "buy", "banking", "dinner", and "the".
- Figure shows how the documents from our spam classification example are represented in the multinomial event space. The only difference between this representation and the multiple-Bernoulli representation is that the events are no longer binary. The maximum likelihood estimate for the multinomial model is very similar to the multiple-Bernoulli model. It is computed as:

$$P(w|c) = \frac{tf_{w,c}}{|c|}$$

- Where $tf_{w;c}$ is the number of times that term *w* occurs in class *c* in the training set,and $|c|$ is the total number of terms that occur in training documents with class label *c*. In the spam classification example, $P(the|\text{spam}) = \frac{4}{20}$, $P(the|\text{not spam}) = \frac{9}{15}$, $P(dinner|\text{spam}) = 0$, and $P(dinner|\text{not spam}) = \frac{1}{15}$.
- Since terms are now distributed according to a multinomial distribution, the likelihood of a document *d* given a class *c* is computed according to:

$$P(d|c) = P(|d|)\left(tf_{w_1,d}, tf_{w_2,d}, \ldots, tf_{w_{\mathcal{V}},d}\right)! \prod_{w \in \mathcal{V}} P(w|c)^{tf_{w,d}}$$

$$\propto \prod_{w \in \mathcal{V}} P(w|c)^{tf_{w,d}}$$

- where $tf_{w;d}$ is the number of times that term w occurs in document d, $|d|$ is the total number of terms that occur in d, $P(|d|)$ is the probability of generating a document of length $|d|$, and $\left(tf_{w_1,d}, tf_{w_2,d}, \ldots, tf_{w_{\mathcal{V}},d}\right)!$ is the multinomial coefficient. Notice that $P(|d|)$ and the multinomial coefficient are document dependent and, for the purposes of classification, can be ignored.
- The Bayesian smoothed estimates of the term likelihoods are computed according to:

$$P(w|c) = \frac{tf_{w,c} + \alpha_w}{|c| + \sum_{w \in \mathcal{V}} \alpha_w}$$

- where α_w is a parameter that depends on w. As with the multiple-Bernoulli model, different settings of the smoothing parameters result in different types of estimates. Setting $\alpha w = 1$ for all w is one possible option. This results in the following estimate:

$$P(w|c) = \frac{tf_{w,c} + 1}{|c| + |\mathcal{V}|}$$

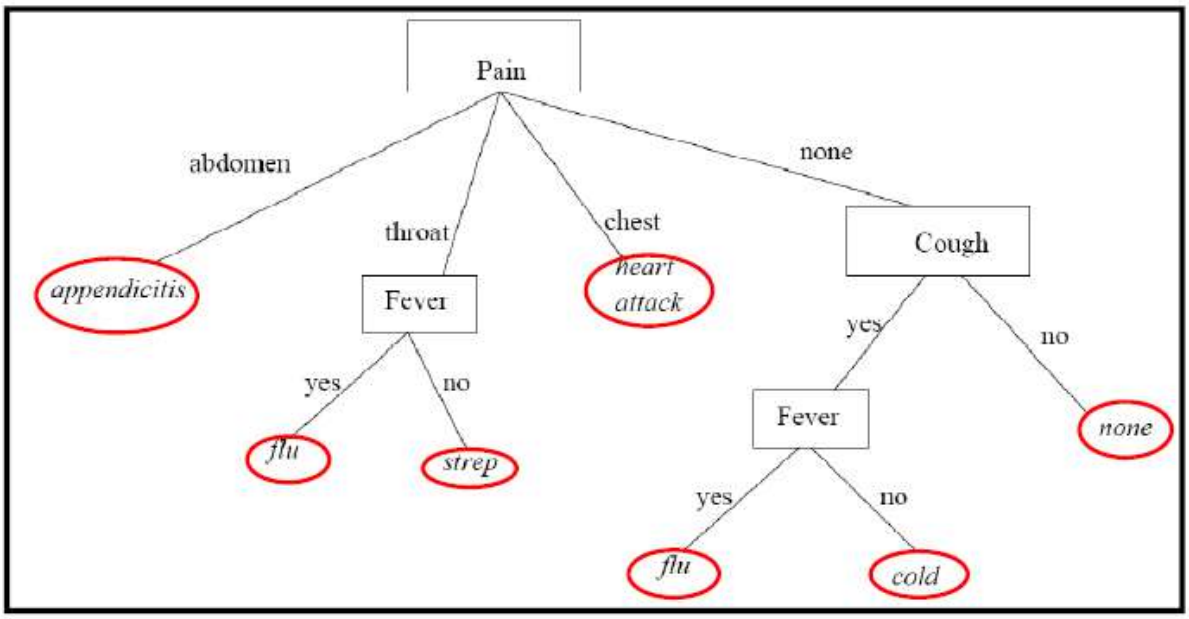
- Another popular choice is to set $\alpha_w = \mu \frac{cf_w}{|C|}$, where cfw is the total number of times that term w occurs in any training document, $|C|$ is the total number of terms in all training documents, and μ, as before, is a tunable parameter. Under this setting, we obtain the following estimate:

$$P(w|c) = \frac{tf_{w,c} + \mu \frac{cf_w}{|C|}}{|c| + \mu}$$

- This estimate may look familiar, as it is exactly the Dirichlet smoothed language modeling estimate.
- In practice, the multinomial model has been shown to consistently outperform the multiple-Bernoulli model. Implementing a classifier based on either of these models is straightforward. Training consists of computing simple term occurrence statistics.
- In most cases, these statistics can be stored in memory, which means that classification can be done efficiently. The simplicity of the model, combined with good accuracy, makes the Naïve Bayes classifier a popular and attractive choice as a general-purpose classification algorithm.

5.4.2. DECISION TREE CLASSIFICATION

Example



- Decision Tree learning is one of the most widely used and practical methods for inductive inference over supervised data.
- A decision tree represents a procedure for classifying categorical data based on their attributes.
- It is also efficient for processing large amount of data, so is often used in data mining application.
- The construction of decision tree does not require any domain knowledge or parameter setting, and therefore appropriate for exploratory knowledge discovery.
- Their representation of acquired knowledge in tree form is intuitive and easy to assimilate by humans.

Decision Tree Algorithm:

- Decide which attribute (splitting-point) to test at node ***N*** by determining the "best" way to separate or partition the tuples in ***D*** into individual classes
- The splitting criteria is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible.

✓ A partition is pure if all of the tuples in it belong to the same class

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1) create a node N;
(2) if tuples in D are all of the same class, C then
(3)     return N as a leaf node labeled with the class C;
(4) if attribute_list is empty then
(5)     return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7) label node N with splitting_criterion;
(8) if splitting_attribute is discrete-valued and
      multiway splits allowed then // not restricted to binary trees
(9)   attribute_list ← attribute_list – splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)   let Dj be the set of data tuples in D satisfying the outcome j; // a partition
(12)   if Dj is empty then
(13)       attach a leaf labeled with the majority class in D to node N;
(14)   else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

5.5. CLUSTERING ALGORITHMS:

- Clustering algorithms provide a different approach to organizing data. Unlike the classification algorithms covered in this chapter, clustering algorithms are based on *unsupervised* learning, which means that they do not require any training data.
- Clustering algorithms take a set of unlabeled instances and group (cluster) them together. One problem with clustering is that it is often an ill-defined problem. Classification has very clear objectives. However, the notion of a good clustering is often defined very subjectively.
- In order to gain more perspective on the issues involved with clustering, let us examine how we, as humans, cluster items. Suppose, once again, that you are at a grocery store and are asked to cluster all of the fresh produce (fruits and vegetables).
- How would you proceed? Before you began, you would have to decide what criteria you would use for clustering. For example, you could group the items by their color, their shape, their vitamin C content, their price, or some meaningful combination of these factors. As with classification, the clustering criteria largely depend on *how the items are represented*. Input instances are assumed to be a feature vector that represents some object, such as a document (or a fruit). If you are interested in clustering according to

some property, it is important to make sure that property is represented in the feature vector.

- After the clustering criteria have been determined, you would have to determine how you would assign items to clusters. Suppose that you decided to cluster the produce according to color and you have created a red cluster (red grapes, red apples) and a yellow cluster (bananas, butternut squash). What do you do if you come across an orange? Do you create a new orange cluster, or do you assign it to the red or yellow cluster? These are important questions that clustering algorithms must address as well. These questions come in the form of *how many clusters to use* and *how to assign items to clusters*.
- Finally, after you have assigned all of the produce to clusters, how do you quantify how well you did? That is, you must *evaluate* the clustering. This is often very difficult, although there have been several automatic techniques proposed.
- In this example, we have described clusters as being defined by some fixed set of properties, such as the "red" cluster. This is, in fact, a very specific form of cluster, called *monothetic*.
- Most clustering algorithms instead produce *polythetic* clusters, where members of a cluster share many properties, but there is no single defining property. In other words, membership in a cluster is typically based on the *similarity* of the feature vectors that represent the objects. This means that a crucial part of defining the clustering algorithm is specifying the similarity measure that is used.
- The classification and clustering literature often refers to a *distance measure*, rather than a similarity measure. Any similarity measure, which typically has a value S from 0 to 1, can be converted into a distance measure by using $1 - S$.

5.5.1. HIERARCHICAL CLUSTERING (DIVISIVE CLUSTERING AND AGGLOMERATIVE CLUSTEING):

- *Hierarchical clustering* is a clustering methodology that builds clusters in a hierarchical fashion. This methodology gives rise to a number of different clustering algorithms. These algorithms are often "clustered" into two groups, depending on how the algorithm proceeds.
- ***Divisive clustering*** algorithms begin with a single cluster that consists of all of the instances. During each iteration it chooses an existing cluster and divides it into two (or possibly more) clusters. This process is repeated until there are a total of K clusters. The output of the algorithm largely depends on how clusters are chosen and split. Divisive clustering is a *top-down* approach.
- The other general type of hierarchical clustering algorithm is called ***agglomerative clustering***, which is a *bottom-up* approach. An agglomerative algorithm starts with each input as a separate cluster. That is, it begins with N clusters, each of which contains a single input. The algorithm then proceeds by joining two (or possibly more) existing clusters to form a new cluster. Therefore, the number of clusters decreases after each iteration. The algorithm terminates when there are K clusters.
- As with divisive clustering, the output of the algorithm is largely dependent on how clusters are chosen and joined.

- The hierarchy generated by an agglomerative or divisive clustering algorithm can be conveniently visualized using a *dendrogram*. A dendrogram graphically represents how a hierarchical clustering algorithm progresses.

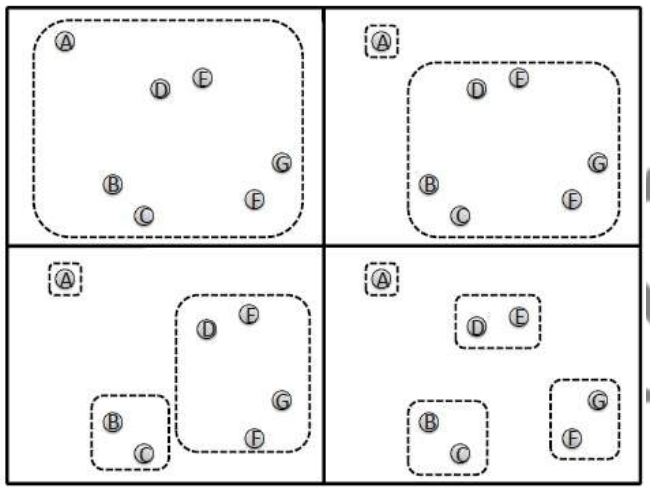


Fig. Example of divisive clustering with $K = 4$. The clustering proceeds from left to right and top to bottom, resulting in four clusters.

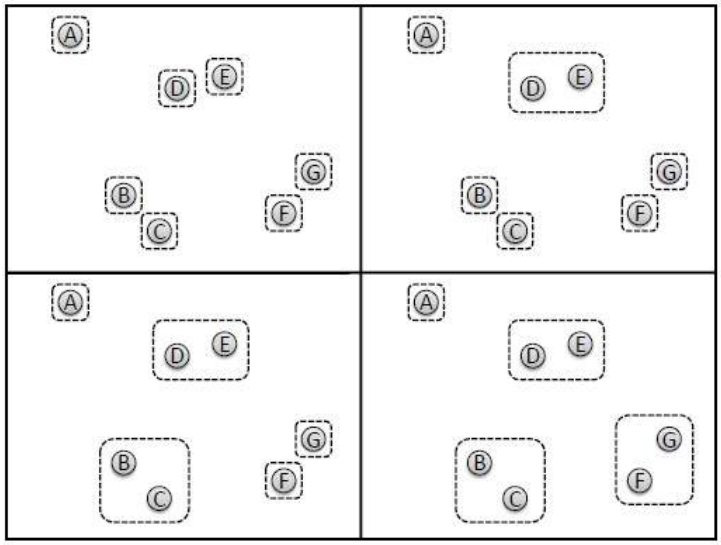


Fig. Example of agglomerative clustering with $K = 4$. The clustering proceeds from left to right and top to bottom, resulting in four clusters.

- Figure shows the dendrogram that corresponds to generating the entire agglomerative clustering hierarchy. In the dendrogram, points D and E are first combined to form a new cluster called H. Then, B and C are combined to form cluster I. This process is continued until a single cluster M is created, which consists of A, B, C, D, E, and F.

- In a dendrogram, the height at which instances combine is significant and represents the similarity (or distance) value at which the combination occurs. For example, the dendrogram shows that D and E are the most similar pair.
- Algorithm 1 is a simple implementation of hierarchical agglomerative clustering. The algorithm takes N vectors $X_1, \ldots, X_N$, representing the instances, and the desired number of clusters$K$ as input. The array (vector) A is the assignment vector. It is used to keep track of which cluster each input is associated with.
- If $A[i] = j$, then it means that input X_i is in cluster j. The algorithm considers joining every pair of clusters. For each pair of clusters (C_i,C_j), a cost $C(C_i,C_j)$ is computed. The cost is some measure of how expensive it would be to merge clusters C_i and C_j . We will return to how the cost is computed shortly. After all pair wise costs have been computed, the pair of clusters with the lowest cost are then merged. The algorithm proceeds until there are K clusters.

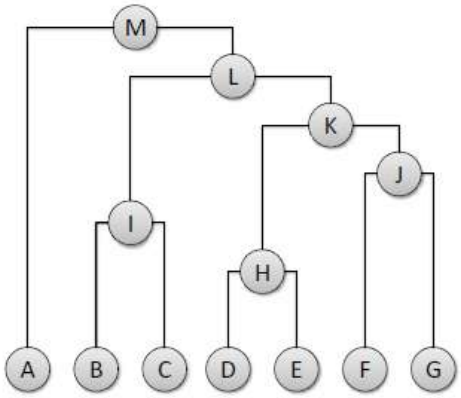


Fig. Dendrogram that illustrates the agglomerative clustering of the points.

Algorithm 1 Agglomerative Clustering

```
1: procedure AGGLOMERATIVECLUSTER(X_1, ..., X_N, K)
2:     A[1], ..., A[N] ← 1, ..., N
3:     ids ← {1, ..., N}
4:     for c = N to K do
5:         bestcost ← ∞
6:         bestclusterA ← undefined
7:         bestclusterB ← undefined
8:         for i ∈ ids do
9:             for j ∈ ids − {i} do
10:                c_{i,j} ← COST(C_i, C_j)
11:                if c_{i,j} < bestcost then
12:                    bestcost ← c_{i,j}
13:                    bestclusterA ← i
14:                    bestclusterB ← j
15:                end if
16:            end for
17:        end for
18:        ids ← ids − {bestClusterA}
19:        for i = 1 to N do
20:            if A[i] is equal to bestClusterA then
21:                A[i] ← bestClusterB
22:            end if
23:        end for
24:    end for
25: end procedure
```

- As shown by Algorithm 1, agglomerative clustering largely depends on the cost function. There are many different ways to define the cost function, each of which results in the final clusters having different characteristics.
- *Single linkage* measures the cost between clusters C_i and C_j by computing the distance between every instance in cluster C_i and every one in C_j . The cost is then the *minimum* of these distances, which can be stated mathematically as:

$$COST(C_i, C_j) = \min\{dist(X_i, X_j) | X_i \in C_i, X_j \in C_j\}$$

- Where *dist* is the distance between input X_i and X_j . It is typically computed using the Euclidean distance between X_i and X_j , but many other distance measures have been used. Single linkage relies only on the minimum distance between the two clusters. It does not consider how far apart the remainder of the instances in the clusters are. For this reason, single linkage could result in very "long" or spread-out clusters, depending on the structure of the two clusters being combined.
- *Complete linkage* is similar to single linkage. It begins by computing the distance between every instance in cluster C_i and C_j . However, rather than using the *minimum* distance as the cost, it uses the *maximum* distance. That is, the cost is:

$$COST(C_i, C_j) = \max\{dist(X_i, X_j) | X_i \in C_i, X_j \in C_j\}$$

- Since the maximum distance is used as the cost, clusters tend to be more compact and less spread out than in single linkage.

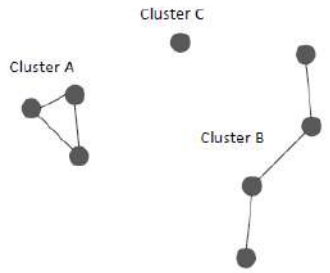


Fig. Examples of clusters in a graph formed by connecting nodes representing instances.

- A link represents a distance between the two instances that is less than some threshold value. To illustrate the difference between single-link and complete-link clusters, consider the graph shown in Figure. This graph is formed by representing in- stances (i.e., the *Xi* s) as nodes and connecting nodes where $dist(X_i,X_j) < T$, where T is some threshold value. In this graph, clusters A, B, and C would all be single-link clusters.
- The **single-link clusters** are the *connected components* of the graph, where every member of the cluster is connected to at least one other member. The complete-link clusters would be cluster A, the singleton cluster C, and the upper and lower pairs of instances from cluster B. The complete link clusters are the *cliques* or maximal complete subgraphs of the graph, where every member of the cluster is connected to every other member.
- *Average linkage* uses a cost that is a compromise between single linkage and complete linkage. As before, the distance between every pair of instances in C_i and C_j is computed. As the name implies, average linkage uses the *average* of all of the pairwise costs. Therefore, the cost is:

$$COST(C_i, C_j) = \frac{\sum_{X_i \in C_i, X_j \in C_j} dist(X_i, X_j)}{|C_i||C_j|}$$

- Where $|Ci|$ and $|Cj|$ are the number of instances in clusters *Ci* and *Cj* , respectively.
- The types of the clusters formed using average linkage depends largely on the structure of the clusters, since the cost is based on the average of the distances between every pair of instances in the two clusters.
- *Average group linkage* is closely related to average linkage. The cost is computed according to:

$$COST(C_i, C_j) = dist(\mu_{C_i}, \mu_{C_j})$$

- Where $\mu_C = \frac{\sum_{X \in C} X}{|C|}$ is the *centroid* of cluster *Ci*. The centroid of a cluster is simply the average of all of the instances in the cluster. Notice that the centroid is also a vector with the same number of dimensions as the input instances. Therefore, average group linkage represents each cluster according to its centroid and measures the cost by the distance between the centroids. The clusters formed using average group linkage are similar to those formed using average linkage.
- Figure provides a visual summary of the four cost functions described up to this point. Specifically, it shows which pairs of instances (or centroids) are involved in computing the cost functions for the set of points used in Figures.

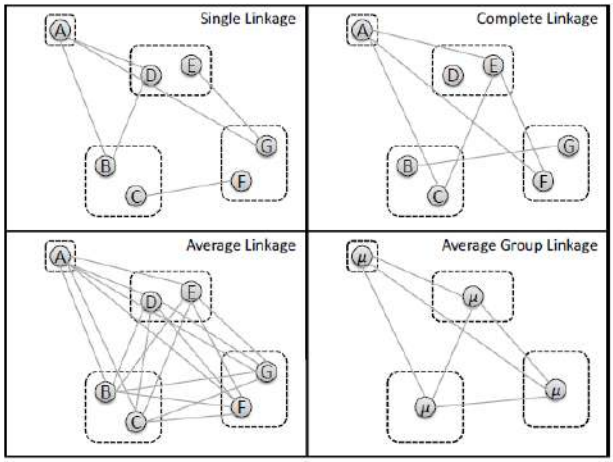


Fig. Illustration of how various clustering cost functions are computed

- **Ward's method** is based on the statistical property of *variance*. The variance of a set of numbers measures how spread out the numbers are. Ward's method attempts to minimize the sum of the variances of the clusters. This results in compact clusters with a minimal amount of spread around the cluster centroids. The cost, which is slightly more complicated to compute than the previous methods, is computed according to:

$$COST(C_i, C_j) = \sum_{k \neq i,j} \sum_{X \in C_k} (X - \mu_{C_k}) \cdot (X - \mu_{C_k}) + \sum_{X \in C_i \cup C_j} (X - \mu_{C_i \cup C_j}) \cdot (X - \mu_{C_i \cup C_j})$$

- Where $C_i \cup C_j$ is the union of the instances in clusters C_i and C_j, and $\mu_{C_i \cup C_j}$ is the centroid of the cluster consisting of the instances in $C_i \cup C_j$. This cost measures what the intracluster variance would be if clusters *i* and *j* were joined.
- So, which of the five agglomerative clustering techniques is the best? Once again the answer depends on the data set and task the algorithm is being applied to. If the underlying structure of the data is known, then this knowledge may be used to make a

more informed decision about the best algorithm to use. Typically, however, determining the best method to use requires experimentation and evaluation.

- In the information retrieval experiments that have involved hierarchical clustering, for example, *average-link* clustering has generally had the best effectiveness. Even though clustering is an unsupervised method, in the end there is still no such thing as a free lunch, and some form of manual evaluation will likely be required.
- Efficiency is a problem with all hierarchical clustering methods. Because the computation involves the comparison of every instance to all other instances, even the most efficient implementations are $O(N^2)$ for N instances. This limits the number of instances that can be clustered in an application. The next clustering algorithm we describe, *K*-means, is more efficient because it produces a *flat* clustering, or *partition*, of the instances, rather than a hierarchy.

5.5.2. K-means:

- The *K*-means algorithm is fundamentally different than the class of hierarchical clustering algorithms.. For example, with agglomerative clustering, the algorithm begins with N clusters and iteratively combines two (or more) clusters together based on how costly it is to do so. As the algorithm proceeds, the number of clusters decreases. Furthermore, the algorithm has the property that once instances Xi and X_j are in the same cluster as each other, there is no way for them to end up in different clusters as the algorithm proceeds.
- With the *K*-means algorithm, the number of clusters never changes. The algorithm starts with K clusters and ends with K clusters. During each iteration of the *K*-means algorithm, each instance is either kept in the same cluster or assigned to a different cluster. This process is repeated until some stopping criteria is met.
- The goal of the *K*-means algorithm is to find the cluster assignments, represented by the assignment vector $A[1], \ldots, A[N]$, that minimize the following cost function:

$$COST(A[1], \ldots, A[N]) = \sum_{k=1}^{K} \sum_{i:A[i]=k} dist(X_i, C_k)$$

- Where $dist(X_i, C_k)$ is the distance between instance X_i and class C_k. As with the various hierarchical clustering costs, this distance measure can be any reasonable measure, although it is typically assumed to be the following:

$$\begin{aligned} dist(X_i, C_k) &= ||X_i - \mu_{C_k}||^2 \\ &= (X_i - \mu_{C_k}) \cdot (X_i - \mu_{C_k}) \end{aligned}$$

- Which is the Euclidean distance between X_i and μC_k squared. Here, as before, μC_k is the centroid of cluster C_k. Notice that this distance measure is very similar to the cost associated with Ward's method for agglomerative clustering. Therefore, the method attempts to find the clustering that minimizes the intracluster variance of the instances.
- Alternatively, the cosine similarity between X_i and μC_k can be used as the distance measure. The cosine similarity measures the angle between two vectors. For some text

applications, the cosine similarity measure has been shown to be more effective than the Euclidean distance. This specific form of *K*-means is often called *spherical K-means*.

- One of the most naïve ways to solve this optimization problem is to try every possible combination of cluster assignments. However, for large data sets this is computationally intractable, because it requires computing an exponential number of costs. Rather than finding the *globally* optimal solution, the *K*-means algorithm finds an approximate, heuristic solution that iteratively tries to minimize the cost. This solution returned by the algorithm is not guaranteed to be the global optimal. In fact, it is not even guaranteed to be locally optimal. Despite its heuristic nature, the algorithm tends to work very well in practice.
- Algorithm 2 lists the pseudocode for one possible *K*-means implementation. The algorithm begins by initializing the assignment of instances to clusters. This can be done either randomly or by using some knowledge of the data to make a more informed decision. An iteration of the algorithm then proceeds as follows.
- Each instance is assigned to the cluster that it is closest to, in terms of the distance measure $dist(X_i, C_k)$. The variable *change* keeps track of whether any of the instances changed clusters during the current iteration. If some have changed, then the algorithm proceeds. If none have changed, then the algorithm ends. Another reasonable stopping criterion is to run the algorithm for some fixed number of iterations.
- In practice, *K*-means clustering tends to converge very quickly to a solution. Even though it is not guaranteed to find the optimal solution, the solutions returned are often optimal or close to optimal. When compared to hierarchical clustering, *K*-means is more efficient. Specifically, since *KN* distance computations are done in every iteration and the number of iterations is small, implementations of *K*-means are $O(KN)$ rather than the $O(N^2)$ complexity of hierarchical methods. Although the clusters produced by *K*-means depend on the starting points chosen (the initial clusters) and the ordering of the input data, *K*-means generally produces clusters of similar quality to hierarchical methods.

Algorithm 2 K-Means Clustering

```
1: procedure KMeansCluster(X_1, ..., X_N, K)
2:     A[1], ..., A[N] ← initial cluster assignment
3:     repeat
4:         change ← false
5:         for i = 1 to N do
6:             k̂ ← arg min_k dist(X_i, C_k)
7:             if A[i] is not equal k̂ then
8:                 A[i] ← k̂
9:                 change ← true
10:            end if
11:        end for
12:    until change is equal to false return A[1], ..., A[N]
13: end procedure
```

Therefore, *K*-means is a good choice for an all-purpose clustering algorithm for a wide range of search engine–related tasks, especially for large data sets.

5.5.3. *K* -Nearest Neighbor Clustering:

- Even though hierarchical and *K*-means clustering are different from an algorithmic point of view, one thing that they have in common is the fact that both algorithms place every input into exactly one cluster, which means that clusters do not overlap. Therefore, these algorithms *partition* the input instances into *K* partitions (clusters).
- However, for certain tasks, it may be useful to allow clusters to overlap. One very simple way of producing overlapping clusters is called *K nearest neighbor clustering*. It is important to note that the *K* here is very different from the *K* in *K*-means clustering.

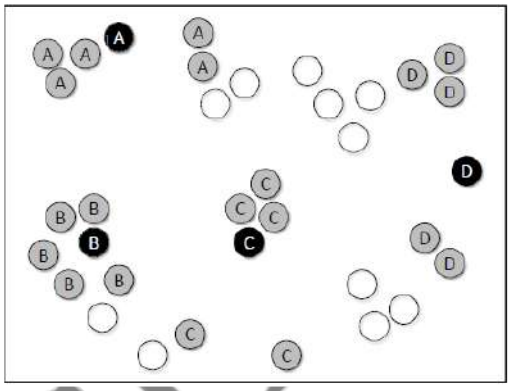


Fig. Example of overlapping clustering using nearest neighbor clustering with $K = 5$. The overlapping clusters for the black points (A, B, C, and D) are shown. The five nearest neighbors for each black point are shaded gray and labeled accordingly.

- In *K* nearest neighbor clustering, a cluster is formed around every input instance. For input instance *x*, the *K* points that are nearest to *x* according to some distance metric and *x* itself form a cluster. Figure shows several examples of nearest neighbor clusters with *K*=5 formed for the points *A*, *B*, *C*, and *D*. Although the figure only shows clusters around four input instances, in reality there would be one cluster per input instance, resulting in *N* clusters.
- The algorithm often fails to find meaningful clusters. In sparse areas of the input space, such as around *D*, the points assigned to cluster *D* are rather far away and probably should not be placed in the same cluster as *D*.
- However, in denser areas of the input space, such as around *B*, the clusters are better defined, even though some related inputs may be missed because *K* is not large enough. Applications that use *K* nearest neighbor clustering tend to emphasize finding a small number of closely related instances in the *K* nearest neighbors (i.e., precision) over finding all the closely related instances (recall).
- *K* nearest neighbor clustering can be rather expensive, since it requires computing distances between every pair of input instances. If we assume that computing the distance between two input instances takes constant time with respect to *K* and *N*, then this computation takes $O(N^2)$ time. After all of the distances have been computed, it takes at most $O(N^2)$ time to find the *K* nearest neighbors for each point. Therefore, the total time

complexity for K nearest neighbor clustering is $O(N^2)$, which is the same as hierarchical clustering.

- For certain applications, K nearest neighbor clustering is the best choice of clustering algorithm. The method is especially useful for tasks with very dense input spaces where it is useful or important to find a number of related items for every input. Examples of these tasks include language model smoothing, document score smoothing, and pseudo-relevance feedback.

Evaluation:

- Evaluating the output of a clustering algorithm can be challenging. Since clustering is an unsupervised learning algorithm, there is often little or no labeled data to use for the purpose of evaluation. When there is no labeled training data, it is sometimes possible to use an objective function, such as the objective function being minimized by the clustering algorithm, in order to evaluate the quality of the clusters produced. This is a chicken and egg problem, however, since the evaluation metric is defined by the algorithm and vice versa.
- If some labeled data exists, then it is possible to use slightly modified versions of standard information retrieval metrics, such as precision and recall, to evaluate the quality of the clustering. Clustering algorithms assign each input instance to a cluster identifier. The cluster identifiers are arbitrary and have no explicit meaning.
- For example, if we were to cluster a set of emails into two clusters, some of the emails would be assigned to cluster identifier 1, while the rest would be assigned to cluster 2. Not only do the cluster identifiers have no meaning, but the clusters may not have a meaningful interpretation. For example, one would hope that one of the clusters would correspond to "spam" emails and the other to "non-spam" emails, but this will not necessarily be the case. Therefore, care must be taken when defining measures of precision and recall.
- One common procedure of measuring precision is as follows. First, the algorithm clusters the input instances into $K = |C|$ clusters. Then, for each cluster C_i, we define MaxClass(C_i) to be the (human-assigned) class label associated with the most instances in C_i . Since MaxClass(C_i) is associated with more of the instances in C_i than any other class label, it is assumed that it is the *true* label for cluster C_i . Therefore, the precision for cluster C_i is the fraction of instances in the cluster with label MaxClass(C_i). This measure is often micro averaged across instances, which results in the following measure of precision:

$$ClusterPrecision = \frac{\sum_{i=1}^{K} |\text{MaxClass}(C_i)|}{N}$$

- Where |MaxClass(C_i)| is the total number of instances in cluster C_i with the label MaxClass(C_i). This measure has the intuitive property that if each cluster corresponds to exactly one class label and every member of a cluster has the same label, then the measure is 1.
- In many search applications, clustering is only one of the technologies that are being used. Typically, the output of a clustering algorithm is used as part of some complex end-to-end system. In these cases, it is important to analyze and evaluate how the clustering

algorithm affects the entire end-to-end system. For example, if clustering is used as a subcomponent of a web search engine in order to improve ranking, then the clustering algorithm can be evaluated and tuned by measuring the impact on the effectiveness of the ranking. This can be difficult, as end-to-end systems are often complex and challenging to understand, and many factors will impact the ranking.

How to Choose *K:*

- Thus far, we have largely ignored how to choose *K*. In hierarchical and *K*-means clustering, *K* represents the number of clusters. In *K* nearest neighbors smoothing, *K* represents the number of nearest neighbors used. Although these two things are fundamentally different, it turns out that both are equally challenging to set properly in a fully automated way. The problem of choosing *K* is one of the most challenging issues involved with clustering, since there is really no good solution.
- No magical formula exists that will predict the optimal number of clusters to use in every possible situation. Instead, the best choice of *K* largely depends on the task and data set being considered. Therefore, *K* is most often chosen experimentally.
- In some cases, the application will dictate the number of clusters to use. This, however, is rare. Most of the time, the application offers no clues as to the best choice of *K*. In fact, even the range of values for *K* to try might not be obvious. Should 2 clusters be used? 10? 100? 1,000? There is no better way of getting an understanding of the best setting for *K* than running experiments that evaluate the quality of the resulting clusters for various values of *K*.
- With hierarchical clustering, it is possible to create the entire hierarchy of clusters and then use some decision mechanism to decide what level of the hierarchy to use for the clustering. In most situations, however, the number of clusters has to be manually chosen, even with hierarchical clustering.

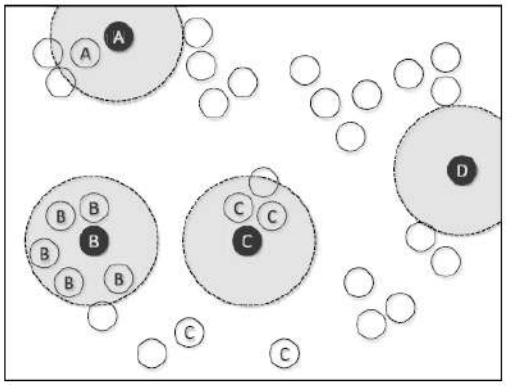


Fig. Example of overlapping clustering using Parzen windows. The clusters for the black points (A, B, C, and D) are shown. The shaded circles indicate the windows used to determine cluster membership. The neighbors for each black point are shaded gray and labeled accordingly.

- When forming *K* nearest neighbor clusters, it is possible to use an adaptive value for *K*. That is, for instances in very dense regions, it may be useful to choose a large *K*, since the neighbors are likely to be related. Similarly, in very sparse areas, it may be best to choose

only a very small number of nearest neighbors, since moving too far away may result in unrelated neighbors being included. This idea is closely related to *Parzen windows*, which are a variant of *K* nearest neighbors used for classification. With Parzen windows, the number of nearest neighbors is not fixed.

- Instead, all of the neighbors within a fixed distance ("window") of an instance are considered its nearest neighbors. In this way, instances in dense areas will have many nearest neighbors, and those in sparse areas will have few.
- In Figure, see that fewer outliers get assigned to incorrect clusters for points in sparse areas of the space (e.g., point C), whereas points in denser regions have more neighbors assigned to them (e.g., B). However, the clusters formed are not perfect. The quality of the clusters now depends on the size of the window. Therefore, although this technique eliminates the need to choose *K*, it introduces the need to choose the window size, which can be an equally challenging problem.

5.5.4. EXPECTATION MAXIMIZATION (EM) ALGORITHM:

- Expectation Maximization is a type of model based clustering method. It attempts to optimize the fit between the given data and some mathematical model. Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions.
- The EM algorithm is an extension of the K-Means algorithm. It is iterative in nature and finds maximum likelihood solutions.
- Expectation Maximization consists of two steps:
  1) The expectation step assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters.
  2) The maximization step finds the new clustering or parameters that maximize the expected likelihood in probabilistic model based clustering.

Algorithm:

Input: c: the number of clusters
D: a dataset containing n objects
output: A set of k clusters

Method:

1) First find initial centers/centroids which will be the initial input.

2) Compute distance between each data point and each centroid using cosine distance formula or any other distance formula.

3) Assign weights for each combination of data point and cluster based on the probability of membership of a data point to a particular cluster.

4) Repeat

i) (Re) assign each data point to the cluster with which it has highest weight i.e., highest probability.

ii) If a data point belongs to more than one cluster with the same probability, then (re)assign the data point to the cluster based on minimum distance.

iii) Update the cluster means for every iteration until clustering converges.

- EM has a strong statistical basis, it is linear in database size, it is robust to noisy data, it can accept the desired number of clusters as input, it provides a cluster membership probability per point, it can handle high dimensionality and it converges fast given a good initialization.
- EM offers many advantages besides having a strong statistical basis and being efficient. One of those advantages is that EM is robust to noisy data and missing information. In fact, EM is intended for incomplete data. The complexity of EM depends upon the number of iterations and time to compute E and M steps.

CS6007 INFORMATION RETRIEVAL

UNIT- 5

TWO MARKS:

1. What is the goal of filtering?
   In filtering, the user's information need stays the same, but the document collection itself is dynamic, with new documents arriving periodically. The goal of filtering, then, is to identify (filter) the relevant new documents and send them to the user. Filtering is a *push* application.

2. What are the two components of Document filtering systems?
   First, the user's long term information needs must be accurately represented. This is done by constructing a *profile* for every information need.
   Second, given a document that has just arrived in the system, a decision mechanism must be devised for identifying which are the relevant profiles for that document. This decision mechanism must not only be efficient, especially since there are likely to be thousands of profiles, but it must also be highly accurate.

3. What are the two common types of filtering models?
   - The first are *static* models. Here, static refers to the fact that the user's profile does not change over time, and therefore the same model can always be applied.
   - The second are *Adaptive* models, where the user's profile is constantly changing over time. This scenario requires the filtering model to be dynamic over time as new information is incorporated into the profile.

4. What is the purpose of text mining?
   The purpose of Text Mining is to process unstructured (textual) information, extract meaningful numeric indices from the text, and, thus, make the information contained in the text accessible to the various data mining (statistical and machine learning) algorithms.

5. What do you mean by two-class classification?
   Given a set of *classes*, we seek to determine which class(es) a given object belongs to. In the example, the standing query serves to divide new newswire articles into

the two classes: documents about multicore computer chips and documents not about multicore computer chips. This is referred as *two-class classification*. Classification using standing queries is also called *routing* or filtering.

6. What is the difference between Classification and clustering?

   Classification, also referred to as categorization, is the task of automatically applying labels to data, such as emails, web pages, or images. People classify items throughout their daily lives. It would be infeasible, however, to manually label every page on the Web according to some criteria, such as "spam" or "not spam." Therefore, there is a need for automatic classification and categorization techniques.

   Clustering can be broadly defined as the task of grouping related items together. In classification, each item is assigned a label, such as "spam" or "not spam." In clustering, however, each item is assigned to one or more clusters, where the cluster does not necessarily correspond to a meaningful concept, such as "spam" or "not spam". items are grouped together according to their similarity. Therefore, rather than mapping items onto a predefined set of labels, clustering allows the data to "speak for itself " by uncovering the implicit structure that relates the items.

7. What is the difference between supervised learning and unsupervised learning?

   In machine learning, the learning algorithms are typically characterized as supervised or unsupervised. In *supervised learning*, a model is learned using a set of fully labeled items, which is often called the *training set*.

   Once a model is learned, it can be applied to a set of unlabeled items, called the *test set*, in order to automatically apply labels. Classification is often cast as a supervised learning problem. For example, given a set of emails that have been labeled as "spam" or "not spam" (the training set), a classification model can be learned. The model then can be applied to incoming emails in order to classify them as "spam" or "not spam".

   Unsupervised learning algorithms learn entirely based on unlabeled data. Unsupervised learning tasks are often posed differently than supervised learning tasks, since the input data is not mapped to a predefined set of labels. Clustering is the most common example of unsupervised learning. Clustering algorithms take a set of unlabeled data as input and then group the items using some notion of similarity.

8. How the classification is performed in naïve bayes classification?

   Naïve Bayes classifies a document *d* as follows:

$$\begin{aligned}\text{Class}(d) &= \arg\max_{c\in\mathcal{C}} P(c|d) \\ &= \arg\max_{c\in\mathcal{C}} \frac{P(d|c)P(c)}{\sum_{c\in C} P(d|c)P(c)}\end{aligned}$$

   Where $\arg\max_{c\in\mathcal{C}} P(c|d)$ means "return the class *c*, out of the set of all possible classes *C*, that maximizes *P*(*c*|*d*)." This is a mathematical way of saying that we are trying to find the most likely class *c* given the document *d*.

9. *What is meant by multiple-Bernoulli* event space?

   Given a class *c*, we define a *binary* random variable w_i for every term in the vocabulary. The outcome for the binary event is either 0 or 1. The probability $P(wi = 1|c)$ can then be interpreted as "the probability that term w_i is generated by class *c*." Conversely, $P(wi = 0|c)$ can be interpreted as "the probability that term w_i is not generated by class *c*." This is exactly the event space used by the binary independence model and is known as the *multiple-Bernoulli* event space.

10. What if the difference between *multinomial* event space and multiple-Bernoulli event space?

    The *multinomial* event space is very similar to the multiple-Bernoulli event space, except rather than assuming that term occurrences are binary ("term occurs" or "term does not occur"), it assumes that terms occur zero or more times ("term occurs zero times", "term occurs one time", etc.).

11. Define Divisive clustering.

Divisive clustering algorithms begin with a single cluster that consists of all of the instances. During each iteration it chooses an existing cluster and divides it into two (or possibly more) clusters. This process is repeated until there are a total of *K* clusters. The output of the algorithm largely depends on how clusters are chosen and split. Divisive clustering is a *top-down* approach.

12. *Define agglomerative clustering.*

Agglomerative clustering is a *bottom-up* approach. An agglomerative algorithm starts with each input as a separate cluster. That is, it begins with *N* clusters, each of which contains a single input. The algorithm then proceeds by joining two (or possibly more) existing clusters to form a new cluster. Therefore, the number of clusters decreases after each iteration. The algorithm terminates when there are *K* clusters.

13. What is the use of *dendrogram*?

The hierarchy generated by an agglomerative or divisive clustering algorithm can be conveniently visualized using a *dendrogram*. A dendrogram graphically represents how a hierarchical clustering algorithm progresses.

14. What is meant by single-link clusters?

The single-link clusters are the *connected components* of the graph, where every member of the cluster is connected to at least one other member. The complete-link clusters would be cluster A, the singleton cluster C, and the upper and lower pairs of instances from cluster B. The complete link clusters are the *cliques* or maximal complete subgraphs of the graph, where every member of the cluster is connected to every other member.

15. What is meant by complete-link clusters?

The complete link clusters are the *cliques* or maximal complete subgraphs of the graph, where every member of the cluster is connected to every other member.

16. Define K-mean algorithm.

With the *K*-means algorithm, the number of clusters never changes. The algorithm starts with *K* clusters and ends with *K* clusters. During each iteration of the *K*-means algorithm, each instance is either kept in the same cluster or assigned to a different cluster. This process is repeated until some stopping criteria is met.

17. How is cluster formed in *K* nearest neighbor clustering?

In *K* nearest neighbor clustering, a cluster is formed around every input instance. For input instance *x*, the *K* points that are nearest to *x* according to some distance metric and *x* itself form a cluster. Figure shows several examples of nearest neighbor clusters with *K*=5 formed for the points *A*, *B*, *C*, and *D*. Although the figure only shows clusters around four input instances, in reality there would be one cluster per input instance, resulting in *N* clusters.

18. What is an EM algorithm.

An expectation–maximization (EM) algorithm is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables.

19. What is the use of decision tree?

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm.

UNIT-V
PART-B

1. Explain about Information filtering.(8)
2. Describe about text mining. (8)
3. Explain various Categorization algorithms.(16)
4. Explain about multiple Bernoulli model. (8)
5. Describe about multinomial model.(8)
6. Explain about naive Bayes classification. (8)
7. Explain the steps to create decision tree. (8)
8. Describe about K- nearest neighbor algorithm(8)
9. Explain in detail about various Clustering algorithms.(16)
10. Briefly explain about agglomerative clustering. (8)
11. Describe bout K-means algorithm.(8)
12. Explain the expectation maximization (EM) algorithm. (8)