CS8080 INFORMATION RETRIEVAL TECHNIQUES L T P C
3 0 0 3

OBJECTIVES:

- To understand the basics of Information Retrieval.
- To understand machine learning techniques for text classification and clustering.
- To understand various search engine system operations.
- To learn different techniques of recommender system.

UNIT I INTRODUCTION 9

Information Retrieval – Early Developments – The IR Problem – The User‘s 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.

UNIT II MODELING AND RETRIEVAL EVALUATION 9

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.

UNIT III TEXT CLASSIFICATION AND CLUSTERING 9

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.

UNIT IV WEB RETRIEVAL AND WEB CRAWLING 9

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.

UNIT V RECOMMENDER SYSTEM 9

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.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, —Modern Information Retrieval: The Concepts and Technology behind Search, Second Edition, ACM Press Books, 2011.
2. Ricci, F, Rokach, L. Shapira, B.Kantor, “Recommender Systems Handbook”, First Edition, 2011.

REFERENCES:

1. C. Manning, P. Raghavan, and H. Schütze, —Introduction to Information Retrieval, Cambridge University Press, 2008.
2. Stefan Buettcher, Charles L. A. Clarke and Gordon V. Cormack, —Information Retrieval: Implementing and Evaluating Search Engines, The MIT Press, 2010.

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.

INFORMATION RETRIEVAL

Information retrieval (IR) is a broad area of Computer Science focused primarily on providing the users with easy access to information of their interest, as follows. Information retrieval deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogs, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be such as to provide the users with easy access to information of their interest. Nowadays, research in IR includes modeling, Web search, text classification, systems architecture, user interfaces, data visualization, filtering, languages.

EARLY DEVELOPMENTS

For more than 5, 000 years, man has organized information f or later retrieval and searching. In its most usual form, this has been done by compiling, storing, organizing, and indexing clay tablets, hieroglyphics, papyrus rolls, and books. For holding the various items, special purpose buildings called libraries, from the Latin word liber for book, or bibliothekes, from the Greek word biblion for papyrus roll, are used. The oldest known library was created in Elba, in the "Fertile Crescent", currently northern Syria, some time between 3,000 and 2,500 BC. In the seventh century BC, Assyrian king Ashurbanipal created the library of Nineveh, on the Tigris River (today, north of Iraq), which contained more than 30,000 clay tablets at the time of its destruction in 612 BC. By 300 BC, Ptolemy Soter, a Macedonian general, created the Great Library in Alexandria – the

Egyptian city at the mouth of the Nile named after the Macedonian king Alexander the Great (356-323 BC). For seven centuries the Great Library, jointly with other major libraries in the city, made Alexandria the intellectual capital of the Western world.

Since then, libraries have expanded and flourished. Nowadays, they are everywhere. They constitute the collective memory of the human race and their popularity is in the rising. In 2008 alone, people in the US visited their libraries some 1.3 billion times and checked out more than 2 billion items – an increase in both yearly figures of more than 10 percent. Since the volume of information in libraries is always growing, it is necessary to build specialized data structures for fast search – the indexes. In one form or another, indexes are at the core of every modern information retrieval system. They provide fast access to the data and allow speeding up query processing.

For centuries indexes have been created manually as sets of categories. Each category in the index is typically composed of labels that identify its associated topics and of pointers to the documents that discuss those topics. While these indexes are usually designed by library and information science researchers, the advent of modern computers has allowed the construction of large indexes automatically, which has accelerated the development of the area of Information Retrieval (IR). Early developments in IR date back to research efforts conducted in the 50's by pioneers such as Hans Peter Luhn, Eugene Garfield, Philip Bagley, and Cal vi n Moores, this last one having allegedly coined the term information retrieval. In 1955, Allen Kent and colleagues published a paper describing the precision and recall metrics, which was followed by the publication in 1962 of the Cranfield studies by Cyril Cleverdon. In 1963, Joseph Becker and Robert Hayes published the first book on information retrieval [164]. Throughout the 60's, Gerard Salton and Karen Sparck Jones, among others, shaped the field by developing the fundamental concepts that led to the modern technologies of ranking in IR.In1968,thefirstIR book authored by Salton was published. In 1971, N.

Jardine and C.J.VanRijsbergen articulated the “cluster hypothesis”. In 1978, the first ACM Conference on IR (ACM SIGIR) was held in Rochester, New York. In 1979, C.J. Van Rijsbergen published Information Retrieval, which focused on probabilistic models. In 1983, Salton and McGill published Introduction to Modern Information Retrieval, a classic book on IR focused on vector models. Since then, the IR community has grown to include thousands of professors, researchers, students, engineers, and practitioners throughout the world. The main conference in the area, the ACM International Conference on Information Retrieval (ACM SIGIR), now attracts hundreds of attendees and receives hundreds of submitted papers on an yearly basis.

THE IR PROBLEM

Users of modern IR systems, such as search engine users, have information needs of varying complexity. In the simplest case, they are looking for the link to the homepage of a company, government, or institution. In the more sophisticated cases, they are looking for information required to execute tasks associated with their jobs or immediate needs. An example of a more 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 need does not necessarily provide the best formulation for querying the IR system. Instead, the user might want to first translate this information need into a query, or sequence of queries, to be posed to the system. In its most common form, this translation 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. The emphasis is on the retrieval of information as opposed to the retrieval of data. To be effective in its attempt to satisfy the user information need, the IR system must somehow ‘interpret’ the contents of the information items i.e., the documents in a collection, and rank them according to a degree of relevance to the user query. This ‘interpretation’ of a document content involves extracting

syntactic and semantic information from the document text and using this information to match the user information need.

The IR Problem: the primary goal of an IR system is to retrieve all the documents that are relevant to a user query while retrieving as few non relevant documents as possible. The difficulty is knowing not only how to extract information from the documents but also knowing how to use it to decide relevance. That is, the notion of relevance is of central importance in IR. One main issue is that relevance is a personal assessment that depends on the task being solved and its context. For example, relevance can change with time (e.g., new information becomes available), with location (e.g., the most relevant answer is the closest one), or even with the device (e.g., the best answer is a short document that is easier to download and visualize). In this sense, no IR system can provide perfect answers to all users all the time.

THE USERS TASK

The user of a retrieval system has to translate their information need into a query in the language provided by the system. With an IR system, such as a search engine, this usually implies specifying a set of words that convey the semantics of the information need. We say that the user is searching or querying for information of their interest. While searching for information of interest is the main retrieval task on the Web, search can also be used for satisfying other user needs distinct from information access, such as the buying of goods and the placing of reservations. Consider now a user who has an interest that is either poorly defined or inherently broad, such that the query to specify is unclear. To illustrate, the user might be interested in documents about car racing in general and might decide to glance related documents about Formula 1 racing, Formula Indy, and the '24 Hours of Le Mans. We say that the user i s browsing or navigating the documents in the collection, not searching. It is still a process of retrieving information, but one whose main objectives are less clearly defined in the beginning. The task in this case is more related to exploratory search and resembles a process of quasi-

sequential search for information of interest. In this book, we make a clear distinction between the different tasks the user of the retrieval system might be engaged in. The task might be then of two distinct types: searching and browsing,asillustratedinFigure1.1.

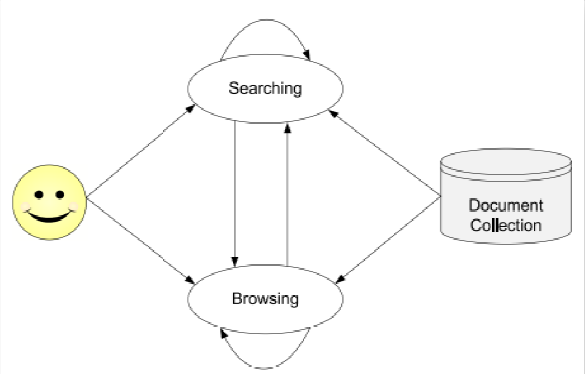


Figure 1.1: The tasks of the user.

INFORMATION VERSUS DATA RETRIEVAL

Data retrieval, in the context of an IR system, consists mainly of determining which documents of a collection contain the keywords in the user query which, most frequently, is not enough to satisfy the user information need. In fact, the user of an IR system is concerned more with retrieving information about a subject than with retrieving data that satisfies a given query. For instance, a user of an IR system is willing to accept documents that contain synonyms of the query terms in the result set, even when those documents do not contain any query terms.

That is, in an IR system the retrieved objects might be inaccurate and small errors are likely to go unnoticed. In a data retrieval system, on the contrary, a single erroneous object among a thousand retrieved objects means total failure. A data retrieval system, such as a relational database, deals with data that has a well defined structure and semantics, while an IR system deals with natural language text which is not well structured. Data retrieval, while providing a solution to the user of a database system, does not solve the problem of retrieving information about a subject or topic.

Sn	Data	Information
1	unorganized raw facts that need processing without which it is seemingly random and useless to humans	Information is a processed, organized data presented in a given context and is useful to humans.
2	Data is an individual unit that contains raw material which does not carry any specific meaning	Information is a group of data that collectively carry a logical meaning
3	Data doesn't depend on information.	Information depends on data.
4	It is measured in bits and bytes.	Information is measured in meaningful units like time, quantity, etc.
5	Data is never suited to the specific needs of a designer	Information is specific to the expectations and requirements because all the irrelevant facts and figures are removed, during the transformation process.
6	An example of data is a student's test score	The average score of a class is the information derived from the given data.

THE IR SYSTEM

In this section we provide a high level view of the software architecture of an IR system. We also introduce the processes of retrieval and ranking of documents in response to a user query.

THE SOFTWARE ARCHITECTURE OF THE IR SYSTEM

To describe the IR system, we use a simple and generic software architecture as shown in Figure 1.2.

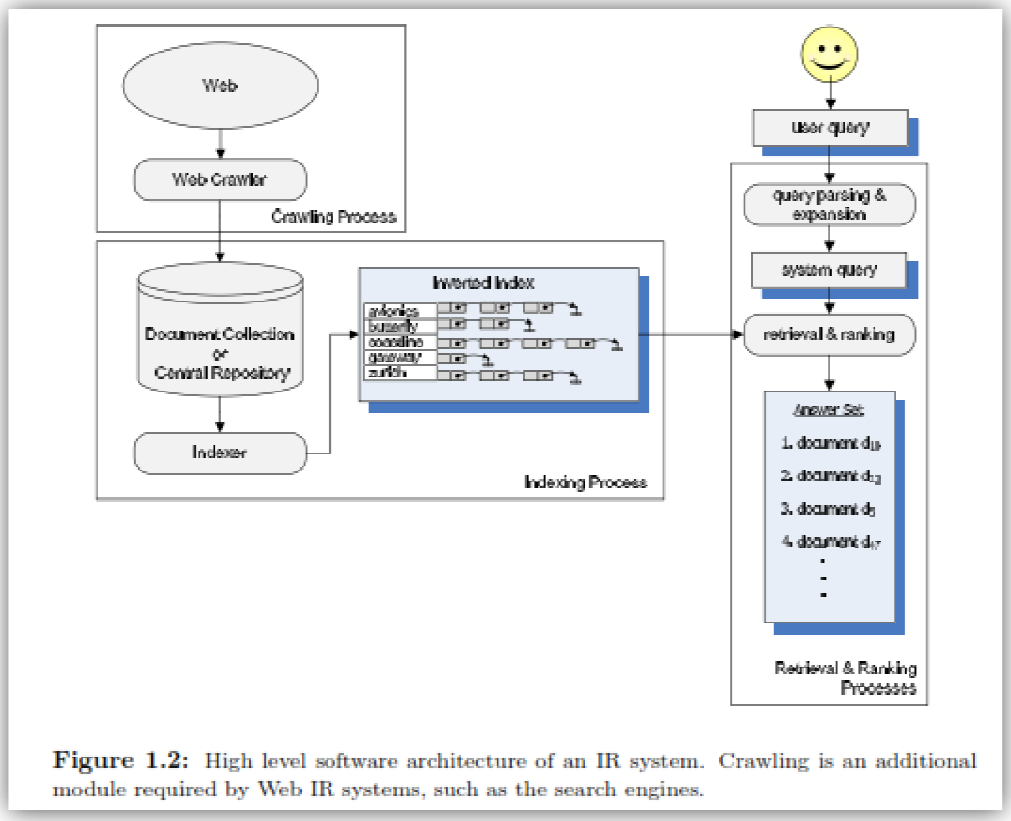


Figure 1.2: High level software architecture of an IR system. Crawling is an additional module required by Web IR systems, such as the search engines.

The first step in setting up an IR system is to assemble the document collection, which can be private or be crawled from the Web. In the second case a crawler module is responsible for collecting the documents. The document collection is stored in disk storage usually referred to as the central repository. The documents in the central repository need to be indexed for fast retrieval and ranking. The most used index structure is an inverted index composed of all the distinct words of the collection and, for each word, a list of the documents that contain it.

Given that the document collection is indexed, the retrieval process can be initiated. It consists of retrieving documents that satisfy either a user query or a click in a hyper link. In the first case, we say that the user is searching for information of interest; in the second case, we say that the user is browsing for information of interest. In the remaining of this section, we use retrieval as it applies to the searching process. For a more detailed discussion on browsing and how it compares to searching. To search, the user first specifies a query that reflects

their information need. Next, the user query is parsed and expanded with, for instance, spelling variants of a query word. The expanded query, which we refer to as the system query, is then processed against the index to retrieve a subset of all documents. Following, the retrieved documents are ranked and the top documents are returned to the user.

The purpose of ranking is to identify the documents that are most likely to be considered relevant by the user, and constitutes the most critical part of the IR system. Given the inherent subjectivity in deciding relevance, evaluating the quality of the answer set is a key step for improving the IR system. A systematic evaluation process allows fine tuning the ranking algorithm and improving the quality of the results. The most common evaluation procedure consists of comparing the set of results produced by the IR system with results suggested by human specialists.

To improve the ranking, we might collect feedback from the users and use this information to change the results. In the Web, the most abundant form of user feedback are the clicks on the documents in the results set. Another important source of information for Web ranking are the hyperlinks among pages, which allow identifying sites of high authority. There are many other concepts and technologies that bear impact on the design of a full fledged IR system, such as a modern search engine.

THE RETRIEVAL AND RANKING PROCESSES

To describe the retrieval and ranking processes, we further elaborate on our description of the modules shown in Figure 1.2, as illustrated in Figure 1.3. Given the documents of the collection, we first apply text operations to them such as eliminating stop words, stemming, and selecting a subset of all terms for use as indexing terms. The indexing terms are then used to compose document representations, which might be smaller than the documents themselves (depending on the subset of index terms selected). Given the document representations, it is

necessary to build an index of the text. Different index structures might be used, but the most popular one is an inverted index. The steps required to generate the index compose the indexing process and must be executed offline, before the system is ready to process any queries.

The resources (time and storage space) spent on the indexing process are amortized by querying the retrieval system many times. Given that the document collection is indexed, the retrieval process can be initiated. The user first specifies a query that reflects their information need. This query is then parsed and modified by operations that resemble those applied to the documents. Typical operations at this point consist of spelling corrections and elimination of terms such as stop words, whenever appropriated. Next, the transformed query is expanded and modified. For instance, the query might be modified using query suggestions made by the system and confirmed by the user.

The expanded and modified query is then processed to obtain the set of retrieved documents, which is composed of documents that contain the query terms. Fast query processing is made possible by the index structure previously built. The steps required to produce the set of retrieved documents constitute the retrieval process. Next, the retrieved documents are ranked according to a likelihood of relevance to the user. This is a most critical step because the quality of the results, as perceived by the users, is fundamentally dependent on the ranking.

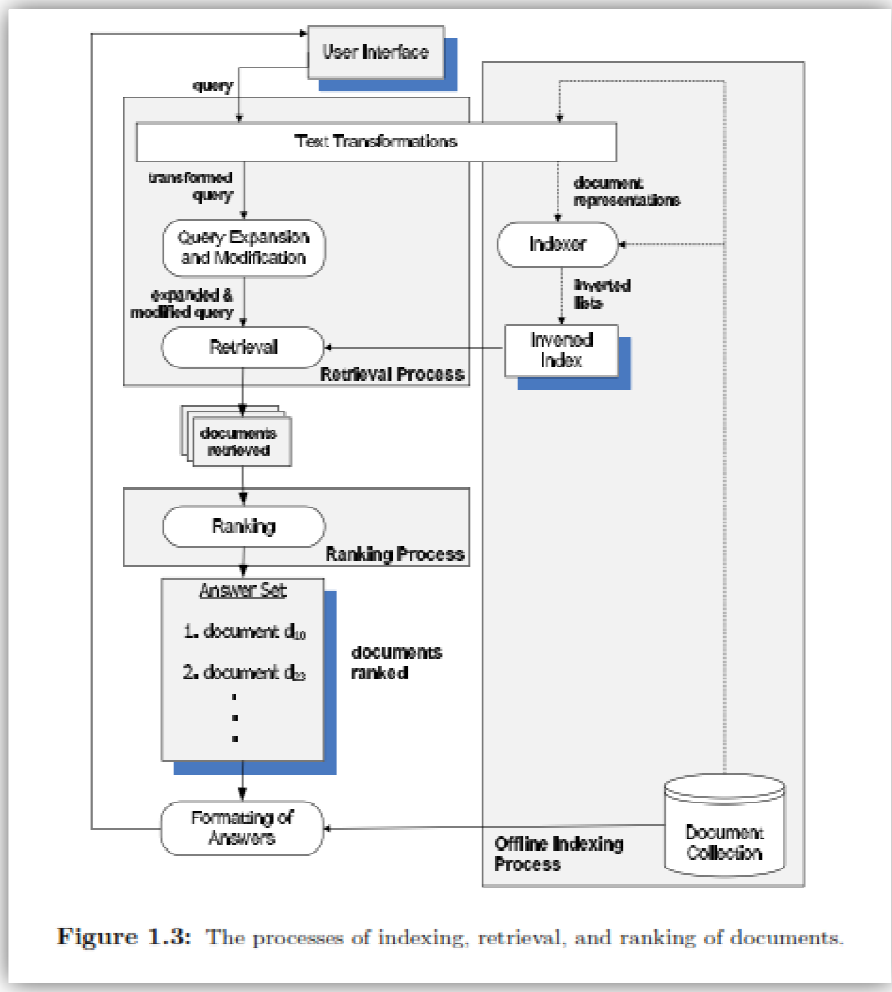


Figure 1.3: The processes of indexing, retrieval, and ranking of documents.

the ranking process in great detail. The top ranked documents are then formatted for presentation to the user. The formatting consists of retrieving the title of the documents and generating snippets for them, i.e., text excerpts that contain the query terms, which are then displayed to the user.

THE WEB

In this section we discuss the creation of the Web and its major implication–the advent of the e-publishing age. We also discuss how the Web changed search, i.e., he major impacts of the Web on the search task. At the end, we cover practical issues, such as security and copyright, which derive directly from the massive presence of millions of users on the Web.

A Brief History

"As We May Think" influenced people like Douglas Engelbart who, at the Fall Joint Computer Conference in San Francisco in December of 1968, ran a demonstration in which he introduced the first ever computer mouse, video conferencing, teleconferencing, and hypertext. It was so incredible that it became known as "the mother of all demos" [1690]. Of the innovations displayed, the one that interests us the most here is hypertext. The term was coined by Ted Nelson in his Project Xanadu.

Hypertext allows the reader to jump from one electronic document to another, which was one important property regarding the problem Tim Berners-Lee faced in 1989. At the time, Berners-Lee worked in Geneva at the CERN – Conseil Europeen pourla Recherche Nucleaire. There, researchers who wanted to share documentation with others had to reformat their documents to make them compatible with an internal publishing system. It was annoying and generated many questions, many of which ended up been directed towards Berners-Lee. He understood that a better solution was required.

It just so happened that CERN was the largest Internet node in Europe. Berners Lee reasoned that it would be nice if the solution to the problem of sharing documents were decentralized, such that the researchers could share their contributions freely. He saw that a networked hypertext, through the Internet, would be a good solution and started working on its implementation. In 1990, he wrote the HTTP protocol, defined the HTML language, wrote the first browser, which he called "World Wide Web", and the first Web server. In 1991, he made hi s browser and server software available in the Internet. The Web was born.

THE E-PUBLISHING ERA

Since its inception, the Web became a huge success. The number of Web pages now far exceeds 20 billion and the number of Web users in the world exceeds 1.7 billion. Further, it is known that there are more than one trillion distinct URLs on the Web, even if many of them are pointers to dynamic pages, not static

HTML pages. Further, a viable model of economic sustainability based on online advertising was developed.

The advent of the Web changed the world in a way that few people could have anticipated. Yet, one has to wonder on the characteristics of the Web that have made it so successful. Is there a single characteristic of the Web that was most decisive for its success? Tentative answers to this question include the simple HTML markup language, the low access costs, the wide spread reach of the Internet, the interactive browser interface, the search engines. However, while providing the fundamental infrastructure for the Web, these technologies were not the root cause of its popularity. What was it then? The fundamental shift in human relationships, introduced by the Web, was freedom to publish. Jane Austen did not have that freedom, so she had to either convince a publisher of the quality of her work or pay for the publication of an edition of it herself. Since she could not pay for it, she had to be patient and wait for the publisher to become convinced. It took 15 years.

In the world of the Web, this is no longer the case. People can now publish their ideas on the Web and reach millions of people over night, without paying anything for it and without having to convince the editorial board of a large publishing company. That is, restrictions imposed by mass communication media companies and by natural geographical barriers were almost entirely removed by the invention of the Web, which has led to a freedom to publish that marks the birth of a new era. One which we refer to as The e-Publishing Era.

HOW THE WEB CHANGED SEARCH

Web search is today the most prominent application of IR and its techniques. Indeed, the ranking and indexing components of any search engine are fundamentally IR pieces of technology. An immediate consequence is that the Web has had a major impact in the development of IR, as we now discuss. The first major impact of the Web on search is related to the characteristics of the document

collection itself. The Web collection is composed of documents (or pages) distributed over millions of sites and connected through hyperlinks , i.e., links that associate a piece of text of a page with other Web pages. The inherent distributed nature of the Web collection 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.

The second major impact of the Web on search is related to the size of the collection and the volume of user queries submitted on a daily basis. Given that the Web grew larger and faster than any previous known text collection, the search engines have now to handle a volume of text that far exceeds 20 billion pages, i.e., a volume of text much larger than any previous text collection. Further, the volume of user queries is also much larger than ever before, even if estimates vary widely. The combination of a very large text collection with a very high query traffic has pushed the performance and scalability of search engines to limits that largely exceed those of any previous IR system.

That is, performance and scalability have become critical characteristics of the IR system, much more than they used to be prior to the Web. While we do not discuss performance and scalability of search engines. The third major impact of the Web on search is also related to the vast size of the document collection. In a very large collection, predicting relevance is much harder than before. Basically, any query retrieves a large number of documents that match its terms, which means that there are many noisy documents in the set of retrieved documents. That is, documents that seem related to the query but are actually not relevant to it according to the judgement of a large fraction of the users are retrieved.

This problem first showed up in the early Web search engines and became more severe as the Web grew. Fortunately, the Web also includes new sources of evidence not present in standard document collections that can be used to alleviate the problem, such as hyperlinks and user clicks in documents in the answer set.

Two other major impacts of the Web on search derive from the fact that the Web is not just a repository of documents and data, but also a medium to do business. One immediate implication is that the search problem has been extended beyond the seeking of text information to also encompass other user needs such as the price of a book, the phone number of a hotel, the link for downloading a software. Providing effective answers to these types of information needs frequently requires identifying structured data associated with the object of interest such as price, location, or descriptions of some of its key characteristics.

The fifth and final impact of the Web on search derives from Web advertising and other economic incentives. The continued success of the Web as an interactive media for the masses created incentives for its economic exploration in the form of, for instance, advertising and electronic commerce. These incentives led also to the abusive availability of commercial information disguised in the form of purely informational content, which is usually referred to as Web spam.

The increasingly pervasive presence of spam on the Web has made the quest for relevance even more difficult than before, i.e., spam content is sometimes so compelling that it is confused with truly relevant content. Because of that, it is not unreasonable to think that spam makes relevance negative, i.e., the presence of spam makes the current ranking algorithms produce answers sets that are worst than they would be if the Web were spam free. This difficulty is so large that today we talk of Adversarial Web Retrieval.

PRACTICAL ISSUES ON THE WEB

Electronic commerce is a major trend on the Web nowadays and one which has benefited millions of people. In an electronic transaction, the buyer usually submits to the vendor credit information to be used for charging purposes. In its most common form, such information consists of a credit card number. For security reasons, this information is usually encrypted, as done by institutions and companies that deploy automatic authentication processes.

Besides security, another issue of major interest is privacy. Frequently, people are willing to exchange information as long as it does not become public. The reasons are many, but the most common one is to protect oneself against misuse of private information by third parties. Thus, privacy is another issue which affects the deployment of the Web and which has not been properly addressed yet.

Two other important issues are 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. This is important because it affects the business of building up and deploying large digital libraries. For instance, is a site which supervises all the information it posts acting as a publisher? And if so, is it responsible for misuse of the information it posts (even if it is not the source)? Additionally, other practical issues of interest include scanning, optical character recognition (OCR), and cross-language retrieval (in which the query is in one language but the documents retrieved are in another language).

HOW PEOPLE SEARCH

Search tasks range from the relatively simple (e.g., looking up disputed facts or finding weather information) to the rich and complex (e.g., job seeking and planning vacations). Search interfaces should support a range of tasks, while taking into account how people think about searching for information. This section summarizes theoretical models about and empirical observations of the process of online information seeking.

Information Lookup versus Exploratory Search

User interaction with search interfaces differs depending on the type of task, the amount of time and effort available to invest in the process, and the domain expertise of the information seeker. The simple interaction dialogue used in Web search engines is most appropriate for finding answers to questions or to finding Web sites or other resources that act as search starting points. But, as Marchionini

notes, the "turn-taking" interface of Web search engines is inherently limited and in many cases is being supplanted by speciality search engines – such as for travel and health information – that offer richer interaction models.

Classic versus Dynamic Model of Information Seeking

Researchers have developed numerous theoretical models of how people go about doing search tasks. The classic notion of the information seeking process model as described by Sutcliffe and Ennis is formulated as a cycle consisting of four main activities:

- problem identification,
- articulation of information need(s),
- query formulation, and
- results evaluation.

The standard model of the information seeking process contains an underlying assumption that the user's information need is static and the information seeking process is one of successively refining a query until all and only those documents relevant to the original information need have been retrieved. More recent models emphasize the dynamic nature of the search process, noting that users learn as they search, and their information needs adjust as they see retrieval results and other document surrogates. This dynamic process is sometimes referred to as the berry picking model of search.

SEARCH INTERFACES TODAY

At the heart of the typical search session is a cycle of query specification, inspection of retrieval results, and query reformulation. As the process proceeds, the searcher learns about their topic, as well as about the available information sources. This section describes several user interface components that have become standard in search interfaces and which exhibit high usability. As these components are described, the design characteristics that they support will be underscored. Ideally, these components are integrated together to support the different parts of the process, but it is useful to discuss each separately.

Query Specification

Once a search starting point has been selected, the primary methods for a searcher to express their information need are either entering words into a search entry form or selecting links from a directory or other information organization display. For Web search engines, the query is specified in textual form. Today this is usually done by typing text on a keyboard, but in future, query specification via spoken commands will most likely become increasingly common, using mobile devices as the input medium.

Typically in Web queries today, the text is very short, consisting of one to three words. Multiword queries are often meant to be construed as a phrase, but can also consist of multiple topics. Short queries reflect the standard usage scenario in which the user "tests the waters" to see what the search engine returns in response to their short query. 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 and pursues more fine-tuned queries on that site.

This search behavior, in which a general query is used to find a promising part of the information space, and then follow hyperlinks within relevant Web sites, is a demonstration of the orienteering strategy of Web search. There is evidence that in many cases searchers would prefer to state their information need in more detail, but past experience with search engines taught them that this method does not work well, and that keyword querying combined with orienteering works better.

Query Specification Interfaces

The standard interface for a textual query is a search box entry form, in which the user types a query, activated by hitting the return key on the keyboard or selecting a button associated with the form. Studies suggest a relationship between query length and the width of the entry form; results find that either small forms

discourage long queries or wide forms encourage longer queries. Some entry forms are divided into multiple components, allowing for a more general free text query followed by a form that filters the query in some way. For instance, at yelp.com, the user enters a general query into the first entry form and refines the search by location in the second form (see Figure 2.1).

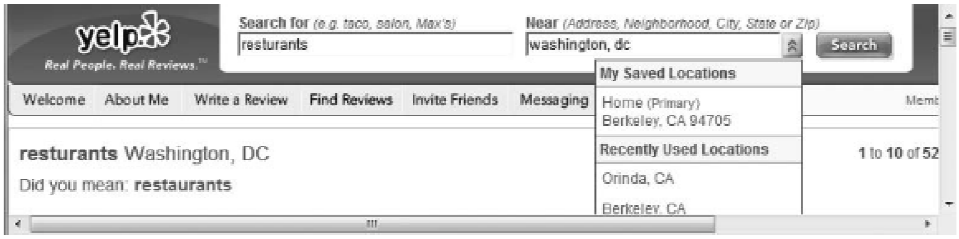


Figure 2.1: Query form, from yelp.com, illustrating support to facilitate structured queries, and stored information about past queries.

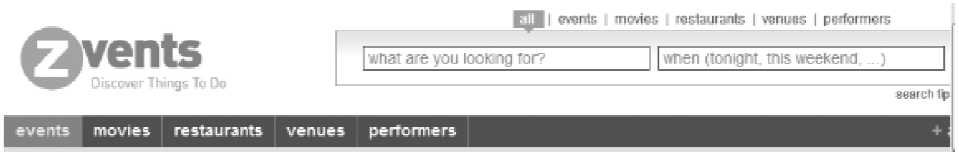


Figure 2.2: Query form, from zvents.com, illustrating greyed-out text that provides hints about what kind of information to type, directly in the form.

Forms allow for selecting information that has been used in the past; sometimes this information is structured and allows for setting parameters to be used in future. For instance, the yelp.com form shows the user's home location (if it has been indicated in the past) along with recently specified locations and the option to add additional locations.

An increasingly common strategy within the search form is to show hints about what kind of information should be entered into each form via greyed-out text. For instance, in zvents.com search (see Figure 2.2), the first box is labeled "what are you looking for?" while the second box is labeled "when (tonight, this weekend, ...)". When the user places the cursor into the entry form, the grey text disappears, and the user can type in their query terms. This example also illustrates

specialized input types that some search engines are supporting today. For instance, the zvents.com site recognizes that words like "tomorrow" are time-sensitive, and interprets them in the expected manner. It also allows flexibility in the syntax of more formal specification of dates. So searching for "comedy" on "wed" automatically computes the date for the nearest future Wednesday.

This is an example of designing the interface to reflect how people think, rather than making how the user thinks conform to the brittle, literal expectations of typical programs. (This approach to "loose" query specifi cation works better for "casual" interfaces in which getting the date right is not critical to the use of the system; casual date specification when filling out a tax form is not acceptable, as the cost of error is too high.)

An innovation that has greatly improved query specification is the inclusion of a dynamically generated list of query suggestions, shown in real time as the user types the query. This is referred to variously as auto-complete, auto-suggest, and dynamic query suggestions. A large log study found that users clicked on dynamic suggestions in the Yahoo Search Assist tool about one third of the time they were presented . This topic is covered in detail for the case of Web search engines. Often the suggestions shown are those whose prefix matches the characters typed so far, but in some cases, suggestions are shown that only have interior letters matching. If the user types a multiple word query, suggestions may be shown that are synonyms of what has been typed so far, but which do not contain lexical matches.

To exemplify, Netflix.com both describes what is wanted in gray and then shows hits via a dropdown box. In dynamic query suggestion interfaces, the display of the matches varies. Some interfaces color the suggestions according to category information. In most cases, the user must move the mouse down to the desired suggestion in order to select it, at which point the suggestion is used to fill the query box. In some cases, the query is then run immediately; in others, the user must hit the Return key or click the Search button. The suggestions can be derived from several sources. In some cases, the list is taken from the user's own query

history, in other cases, it is based on popular queries issued by other users. The list can be derived from a set of metadata that a Web site's designer considers important, such as a list of known diseases or gene names for a search over pharmacological literature (see Figure 2.3), a list of product names when searching within an e-commerce site, or a list of known film names when searching a movie site.

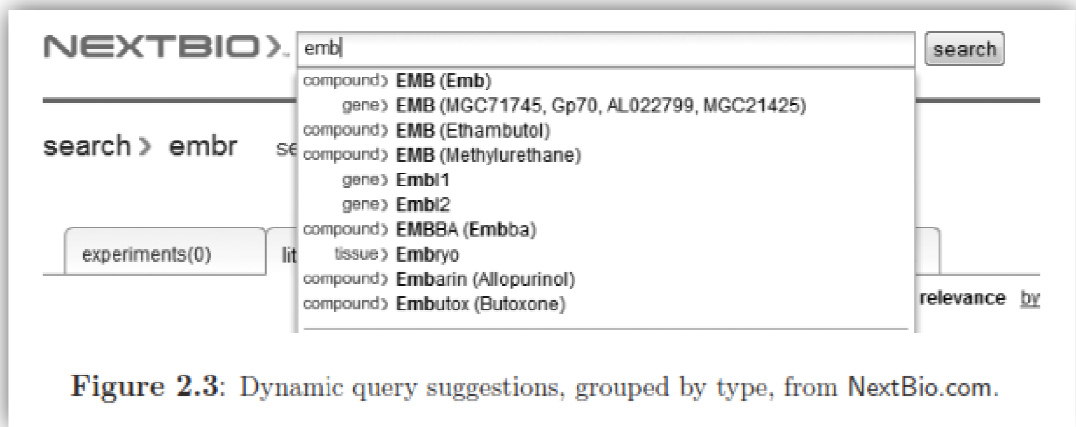


Figure 2.3: Dynamic query suggestions, grouped by type, from NextBio.com.

The suggestions can also be derived from all of the text contained within a Web site. Another form of query specification consists of choosing from a display of information, typically in the form of hyperlinks or saved bookmarks. In some cases, the action of selecting a link produces more links for further navigation, in addition to results listings. This kind of query specification is discussed in more detail in the section on organizing search results below.

Query Reformulation

After a query is specified and results have been produced, a number of tools exist to help the user reformulate their query, or take their information seeking process in a new direction. Analysis of search engine logs shows that reformulation is a common activity; one study found that more than 50% of searchers modified at least one query during a session, with nearly a third of these involving three or more queries.

Organizing Search Results

Searchers often express a desire for a user interface that organizes search results into meaningful groups to help understand the results and decide what to do next. A longitudinal study in which users were provided with the ability to group search results found that users changed their search habits in response to having the grouping mechanism available. Currently two methods for grouping search results are popular: category systems, especially faceted categories, and clustering. Both are described in more detail in this section, and their usability is compared.

A category system is a set of meaningful labels organized in such a way as to reflect the concepts relevant to a domain. They are usually created manually, although assignment of documents to categories can be automated to a certain degree of accuracy. Good category systems have the characteristics of being coherent and (relatively) complete, and their structure is predictable and consistent across search results for an information collection.

VISUALIZATION IN SEARCH INTERFACES.

Text as a representation is highly effective for conveying abstract information, but reading and even scanning text is a cognitively taxing activity, and must be done in a linear fashion. By contrast, images can be scanned quickly and the visual system perceives information in parallel. People are highly attuned to images and visual information, and pictures and graphics can be captivating and appealing. A visual representation can communicate some kinds of information much more rapidly and effectively than any other method. Consider the difference between a written description of a person's face and a photograph of it, or the difference between a table of numbers containing a correlation and a scatter plot showing the same information.

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

As noted above, 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, in order to make it more understandable. A common approach is to show Venn diagrams visually; Hertzum and Frokjaer [755] found that a simple Venn diagram representation produced more accurate results than Boolean syntax. A more flexible version of this idea was seen in the VQuery system (see Figure 2.15). Each query term is represented by a circle or oval, and the intersection among circles indicates ANDing (conjoining) of terms. VQuery represented disjunction by sets of circles within an active area of the canvas, and negation by deselecting a circle with in the active area.

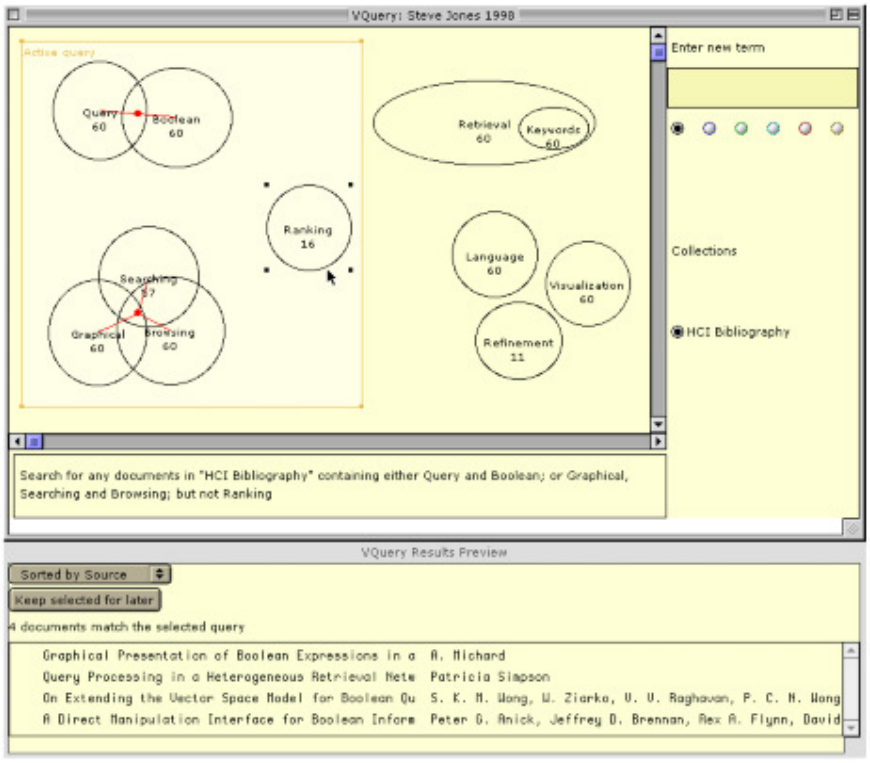


Figure 2.15: The VQuery [851] Venn Diagram interface for Boolean query specification.

One problem with Boolean queries is that they can easily end up with empty results or too many results. To remedy this, the filter-flow visualization allows users to lay out the different components of the query, and show via a graphical flow how many hits would result after each operator is applied . Other visual representations of Boolean queries include lining up blocks vertically and horizontally and representing components of queries as overlapping "magic" lenses.

Visualizing Query Terms within Retrieval Results

As discussed above, understanding the role played by the query terms within the retrieved documents can help with the assessment of relevance. In standard search results listings, summary sentences are often selected that contain query terms, and the occurrence of these terms are highlighted or boldfaced where they appear in the title, summary, and URL. Highlighting of this kind has been shown to be effective from a usability perspective.

Experimental visualizations have been designed that make this relationship more explicit. One of the best known is the TileBars interface, in which documents are shown as horizontal glyphs with the locations of the query term hits marked along the glyph (see Figure 2.16).

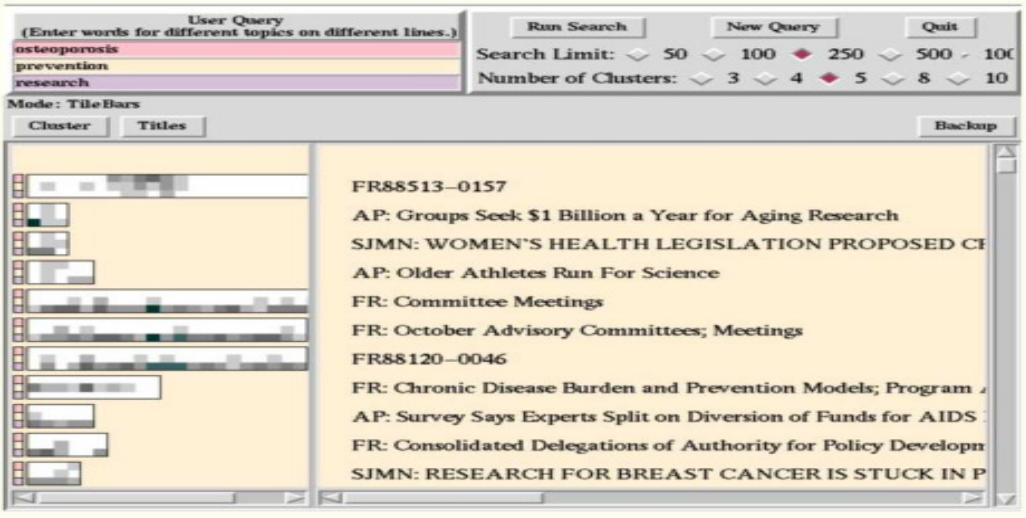


Figure 2.16: The TileBars visualization of query term hits within retrieved documents, from [732].

The user is encouraged to break the query into its different facets, with one concept per line, and then the horizontal rows within each document's

representation show the frequency of occurrence of query terms within each topic. Longer documents are divided into subtopic segments, either using paragraph or section breaks, or an automated discourse segmentation technique called TextTiling. Grayscale implies the frequency of the query term occurrences. The visualization shows where the discussion of the different query topics overlaps within the document.

Visualizing Relationships Among Words and Documents

Numerous visualization developers have proposed variations on the idea of placing words and documents on a two-dimensional canvas, where proximity of glyphs represents semantic relationships among the terms or documents. An early version of this idea is seen in the VIBE interface, where queries are laidout on a plane, and documents that contain combinations of the queries are placed midway between the icons representing those terms (see Figure 2.19). A more modern version of this idea is seen in the Aduna Autofocus product, and the Lyberworld project presented a 3D version of the ideas behind VIBE.

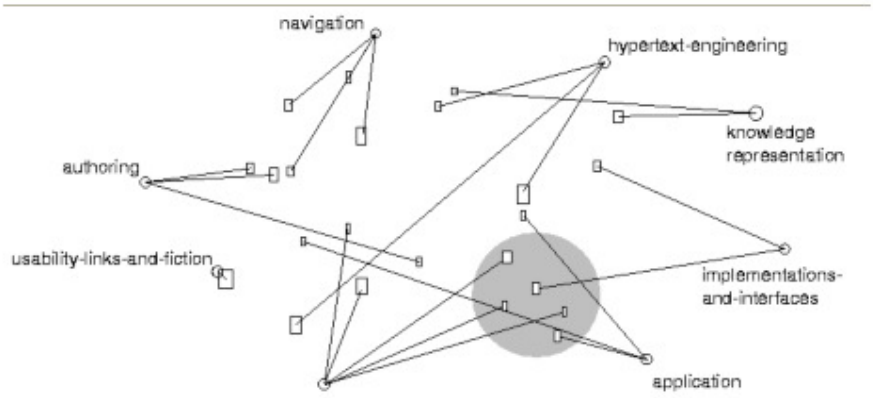


Figure 2.19: The VIBE display, in which query terms are laid out in a 2D space, and documents are arranged according to which subset of the text they share, from [1228].

Another variation of this idea is to map documents or words from a very high dimensional term space down into a two-dimensional plane, and show where the documents or words fall within that plane, using 2D or 3D. This variation on clustering can be done to documents retrieved as a result of a query, or documents that match a query can be highlighted within a preprocessed set of documents.

Visualization for Text Mining

The subsections above show that usability results for visualization in search are not particularly strong. It seems in fact that visualization is better used for purposes of analysis and exploration of textual data. Most users of search systems are not interested in seeing how words are distributed across documents, or in viewing the most common words within a collection, but these are interesting activities for computational linguists, analysts, and curious word enthusiasts. Visualizations such as the Word Tree show a piece of a text concordance, allowing the user to view which words and phrases commonly precede or follow a given word (see Figure 2.22), or the Name Voyager explorer, which shows frequencies of baby names for U.S. children across time (see Figure 2.23 on page 51).

Figure 2.22: The Word Tree visualization, on Martin Luther King's *I have a dream* speech, from The word tree, an interactive visual concordance, *IEEE Transactions on Visualization and Computer Graphics*, 14(6), pp. 1221-8 (Wattenberg, M. and Fernanda, B., 2008), ©2008 IEEE [1669].

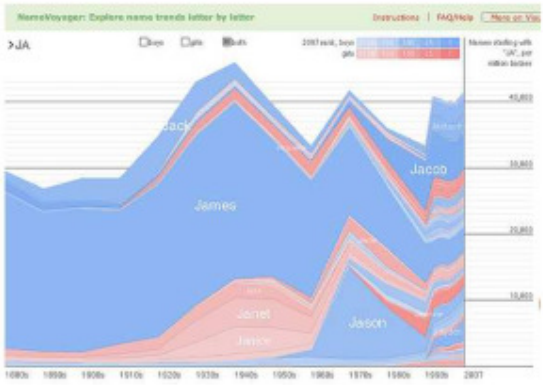


Figure 2.23: A visualization of the relative popularity of baby names over time, with names beginning with the letters JA, from babynamewizard.com.

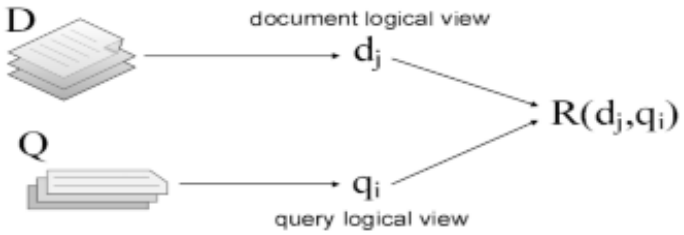
UNIT II
MODELING AND RETRIEVAL EVALUATION

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.

Modeling

- ✓ Modeling in IR is a complex process aimed at producing a ranking function.
- ✓ Ranking function: a function that assigns scores to documents with regard to a given query.
- ✓ This process consists of two main tasks:
  - The conception of a logical framework for representing documents and queries
  - The definition of a ranking function that allows quantifying the similarities among documents and queries
- ✓ IR systems usually adopt index terms to index and retrieve documents

IR Model Definition:



An IR model is a quadruple [D, Q, F, R(qi, dj)] where
1. D is a set of logical views for the documents in the collection
2. Q is a set of logical views for the user queries
3. F is a framework for modeling documents and queries
4. R(qi, dj) is a ranking function

Types of Information Retrieval (IR) Model

✓ An information model (IR) model can be classified into the following three models

- Classical IR Model
- Non-Classical IR Model
- Alternative IR Model

Classical IR Model

It is the simplest and easy to implement IR model. This model is based on mathematical knowledge that was easily recognized and understood as well. Boolean, Vector and Probabilistic are the three classical IR models.

Non-Classical IR Model

It is completely opposite to classical IR model. Such kind of IR models are based on principles other than similarity, probability, Boolean operations. Information logic model, situation theory model and interaction models are the examples of non-classical IR model.

Alternative IR Model

It is the enhancement of classical IR model making use of some specific techniques from some other fields. Cluster model, fuzzy model and latent semantic indexing (LSI) models are the example of alternative IR model.

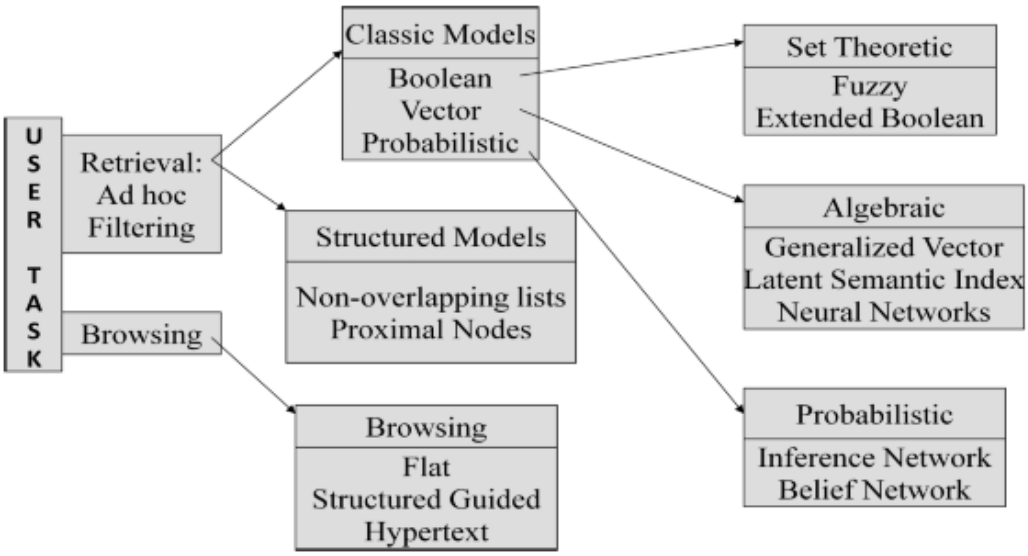
Classic IR model:

✓ Each document is described by a set of representative keywords called index terms.

✓ Assign a numerical weights to distinct relevance between index terms.

✓ Three classic models:

- Boolean,
- Vector,
- Probabilistic



BOOLEAN MODEL

The Boolean retrieval model is a model for information retrieval in which we can pose any query which is in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR, and NOT. The model views each document as just a set of words. Based on a binary decision criterion without any notion of a grading scale. Boolean expressions have precise semantics.

It is the oldest information retrieval (IR) model. The model is based on set theory and the Boolean algebra, where documents are sets of terms and queries are Boolean expressions on terms. The Boolean model can be defined as –

- D – A set of words, i.e., the indexing terms present in a document. Here, each term is either present (1) or absent (0).
- Q – A Boolean expression, where terms are the index terms and operators are logical products – AND, logical sum – OR and logical difference – NOT
- F – Boolean algebra over sets of terms as well as over sets of documents

If we talk about the relevance feedback, then in Boolean IR model the Relevance prediction can be defined as follows –

- R – A document is predicted as relevant to the query expression if and only if it satisfies the query expression as –

$$((text \vee information) \wedge rerieval \wedge \tilde{} \; theory)$$

We can explain this model by a query term as an unambiguous definition of a set of documents. For example, the query term *"economic"* defines the set of documents that are indexed with the term *"economic"*.

Now, what would be the result after combining terms with Boolean AND Operator? It will define a document set that is smaller than or equal to the document sets of any of the single terms. For example, the query with terms *"social"* and *"economic"* will produce the documents set of documents that are indexed with both the terms. In other words, document set with the intersection of both the sets.

Now, what would be the result after combining terms with Boolean OR operator? It will define a document set that is bigger than or equal to the document sets of any of the single terms. For example, the query with terms *"social"* or *"economic"* will produce the documents set of documents that are indexed with either the term *"social"* or *"economic"*. In other words, document set with the union of both the sets.

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 of the Boolean Mode

The advantages of the Boolean model are as follows –

- The simplest model, which is based on sets.
- Easy to understand and implement.
- It only retrieves exact matches
- It gives the user, a sense of control over the system.

Disadvantages of the Boolean Model

The disadvantages of the Boolean model are as follows –

- The model's similarity function is Boolean. Hence, there would be no partial matches. This can be annoying for the users.
- In this model, the Boolean operator usage has much more influence than a critical word.

- The query language is expressive, but it is complicated too.
- No ranking for retrieved documents.

TF-IDF (Term Frequency/Inverse Document Frequency)

Term Frequency (tf_{ij})

It may be defined as the number of occurrences of w_i in d_j. The information that is captured by term frequency is how salient a word is within the given document or in other words we can say that the higher the term frequency the more that word is a good description of the content of that document.

Document Frequency (df_i)

It may be defined as the total number of documents in the collection in which w_i occurs. It is an indicator of informativeness. Semantically focused words will occur several times in the document unlike the semantically unfocused words.

Assign to each term in a document a weight for that term, that depends on the number of occurrences of the term in the document. We would like to compute a score between a query term t and a document d, based on the weight of t in d. The simplest approach is to 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 $tf_{t,d}$ with the subscripts denoting the term and the document in order.

Inverse document frequency

This is another form of document frequency weighting and often called idf weighting or inverse document frequency weighting. The important point of idf weighting is that the term's scarcity across the collection is a measure of its importance and importance is inversely proportional to frequency of occurrence.

Raw term frequency as above suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy on a query. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to

have the term auto in almost every document. A mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high collection frequency, defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the tf weight of a term by a factor that grows with its collection frequency.

Mathematically,

$$idf_t = log\left(1 + \frac{N}{n_t}\right)$$

$$idf_t = log\left(\frac{N - n_t}{n_t}\right)$$

Here,

N = documents in the collection

n_t = documents containing term *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

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t.$$

In other words, $\text{tf-idf}_{t,d}$ assigns to term *t* a weight in document *d* that is

1. highest when *t* occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

VECTOR MODEL

Assign non-binary weights to index terms in queries and in documents. Compute the similarity between documents and query. More precise than Boolean model.

Due to the disadvantages of the Boolean model, Gerard Salton and his colleagues suggested a model, which is based on Luhn's similarity criterion. The similarity criterion formulated by Luhn states, "the more two representations agreed in given elements and their distribution, the higher would be the probability of their representing similar information."

Consider the following important points to understand more about the Vector Space Model –

- The index representations (documents) and the queries are considered as vectors embedded in a high dimensional Euclidean space.
- The similarity measure of a document vector to a query vector is usually the cosine of the angle between them.

Binary values are changed into count values , later it is represented as weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	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
. . .						

Each document is represented as a binary vector

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
. . .						

Each document is now represented as a count vector

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTONY	5.255	3.18	0	0	0	0.35
BRUTUS	1.21	6.1	0	1	0	0
CAESAR	8.59	2.54	0	1.51	0.25	1
CALPURNIA	0	1.54	0	0	0	0
CLEOPATRA	2.85	0	0	0	0	0
MERCY	1.51	0	1.9	0.12	5.25	0.88
WORSER	1.37	0	0.11	4.15	0.25	1.95

Document represented by tf-idf weight vector

Binary →Count → Weight Matrix

Documents as Vectors

Each document is now represented as a real-valued vector of tf-idf weights ∈ R|V|.
So we have a |V|-dimensional real-valued vector space. Terms are axes of the space.
Documents are points or vectors in this space.
Very high-dimensional: tens of millions of dimensions when you apply this to web search engines. Each vector is very sparse - most entries are zero.

To represent the document as vector , We can consider
each document as → a vector
each term of doc → one component of vector
weight for each component is given by TFIDF(t, d) = TF(t, d) x IDF(t)

Queries as Vectors

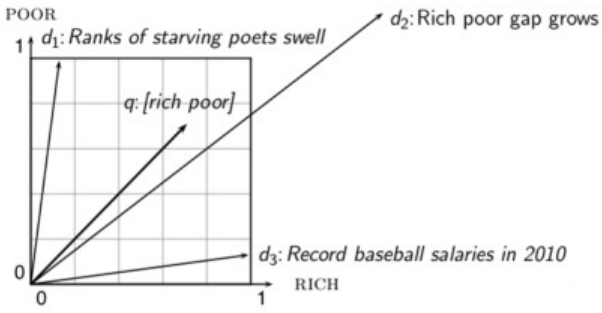
Key idea 1: Represent queries as vectors in same space
Key idea 2: Rank documents according to proximity to query in this space
proximity = similarity of vectors
proximity ≈ inverse of distance
Get away from Boolean model , Rank more relevant documents higher than less relevant documents

Formalizing Vector Space Proximity

We start by calculating Euclidean distance:

$$\|q - d_n\|_2$$

Euclidean distance is a bad idea . . . because Euclidean distance is large for vectors of different lengths



Cosine Similarity Measure Formula

Cosine is a normalized dot product, which can be calculated with the help of the following formula –

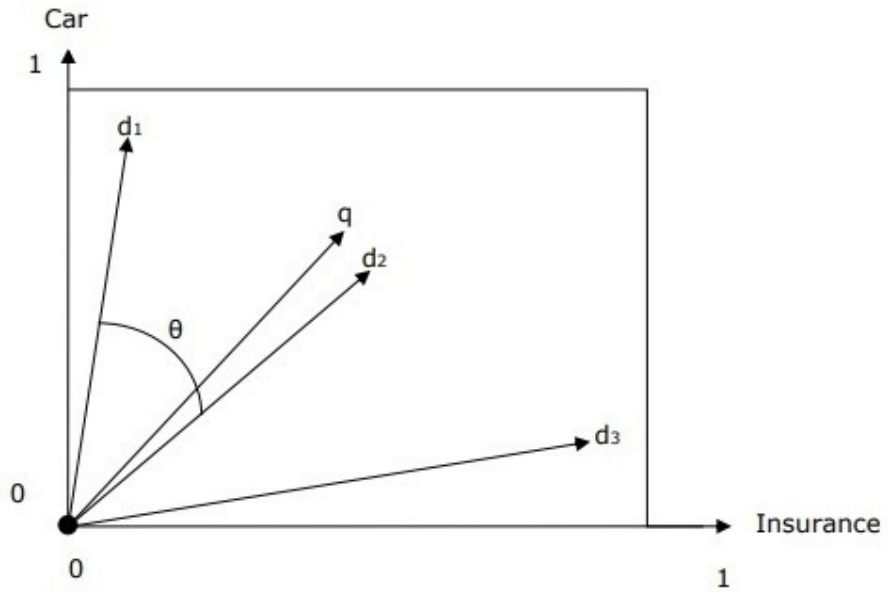
$$Score(\vec{d}\vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m}(d_k)^2} \cdot \sqrt{\sum_{k=1}^{m} m(q_k)^2}}$$

$$Score(\vec{d}\vec{q}) = 1 \; when \; d = q$$

$$Score(\vec{d}\vec{q}) = 0 \; when \; d \; and \; q \; share \; no \; items$$

Vector Space Representation with Query and Document

The query and documents are represented by a two-dimensional vector space. The terms are ***car*** and ***insurance***. There is one query and three documents in the vector space.



The top ranked document in response to the terms car and insurance will be the document $\mathbf{d_2}$ because the angle between **q** and $\mathbf{d_2}$ is the smallest. The reason behind this is that both the concepts car and insurance are salient in d_2 and hence have the high weights. On the other side, $\mathbf{d_1}$ and $\mathbf{d_3}$ also mention both the terms but in each case, one of them is not a centrally important term in the document.

PROBABILISTIC MODEL

The probabilistic model tries to estimate the probability that the user will find the document dj relevant with ratio

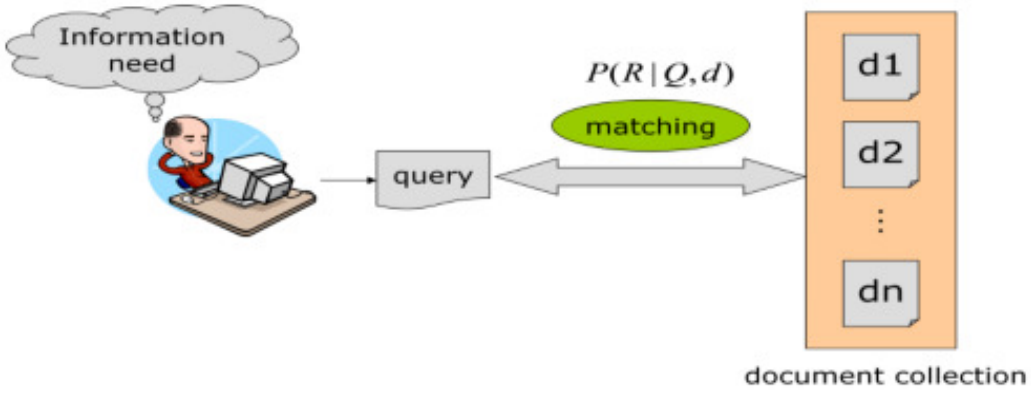
P(dj relevant to q) / P(dj non relevant to q)

Given a user query q, and the ideal answer set R of the relevant documents, the problem is to specify the properties for this set. Assumption (probabilistic principle): the probability of relevance depends on the query and document representations only; ideal answer set R should maximize the overall probability of relevance.

Given a query q, there exists a subset of the documents R which are relevant to q But membership of R is uncertain (not sure) , A Probabilistic retrieval model ranks documents in decreasing order of probability of relevance to the information need: P(R | q,d I). Users gives with ***information needs***, which they translate into ***query representations***. Similarly, there are ***documents***, which are converted into ***document representations*** . Given only a query, an IR system has an uncertain understanding of the information need. So IR is an uncertain process , Because,

- Information need to query
- Documents to index terms
- Query terms and index terms mismatch

Probability theory provides a principled foundation for such reasoning under uncertainty. This model provides how likely a document is relevant to an information need.



Document can be relevant and non relevant document, we can estimate the probability of a term t appearing in a relevant document P(t | R=1) . Probabilistic methods are one of the oldest but also one of the currently hottest topics in IR .

Basic probability theory

For events A , the probability of the event lies between 0= P(A) = 1 , For 2 events A and B

- Joint probability P(***A***, ***B***) of both events occurring
- Conditional probability P(***A***|***B***) of event ***A*** occurring given that event B has occurred
- Chain rule gives fundamental relationship between joint and conditional probabilities:

$$P(A,B) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

Similarly for the complement of an event P(A,B) :

$$P(\overline{A},B) = P(B|\overline{A})P(\overline{A})$$

Partition rule: if B can be divided into an exhaustive set of disjoint sub cases, then P(B) is the sum of the probabilities of the sub cases. A special case of this rule gives:

$$P(B) = P(A,B) + P(\overline{A},B)$$

- Bayes' Rule for inverting conditional probabilities:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \left[\frac{P(B|A)}{\sum_{X\in\{A,\overline{A}\}} P(B|X)P(X)}\right] P(A)$$

  Can be thought of as a way of updating probabilities:
  - Start off with prior probability $P(A)$ (initial estimate of how likely event A is in the absence of any other information)
  - Derive a posterior probability $P(A|B)$ after having seen the evidence B, based on the likelihood of B occurring in the two cases that A does or does not hold
- Odds of an event (is the ratio of the probability of an event to the probability of its complement.) provide a kind of multiplier for how probabilities change:

  Odds: $O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1-P(A)}$

LATENT SEMANTIC INDEXING MODEL

Latent semantic indexing (**LSI**) is an indexing and retrieval method that uses a mathematical technique called singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. A key feature of LSI is its ability to extract the conceptual content of a body of text by establishing associations between those terms that occur in similar contexts.

- Classic IR might lead to poor retrieval due to:
  - unrelated documents might be included in the answer set
  - relevant documents that do not contain at least one index term are not retrieved
  - **Reasoning**: retrieval based on index terms is vague and noisy
- The user information need is more related to concepts and ideas than to index terms
- A document that shares concepts with another document known to be relevant might be of interest

- The idea here is to map documents and queries into a dimensional space composed of concepts
- Let
  - t: total number of index terms
  - N: number of documents
  - $\mathbf{M} = [m_{ij}]$: term-document matrix $t \times N$
- To each element of $\mathbf{M}$ is assigned a weight $w_{i,j}$ associated with the term-document pair $[k_i, d_j]$
  - The weight $w_{i,j}$ can be based on a *tf-idf* weighting scheme

- The matrix $\mathbf{M} = [m_{ij}]$ can be decomposed into three components using singular value decomposition

$$\mathbf{M} = \mathbf{K} \cdot \mathbf{S} \cdot \mathbf{D}^T$$

- were
  - $\mathbf{K}$ is the matrix of eigenvectors derived from $\mathbf{C} = \mathbf{M} \cdot \mathbf{M}^T$
  - $\mathbf{D}^T$ is the matrix of eigenvectors derived from $\mathbf{M}^T \cdot \mathbf{M}$
  - $\mathbf{S}$ is an $r \times r$ diagonal matrix of singular values where $r = \min(t, N)$ is the rank of $\mathbf{M}$

Computing an Example

- Let $\mathbf{M}^T = [m_{ij}]$ be given by

	K_1	K_2	K_3	$q \bullet d_j$
d_1	2	0	1	5
d_2	1	0	0	1
d_3	0	1	3	11
d_4	2	0	0	2
d_5	1	2	4	17
d_6	1	2	0	5
d_7	0	5	0	10
q	1	2	3	

- Compute the matrices $\mathbf{K}$, $\mathbf{S}$, and $\mathbf{D}^t$

- In the matrix $\mathbf{S}$, consider that only the s largest singular values are selected
- Keep the corresponding columns in $\mathbf{K}$ and $\mathbf{D}^T$
- The resultant matrix is called $\mathbf{M}_s$ and is given by

$$\mathbf{M}_s = \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T$$

- where s, $s < r$, is the dimensionality of a reduced concept space
- The parameter s should be
  - large enough to allow fitting the characteristics of the data
  - small enough to filter out the non-relevant representational details

Latent Ranking

- The relationship between any two documents in s can be obtained from the $\mathbf{M}_s^T \cdot \mathbf{M}_s$ matrix given by

$$\begin{aligned}
\mathbf{M}_s^T \cdot \mathbf{M}_s &= (\mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T)^T \cdot \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= \mathbf{D}_s \cdot \mathbf{S}_s \cdot \mathbf{K}_s^T \cdot \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= \mathbf{D}_s \cdot \mathbf{S}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= (\mathbf{D}_s \cdot \mathbf{S}_s) \cdot (\mathbf{D}_s \cdot \mathbf{S}_s)^T
\end{aligned}$$

- In the above matrix, the (i, j) element quantifies the relationship between documents d_i and d_j

- The user query can be modelled as a pseudo-document in the original $\mathbf{M}$ matrix
- Assume the query is modelled as the document numbered 0 in the $\mathbf{M}$ matrix
- The matrix $\mathbf{M}_s^T \cdot \mathbf{M}_s$ quantifies the relationship between any two documents in the reduced concept space
- The first row of this matrix provides the rank of all the documents with regard to the user query

Problems with Lexical Semantics

- Ambiguity and association in natural language
  - **Polysemy**: Words often have a **multitude of meanings** and different types of usage (more severe in very heterogeneous collections).
  - The vector space model is unable to discriminate between different meanings of the same word.

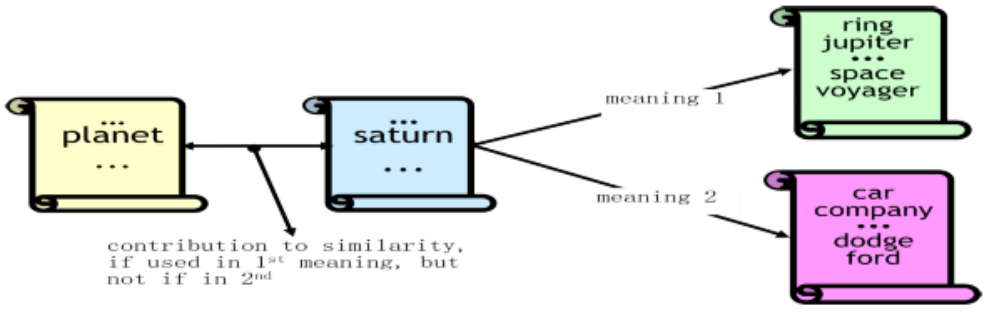
$$\mathrm{sim}_{\mathrm{true}}(d, q) < \cos(\angle(\vec{d}, \vec{q}))$$

  - Synonymy: Different terms may have an identical or a similar meaning (weaker: words indicating the same topic).
  - No associations between words are made in the vector space representation.

$$\mathrm{sim}_{\mathrm{true}}(d, q) > \cos(\angle(\vec{d}, \vec{q}))$$

  Polysemy and Context

- Document similarity on single word level: polysemy and context
- LSI Perform a **low-rank approximation** of **document-term matrix**) .



In LSI

- Map documents (and terms) to a **low-dimensional** representation.
- Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
- Compute document similarity based on the **inner product** in this **latent semantic space**
- We will decompose the term-document matrix into a product of matrices.
- The particular decomposition we'll use: singular value decomposition (SVD).
- SVD: $C = U\Sigma V^T$ (where C = term-document matrix)
- We will then use the SVD to compute a new, improved term-document matrix C'.
- We'll get better similarity values out of C' (compared to C).
- Using SVD for this purpose is called latent semantic indexing or LSI.

NEURAL NETWORK MODEL

Neural Network Definition

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. (Neural networks can also extract features that are fed to other algorithms for clustering and classification; so you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

Neural Network Model

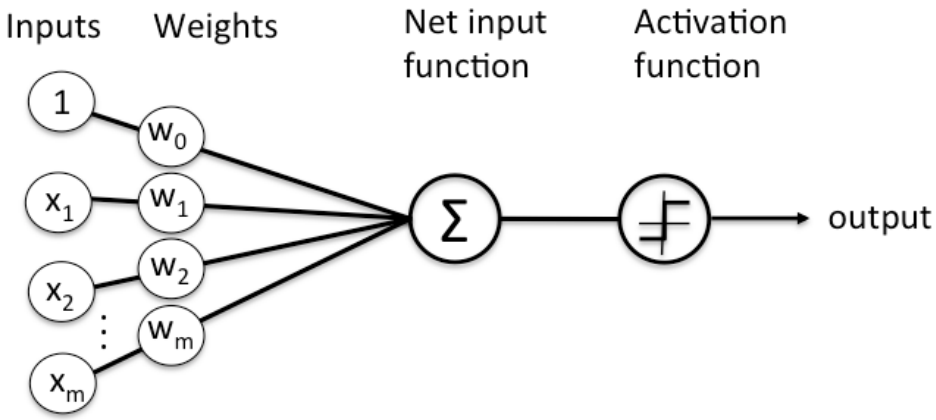
- The human brain is composed of billions of neurons
- Each neuron can be viewed as a small processing unit
- A neuron is stimulated by input signals and emits output signals in reaction
- A chain reaction of propagating signals is called a **spread activation process**
- As a result of spread activation, the brain might command the body to take physical reactions

- A neural network is an oversimplified representation of the neuron interconnections in the human brain:
  - nodes are processing units
  - edges are synaptic connections
  - the strength of a propagating signal is modelled by a weight assigned to each edge
  - the state of a node is defined by its **activation level**
  - depending on its activation level, a node might issue an output signal

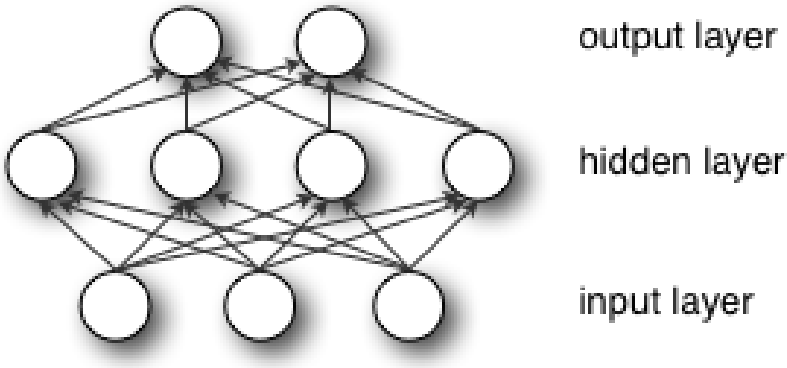
Neural Network Elements

Deep learning is the name we use for "stacked neural networks"; that is, networks composed of several layers. The layers are made of *nodes*. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regard to the task the algorithm is trying to learn; e.g. which input is most helpful is classifying data without error? These input-weight products are summed and then the sum is passed

through a node's so-called activation function, to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome, say, an act of classification. If the signals passes through, the neuron has been "activated." Here's a diagram of what one node might look like.



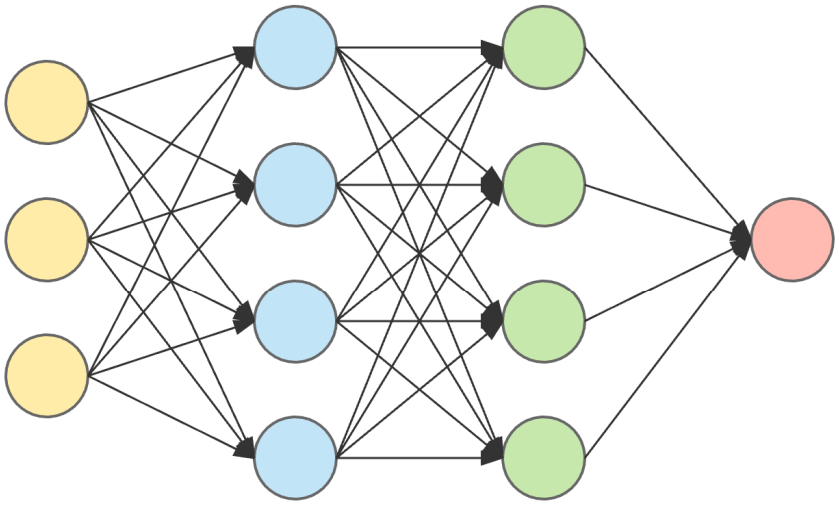
A node layer is a row of those neuron-like switches that turn on or off as the input is fed through the net. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving your data.



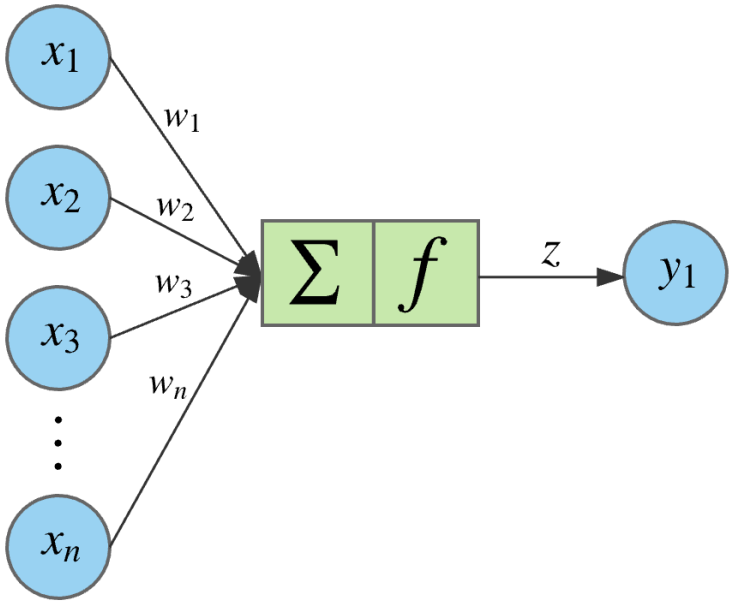
Pairing the model's adjustable weights with input features is how we assign significance to those features with regard to how the neural network classifies and clusters input.

What is an Artificial Neural Network Model?

A multi-layer, fully-connected neural network containing an input layer, hidden layers, and an output layer is called an artificial neural network or ANN. The image below represents an ANN.



If you see carefully, you will notice that each node in one layer is connected to every node in the layer next to it. As you increase the number of hidden layers, the network becomes deeper. Let's see what an individual node in the output or hidden layer looks like.



As you can see, the node gets many inputs. It sums up all the weights and passes it on as output, via a non-linear activation function. This output of the node becomes the input of the node in the next layer.

Glossary of Artificial Neural Network Model

Let's look at the basic terms you should know when it comes to an artificial neural network model.

Inputs

The data first fed into the neural network from the source is called the input. Its goal is to give the network data to make a decision or prediction about the information fed into it. The neural network model usually accepts real value sets of inputs and it should be fed into a neuron in the input layer.

Training set

The inputs for which you already know the correct outputs are called training sets. These are used to help the neural network get trained and memorize the result for the given input set.

Outputs

Every neural network generates an output as a prediction or decision about the data fed into it. This output is in the form of real values set or Boolean decisions. Only of the neurons in the output layer generates the output value.

Neuron

Also known as a perceptron, a neuron is the basic unit of a neural network. It accepts an input value and generates an output based on it. As discussed before, every neuron receives a part of the input and passes it through the non-linear activation function to the node in the next layer. These activation functions can be TanH, sigmoid, or ReLu. The non-linear feature of these functions helps to train the network.

Weight space

Every neuron has a numeric weight. When it delivers input to another note, its weight is totaled with the others to generate an output. By making small changes to

these weights are how neural networks are trained. The fine-tuning of weights helps determine the correct set of weights and biases that would generate the best outcome. This is where back propagation comes in.

RETRIEVAL EVALUATION

Introduction

- To evaluate an IR system is to measure how well the system meets the information needs of the users
  - This is troublesome, given that a same result set might be interpreted differently by distinct users
  - To deal with this problem, some metrics have been defined that, on average, have a correlation with the preferences of a group of users
- Without proper *retrieval evaluation*, one cannot
  - determine how well the IR system is performing
  - compare the performance of the IR system with that of other systems, objectively
- **Retrieval evaluation** is a critical and integral component of any modern IR system
- Systematic evaluation of the IR system allows answering questions such as:
  - a modification to the ranking function is proposed, should we go ahead and launch it?
  - a new probabilistic ranking function has just been devised, is it superior to the vector model and BM25 rankings?
  - for which types of queries, such as business, product, and geographic queries, a given ranking modification works best?
- Lack of evaluation prevents answering these questions and precludes fine tunning of the ranking function

- *Retrieval performance evaluation* consists of associating a quantitative metric to the results produced by an IR system
  - This metric should be directly associated with the relevance of the results to the user
  - Usually, its computation requires comparing the results produced by the system with results suggested by humans for a same set of queries

The Cranfield Paradigm

- Evaluation of IR systems is the result of early experimentation initiated in the 50's by Cyril Cleverdon
- The insights derived from these experiments provide a foundation for the evaluation of IR systems
- Back in 1952, Cleverdon took notice of a new indexing system called **Uniterm**, proposed by Mortimer Taube
  - Cleverdon thought it appealing and with Bob Thorne, a colleague, did a small test
  - He manually indexed 200 documents using Uniterm and asked Thorne to run some queries
  - This experiment put Cleverdon on a life trajectory of reliance on experimentation for evaluating indexing systems

- Cleverdon obtained a grant from the National Science Foundation to compare distinct indexing systems
- These experiments provided interesting insights, that culminated in the modern metrics of precision and recall
  - **Recall ratio:** the fraction of relevant documents retrieved
  - **Precision ration:** the fraction of documents retrieved that are relevant
- For instance, it became clear that, in practical situations, the majority of searches does not require high recall
- Instead, the vast majority of the users require just a few relevant answers

- The next step was to devise a set of experiments that would allow evaluating each indexing system in isolation more thoroughly
- The result was a **test reference collection** composed of documents, queries, and relevance judgements
  - It became known as the *Cranfield-2* collection
- The reference collection allows using the same set of documents and queries to evaluate different ranking systems
- The uniformity of this setup allows quick evaluation of new ranking functions

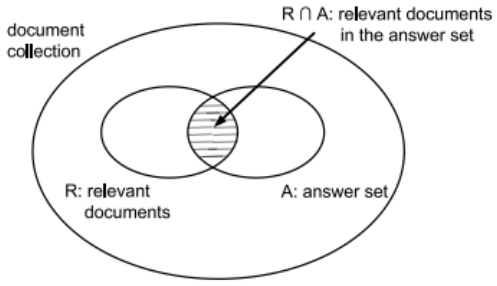
Reference Collections

- Reference collections, which are based on the foundations established by the Cranfield experiments, constitute the most used evaluation method in IR
- A reference collection is composed of:
  - A set $\mathcal{D}$ of pre-selected documents
  - A set I of information need descriptions used for testing
  - A set of relevance judgements associated with each pair $[i_m, d_j]$, $i_m \in \mathcal{I}$ and $d_j \in \mathcal{D}$
- The relevance judgement has a value of 0 if document d_j is non-relevant to i_m, and 1 otherwise
- These judgements are produced by human specialists

RETRIEVAL METRICS - PRECISION AND RECALL

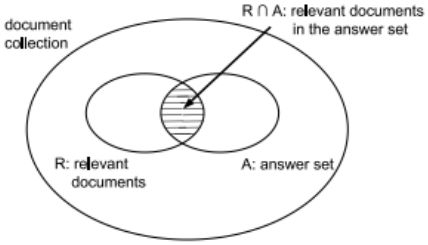
Precision and Recall

- Consider,
  - I: an information request
  - R: the set of relevant documents for I
  - A: the answer set for I, generated by an IR system
  - $R \cap A$: the intersection of the sets R and A



- The recall and precision measures are defined as follows
  - **Recall** is the fraction of the relevant documents (the set R) which has been retrieved i.e.,

$$Recall = \frac{|R \cap A|}{|R|}$$



  - **Precision** is the fraction of the retrieved documents (the set A) which is relevant i.e.,

$$Precision = \frac{|R \cap A|}{|A|}$$

- The definition of precision and recall assumes that all docs in the set A have been examined
- However, the user is not usually presented with all docs in the answer set A at once
  - User sees a ranked set of documents and examines them starting from the top
- Thus, precision and recall vary as the user proceeds with their examination of the set A
- Most appropriate then is to plot a **curve of precision versus recall**

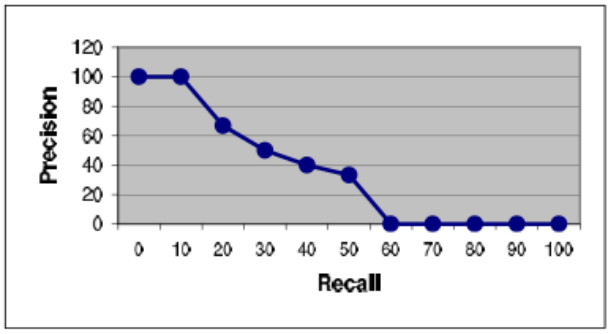
- Consider a reference collection and a set of test queries
- Let R_{q_1} be the set of relevant docs for a query q_1:
  - $R_{q_1} = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$
- Consider a new IR algorithm that yields the following answer to q_1 (relevant docs are marked with a bullet):

01. d_{123} •	06. d_9 •	11. d_{38}
02. d_{84}	07. d_{511}	12. d_{48}
03. d_{56} •	08. d_{129}	13. d_{250}
04. d_6	09. d_{187}	14. d_{113}
05. d_8	10. d_{25} •	15. d_3 •

- If we examine this ranking, we observe that
  - The document d_{123}, ranked as number 1, is relevant
    - This document corresponds to 10% of all relevant documents
    - Thus, we say that we have a precision of 100% at 10% recall
  - The document d_{56}, ranked as number 3, is the next relevant
    - At this point, two documents out of three are relevant, and two of the ten relevant documents have been seen
    - Thus, we say that we have a precision of 66.6% at 20% recall

01. d_{123} •	06. d_9 •	11. d_{38}
02. d_{84}	07. d_{511}	12. d_{48}
03. d_{56} •	08. d_{129}	13. d_{250}
04. d_6	09. d_{187}	14. d_{113}
05. d_8	10. d_{25} •	15. d_3 •

- If we proceed with our examination of the ranking generated, we can plot a curve of precision versus recall as follows:



Recall	Precision
0	100
10	100
20	66.6
30	50
40	40
50	33.3
60	0
70	0
80	0
90	0
100	0

- Consider now a second query q_2 whose set of relevant answers is given by

$$R_{q_2} = \{d_3, d_{56}, d_{129}\}$$

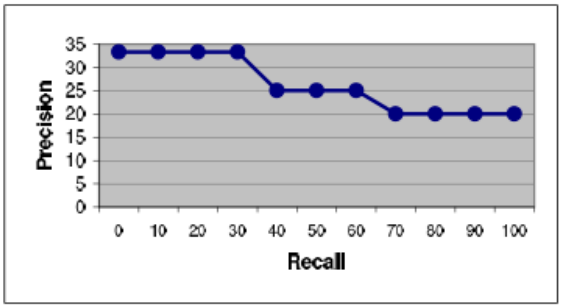
- The previous IR algorithm processes the query q_2 and returns a ranking, as follows

01. d_{425}	06. d_{615}	11. d_{193}
02. d_{87}	07. d_{512}	12. d_{715}
03. d_{56} •	08. d_{129} •	13. d_{810}
04. d_{32}	09. d_4	14. d_5
05. d_{124}	10. d_{130}	15. d_3 •

- If we examine this ranking, we observe
  - The first relevant document is d_{56}
    - It provides a recall and precision levels equal to 33.3%
  - The second relevant document is d_{129}
    - It provides a recall level of 66.6% (with precision equal to 25%)
  - The third relevant document is d_3
    - It provides a recall level of 100% (with precision equal to 20%)

01. d_{425}	06. d_{615}	11. d_{193}
02. d_{87}	07. d_{512}	12. d_{715}
03. d_{56} •	08. d_{129} •	13. d_{810}
04. d_{32}	09. d_4	14. d_5
05. d_{124}	10. d_{130}	15. d_3 •

- The precision figures at the 11 standard recall levels are interpolated as follows
- Let r_j, $j \in \{0, 1, 2, \ldots, 10\}$, be a reference to the j-th standard recall level
- Then, $\boxed{P(r_j) = max_{\forall r \mid r_j \leq r} \; P(r)}$
- In our last example, this interpolation rule yields the precision and recall figures illustrated below



Recall	Precision
0	33.3
10	33.3
20	33.3
30	33.3
40	25
50	25
60	25
70	20
80	20
90	20
100	20

Precision-Recall Appropriateness

- Precision and recall have been extensively used to evaluate the retrieval performance of IR algorithms
- However, a more careful reflection reveals problems with these two measures:
  - First, the proper estimation of maximum recall for a query requires detailed knowledge of all the documents in the collection
  - Second, in many situations the use of a single measure could be more appropriate
  - Third, recall and precision measure the effectiveness over a set of queries processed in batch mode
  - Fourth, for systems which require a weak ordering though, recall and precision might be inadequate

REFERENCE COLLECTION

Reference Collections

- With small collections one can apply the Cranfield evaluation paradigm to provide relevance assessments
- With large collections, however, not all documents can be evaluated relatively to a given information need
- The alternative is consider only the top k documents produced by various ranking algorithms for a given information need
  - This is called the **pooling method**
- The method works for reference collections of a few million documents, such as the **TREC collections**

The TREC Conferences

- TREC is an yearly promoted conference dedicated to experimentation with large test collections
- For each TREC conference, a set of experiments is designed
- The research groups that participate in the conference use these experiments to compare their retrieval systems
- As with most test collections, a TREC collection is composed of three parts:
  - the documents
  - the example information requests (called **topics**)
  - a set of relevant documents for each example information request

The Document Collections

- The main TREC collection has been growing steadily over the years
- The TREC-3 collection has roughly 2 gigabytes
- The TREC-6 collection has roughly 5.8 gigabytes
  - It is distributed in 5 CD-ROM disks of roughly 1 gigabyte of compressed text each
  - Its 5 disks were also used at the TREC-7 and TREC-8 conferences
- The *Terabyte test collection* introduced at TREC-15, also referred to as GOV2, includes 25 million Web documents crawled from sites in the ".gov" domain

The Document Collections

- TREC documents come from the following sources:

WSJ	→ *Wall Street Journal*
AP	→ Associated Press (news wire)
ZIFF	→ Computer Selects (articles), Ziff-Davis
FR	→ Federal Register
DOE	→ US DOE Publications (abstracts)
SJMN	→ *San Jose Mercury News*
PAT	→ US Patents
FT	→ *Financial Times*
CR	→ Congressional Record
FBIS	→ Foreign Broadcast Information Service
LAT	→ *LA Times*

The Document Collections

- Contents of TREC-6 disks 1 and 2

Disk	Contents	Size **Mb**	Number **Docs**	Words/Doc. **(median)**	Words/Doc. **(mean)**
1	WSJ, 1987-1989	267	98,732	245	434.0
	AP, 1989	254	84,678	446	473.9
	ZIFF	242	75,180	200	473.0
	FR, 1989	260	25,960	391	1315.9
	DOE	184	226,087	111	120.4
2	WSJ, 1990-1992	242	74,520	301	508.4
	AP, 1988	237	79,919	438	468.7
	ZIFF	175	56,920	182	451.9
	FR, 1988	209	19,860	396	1378.1

The TREC Web Collections

- A Web Retrieval track was introduced at TREC-9
  - The VLC2 collection is from an Internet Archive crawl of 1997
  - WT2g and WT10g are subsets of the VLC2 collection
  - .GOV is from a crawl of the .gov Internet done in 2002
  - .GOV2 is the result of a joint NIST/UWaterloo effort in 2004

Collection	# Docs	Avg Doc Size	Collection Size
VLC2 (WT100g)	18,571,671	5.7 KBytes	100 GBytes
WT2g	247,491	8.9 KBytes	2.1 GBytes
WT10g	1,692,096	6.2 KBytes	10 GBytes
.GOV	1,247,753	15.2 KBytes	18 GBytes
.GOV2	27 million	15 KBytes	400 GBytes

Information Requests Topics

- Each TREC collection includes a set of example **information requests**
- Each request is a description of an information need in natural language
- In the TREC nomenclature, each test information request is referred to as a **topic**

The Relevant Documents

- The set of relevant documents for each topic is obtained from a pool of possible relevant documents
  - This pool is created by taking the top K documents (usually, $K = 100$) in the rankings generated by various retrieval systems
  - The documents in the pool are then shown to human assessors who ultimately decide on the relevance of each document
- This technique of assessing relevance is called the pooling method and is based on two assumptions:
  - First, that the vast majority of the relevant documents is collected in the assembled pool
  - Second, that the documents which are not in the pool can be considered to be not relevant

USER-BASED EVALUATION

User Based Evaluation

- User preferences are affected by the characteristics of the user interface (UI)
- For instance, the users of search engines look first at the upper left corner of the results page
- Thus, changing the layout is likely to affect the assessment made by the users and their behavior
- Proper evaluation of the user interface requires going beyond the framework of the Cranfield experiments

Human Experimentation in the Lab

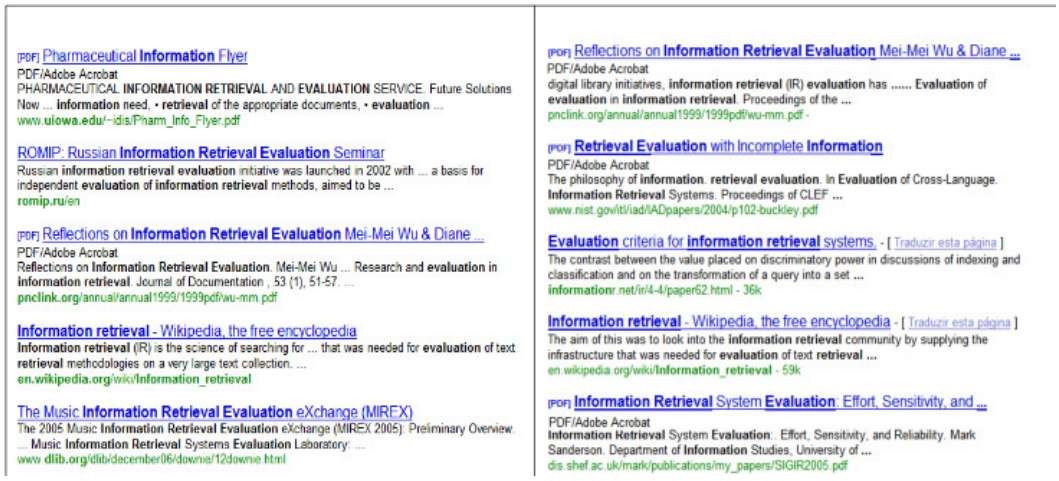
- Evaluating the impact of UIs is better accomplished in laboratories, with human subjects carefully selected
- The downside is that the experiments are costly to setup and costly to be repeated
- Further, they are limited to a small set of information needs executed by a small number of human subjects
- However, human experimentation is of value because it complements the information produced by evaluation based on reference collections

Side-by-Side Panels

- A form of evaluating two different systems is to evaluate their results side by side
- Typically, the top 10 results produced by the systems for a given query are displayed in side-by-side panels
- Presenting the results side by side allows controlling:
  - differences of opinion among subjects
  - influences on the user opinion produced by the ordering of the top results

Side-by-Side Panels

- Side by side panels for Yahoo! and Google
  - Top 5 answers produced by each search engine, with regard to the query *"information retrieval evaluation"*



[PDF] Pharmaceutical **Information** Flyer
PDF/Adobe Acrobat
PHARMACEUTICAL **INFORMATION RETRIEVAL** AND **EVALUATION** SERVICE. Future Solutions Now ... **information** need, • **retrieval** of the appropriate documents, • **evaluation** ...
www.**uiowa.edu**/~idis/Pharm_Info_Flyer.pdf

ROMIP: Russian **Information Retrieval Evaluation** Seminar
Russian **information retrieval evaluation** initiative was launched in 2002 with ... a basis for independent **evaluation** of **information retrieval** methods, aimed to be ...
romip.ru/en

[PDF] Reflections on **Information Retrieval Evaluation** Mei-Mei Wu & Diane ...
PDF/Adobe Acrobat
Reflections on **Information Retrieval Evaluation**. Mei-Mei Wu ... Research and **evaluation** in **information retrieval**. Journal of Documentation , 53 (1), 51-57. ...
pnclink.org/annual/annual1999/1999pdf/wu-mm.pdf

Information retrieval - Wikipedia, the free encyclopedia
Information retrieval (IR) is the science of searching for ... that was needed for **evaluation** of text **retrieval** methodologies on a very large text collection. ...
en.wikipedia.org/wiki/**Information_retrieval**

The Music **Information Retrieval Evaluation** eXchange (MIREX)
The 2005 Music **Information Retrieval Evaluation** eXchange (MIREX 2005): Preliminary Overview. ... Music **Information Retrieval** Systems **Evaluation** Laboratory: ...
www.**dlib.org**/dlib/december06/downie/12downie.html

[PDF] Reflections on **Information Retrieval Evaluation** Mei-Mei Wu & Diane ...
PDF/Adobe Acrobat
digital library initiatives, **information retrieval** (IR) **evaluation** has ..... **Evaluation** of **evaluation** in **information retrieval**. Proceedings of the ...
pnclink.org/annual/annual1999/1999pdf/wu-mm.pdf -

[PDF] **Retrieval Evaluation** with Incomplete **Information**
PDF/Adobe Acrobat
The philosophy of **information**. **retrieval evaluation**. In **Evaluation** of Cross-Language. **Information Retrieval** Systems. Proceedings of CLEF ...
www.nist.gov/itl/iad/IADpapers/2004/p102-buckley.pdf

Evaluation criteria for **information retrieval** systems. - [Traduzir esta página]
The contrast between the value placed on discriminatory power in discussions of indexing and classification and on the transformation of a query into a set ...
informationr.net/ir/4-4/paper62.html - 36k

Information retrieval - Wikipedia, the free encyclopedia - [Traduzir esta página]
The aim of this was to look into the **information retrieval** community by supplying the infrastructure that was needed for **evaluation** of text **retrieval** ...
en.wikipedia.org/wiki/**Information_retrieval** - 59k

[PDF] **Information Retrieval** System **Evaluation**: Effort, Sensitivity, and ...
PDF/Adobe Acrobat
Information Retrieval System **Evaluation**: Effort, Sensitivity, and Reliability. Mark Sanderson. Department of **Information** Studies, University of ...
dis.shef.ac.uk/mark/publications/my_papers/SIGIR2005.pdf

- The side-by-side experiment is simply a judgement on which side provides better results for a given query
  - By recording the interactions of the users, we can infer which of the answer sets are preferred to the query
- Side by side panels can be used for quick comparison of distinct search engines

RELEVANCE FEEDBACK AND QUERY EXPANSION

Most users find it difficult to formulate queries that are well designed for retrieval purposes Yet, most users often need to reformulate their queries to obtain the results of their interest Thus, the first query formulation should be treated as an initial attempt to retrieve relevant information. Documents initially retrieved could be analyzed for relevance and used to improve initial query

The process of query modification is commonly referred as **relevance feedback**, when the user provides information on relevant documents to a query, or **query expansion**, when information related to the query is used to expand it We refer to both of them as feedback methods Two basic approaches of feedback methods:

1.explicit feedback, in which the information for query reformulation is provided directly by the users, and

2.implicit feedback, in which the information for query reformulation is implicitly derived by the system

A Framework for Feedback Methods

Consider a set of documents D_r that are known to be relevant to the current query q. In relevance feedback, the documents in D_r are used to transform q into a modified query q_m. However, obtaining information on documents relevant to a query requires the direct interference of the user. Most users are unwilling to provide this information, particularly in the Web.

Because of this high cost, the idea of relevance feedback has been relaxed over the years Instead of asking the users for the relevant documents, we could: Look at documents they have clicked on; or Look at terms belonging to the top documents in the result set. In both cases, it is expect that the feedback cycle will produce results of higher quality.

A **feedback cycle** is composed of two basic steps:

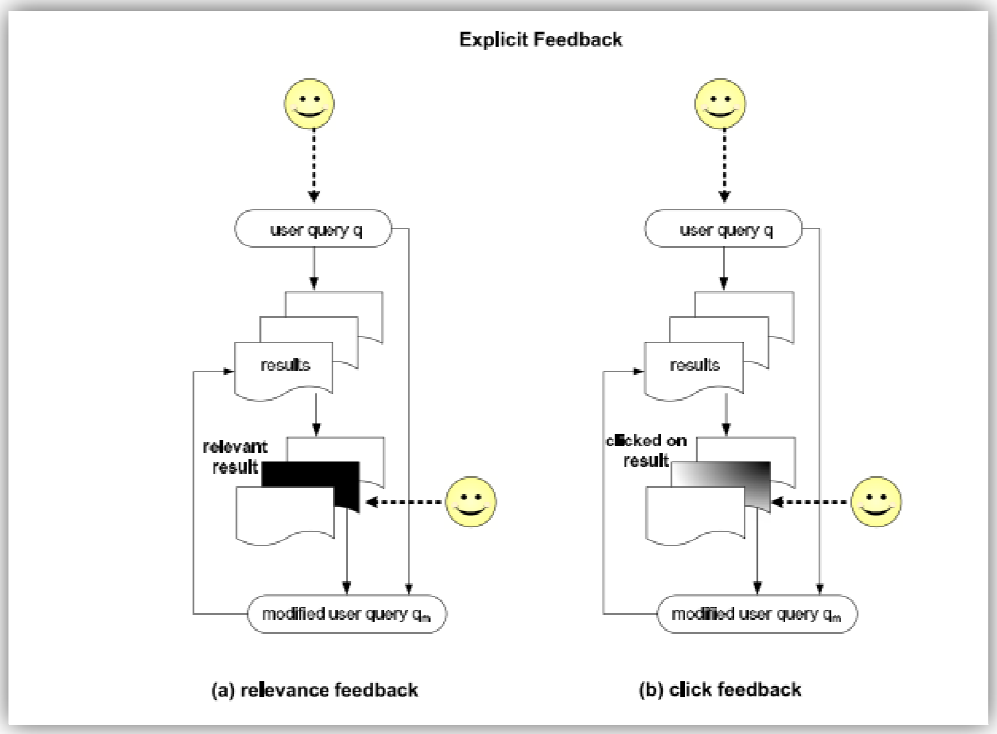
- Determine feedback information that is either related or expected to be related to the original query q and
- Determine how to transform query q to take this information effectively into account

✓ The first step can be accomplished in two distinct ways:

- Obtain the feedback information explicitly from the users
- Obtain the feedback information implicitly from the query results or from external sources such as a **thesaurus.**

EXPLICIT RELEVANCE FEEDBACK

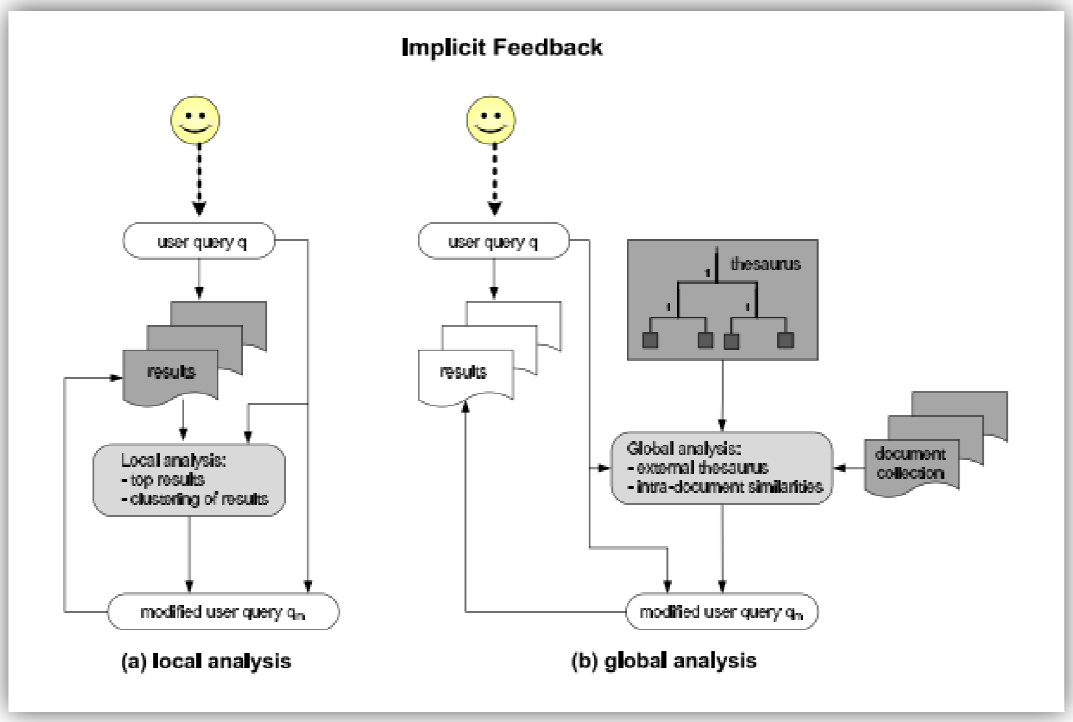
In an **explicit relevance feedback** cycle, the feedback information is provided directly by the users However, collecting feedback information is expensive and time consuming In the Web, **user clicks** on search results constitute a new source of feedback information A click indicate a document that is of interest to the user in the context of the current query Notice that a click does not necessarily indicate a document that is relevant to the query.



In an **implicit relevance feedback** cycle, the feedback information is derived implicitly by the system There are two basic approaches for compiling implicit feedback information:

local analysis, which derives the feedback information from the top ranked documents in the result set.

global analysis, which derives the feedback information from external sources such as a thesaurus.



Classic Relevance Feedback

In a classic relevance feedback cycle, the user is presented with a list of the retrieved documents Then, the user examines them and marks those that are relevant In practice, only the top 10 (or 20) ranked documents need to be examined The main idea consists of selecting important terms from the documents that have been identified as relevant, and enhancing the importance of these terms in a new query formulation.

Expected effect: the new query will be moved towards the relevant docs and away from the non-relevant ones Early experiments have shown good improvements in precision for small test collections Relevance feedback presents the following characteristics:

- it shields the user from the details of the query reformulation process (all the user has to provide is a relevance judgement)
- it breaks down the whole searching task into a sequence of small steps which are easier to grasp.

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.

TEXT CLASSIFICATION

First tactic for categorizing documents is to assign a label to each document, but this solve the problem only when the users know the labels of the documents they looking for. This tactic does not solve more generic problem of finding documents on specific topic or subject. For that case, better solution is to group documents by common generic topics and label each group with a meaningful name. Each labeled group is called category or class.

Document classification is the process of categorizing documents under a given cluster or category using fully supervised learning process. Classification could be performed manually by domain experts or automatically using well-known and widely used classification algorithms such as decision tree and Naïve Bayes. Documents are classified according to other attributes (e.g. author, document type, publishing year etc.) or according to their subjects. However, there are two main kind of subject classification of documents: The content based approach and the request based approach. In Content based classification**,** the weight that is given to subjects in a document decides the class to which the document is assigned. For example, it is a rule in some library classification that at least 15% of the content of a book should be about the class to which the book is assigned. In automatic classification, the number of times given words appears in a document determine the class. In Request oriented classification, the anticipated request from users is impacting how documents are being classified. The classifier asks himself:

"Under which description should this entity be found?" and "think of all the possible queries and decide for which ones the entity at hand is relevant". Automatic document classification tasks can be divided into three types

- Unsupervised document classification (document clustering): the classification must be done totally without reference to external information.
- Semi-supervised document classification: parts of the documents are labeled by the external method.
- Supervised document classification where some external method (such as human feedback) provides information on the correct classification for documents

Unsupervised Learning

Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

"*Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision".*

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will

perform this task by clustering the image dataset into the groups according to similarities between images.

By Simply,

- ✓ no training data is provided

Examples:

- neural network models
- independent component analysis
- clustering

UNSUPERVISED ALGORITHMS

Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

Difference between Supervised and Unsupervised Learning

Supervised Learning	**Unsupervised Learning**
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in **Classification** and **Regression** p roblems.	Unsupervised Learning can be classified in **Clustering** and **Associations** prob lems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
It includes various algorithms such	It includes various algorithms such

as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	as Clustering, KNN, and Apriori algorithm.

CLUSTERING

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

The clustering technique is commonly used for **statistical data analysis.**

Clustering Algorithms

The Clustering algorithms can be divided based on their models that are explained above. There are different types of clustering algorithms published, but only a few are commonly used. The clustering algorithm is based on the kind of data that we are using. Such as, some algorithms need to guess the number of clusters in the given dataset, whereas some are required to find the minimum distance between the observation of the dataset. Here we are discussing mainly popular Clustering algorithms that are widely used in machine learning:

- K-Means algorithm
- Mean-shift algorithm
- DBSCAN Algorithm
- Expectation-Maximization Clustering using GMM
- Agglomerative Hierarchical algorithm
- Affinity Propagation

NAIVE TEXT CLASSIFICATION

Naive Bayes classifiers are a collection of classification algorithms based on Bayes Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Naive Bayes classifiers have been heavily used for text classification and text analysis machine learning problems.

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols (i.e. strings) cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

The Naive Bayes algorithm

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The dataset is divided into two parts, namely, **feature matrix** and the **response/target vector**.

- The **Feature matrix** (X) contains all the vectors(rows) of the dataset in which each vector consists of the value of **dependent features**. The number of features is **d** i.e. **X = (x1,x2,x2, xd).**
- The **Response/target vector** (y) contains the value of **class/group variable** for **each row of feature matrix.**

The Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as follows:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

where:

- **A** and **B** are called **events.**
- P(A | B) is the probability of event A, given the event B is true (has occured). Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior independent probability, i.e. probability of event before evidence is seen).
- P(B | A) is the probability of B given event A, i.e. probability of event B after evidence A is seen.

Summary

A, B = events
$P(A \mid B)$ = probability of A given B is true
$P(B \mid A)$ = probability of B given A is true
$P(A), P(B)$ = the independent probabilities of A and B

The Naive Bayes Model

Given a data matrix **X** and a target vector **y,** we state our problem as:

$$P(y \mid X) = \frac{P(X \mid y)P(y)}{P(X)}$$

where, **y** is **class variable** and **X** is a **dependent feature vector with dimension d i.e. X = (x1,x2,x2, xd),** where **d** is the number of variables/features of the sample. P(y|X) is the probability of observing the class **y** given the sample **X** with **X = (x1,x2,x2, xd),** where **d** is the number of variables/features of the sample. Now the "naïve" conditional independence assumptions come into play: assume that all features in **X** are mutually independent, conditional on the category **y**:

$$P(y \mid x_1, \ldots, x_d) = \frac{P(y) \prod_{i=1}^{d} P(x_i|y)}{P(x_1) P(x_2) \ldots P(x_d)}$$

The denominator remains constant for a given input, so we can remove that term:

$$P(y \mid x_1, \ldots, x_d) \propto P(y) \prod_{i-1}^{d} P(x_i \mid y)$$

Finally, to find the probability of a given **sample** for all possible values of the class variable ***y***, we just need to find the output with maximum probability:

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Dealing with text data

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols (i.e. strings) cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

In order to address this, scikit-learn provides utilities for the most common ways to extract numerical features from text content, namely:

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **counting** the occurrences of tokens in each document.

In this scheme, features and samples are defined as follows:

- each **individual token occurrence frequency** is treated as a **feature**.
- the vector of all the token frequencies for a given **document** is considered a multivariate **sample**.

Example 1 : Using the Naive Bayesian Classifier

We will consider the following training set. The data samples are described by attributes age, income, student, and credit. The class label attribute, buy, tells whether the person buys a computer, has two distinct values, yes (class C_1) and no (class C_2).

RID	Age	Income	student	credit	C_i: buy
1	Youth	High	no	fair	C_2: no
2	Youth	High	no	excellent	C_2: no
3	middle-aged	High	no	fair	C1: yes
4	Senior	medium	no	fair	C1: yes
5	Senior	Low	yes	fair	C1: yes
6	Senior	Low	yes	excellent	C2: no
7	middle-aged	Low	yes	excellent	C1: yes
8	Youth	medium	no	fair	C2: no
9	Youth	Low	yes	fair	C1: yes
10	Senior	medium	yes	fair	C1: yes
11	Youth	medium	yes	excellent	C1: yes
12	middle-aged	medium	no	excellent	C1: yes
13	middle-aged	High	yes	fair	C1: yes
14	Senior	medium	no	excellent	C2: no

The sample we wish to classify is

X = (age = youth, income = medium, student = yes, credit = fair)

We need to maximize $P(X|C_i)P(C_i)$, for i = 1, 2. $P(C_i)$, the a priori probability of each

class, can be estimated based on the training samples:

P(buy =yes) = 9 /14
P(buy =no) = 5 /14

To compute P (X|Ci), for i = 1, 2, we compute the following conditional probabilities:

P(age=youth | buy =yes) = 2/9
P(income =medium | buy =yes) = 4/9
P(student =yes | buy =yes) = 6/9
P(credit =fair | buy =yes) = 6/9

P(age=youth | buy =no) = 3/5
P(income =medium | buy =no) = 2/5
P(student =yes | buy =no) = 1/5
P(credit =fair | buy =no) = 2/5

Using the above probabilities, we obtain

$$\begin{aligned} P(\mathbf{X}|buy = yes) &= P(age = youth|buy = yes) \\ &\quad P(income = medium|buy = yes) \\ &\quad P(student = yes|buy = yes) \\ &\quad P(credit = fair|buy = yes) \\ &= \frac{2}{9}\frac{4}{9}\frac{6}{9}\frac{6}{9} = 0.044. \end{aligned}$$

Similarly

$$P(\mathbf{X}|buy = no) = \frac{3}{5}\frac{2}{5}\frac{1}{5}\frac{2}{5} = 0.019$$

To find the class that maximizes P (X|Ci)P (Ci), we compute

$$P(\mathbf{X}|buy = yes)P(buy = yes) = 0.028$$
$$P(\mathbf{X}|buy = no)P(buy = no) = 0.007$$

Thus the naive Bayesian classifier predicts buy = yes for sample X.

Example 2:

Predicting a class label using naïve Bayesian classification. The training data set is given below. The data tuples are described by the attributes Owns Home?, Married, Gender and Employed. The class label attribute Risk Class has three distinct values. Let C1 corresponds to the class A, and C2 corresponds to the class B and C3 corresponds to the class C.

The tuple is to classify is,

X = (Owns Home = Yes, Married = No, Gender = Female, Employed = Yes)

Owns Home	Married	Gender	Employed	Risk Class
Yes	Yes	Male	Yes	B
No	No	Female	Yes	A
Yes	Yes	Female	Yes	C
Yes	No	Male	No	B
No	Yes	Female	Yes	C
No	No	Female	Yes	A

No	No	Male	No	B
Yes	No	Female	Yes	A
No	Yes	Female	Yes	C
Yes	Yes	Female	Yes	C

Solution

There are 10 samples and three classes.

Risk class A = 3

Risk class B = 3

Risk class C = 4

The prior probabilities are obtained by dividing these frequencies by the total number in the training data,

P(A) = 3/10 = 0.3
P(B) = 3/10 = 0.3
P(C) = 4/10 = 0.4

To compute P(X/Ci) =P {yes, no, female, yes}/Ci) for each of the classes, the conditional probabilities for each:

P(Owns Home = Yes/A) = 1/3 =0.33
P(Married = No/A) = 3/3 =1
P(Gender = Female/A) = 3/3 = 1
P(Employed = Yes/A) = 3/3 = 1

P(Owns Home = Yes/B) = 2/3 =0.67
P(Married = No/B) = 2/3 =0.67
P(Gender = Female/B) = 0/3 = 0
P(Employed = Yes/B) = 1/3 = 0.33

P(Owns Home = Yes/C) = 2/4 =0.5
P(Married = No/C) = 0/4 =0
P(Gender = Female/C) = 4/4 = 1
P(Employed = Yes/C) = 4/4 = 1

Using the above probabilities, we obtain
P(X/A)= P(Owns Home = Yes/A) X P(Married = No/A) x P(Gender = Female/A) X P(Employed = Yes/A)

= 0.33 x 1 x 1 x 1 = 0.33

Similarly, P(X/B)= 0 , P(X/C) =0

To find the class, G, that maximizes,
P(X/Ci)P(Ci), we compute,

P(X/A) P(A) = 0.33 X 0.3 = 0.099
P(X/B) P(B) =0 X 0.3 = 0
P(X/C) P(C) = 0 X 0.4 = 0.0

Therefore x is assigned to class A

Advantages:

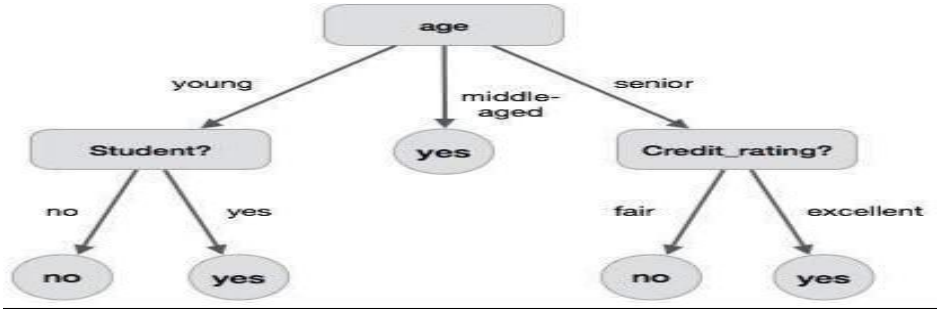
- Have the minimum error rate in comparison to all other classifiers.
- Easy to implement
- Good results obtained in most of the cases.
- They provide theoretical justification for other classifiers that do not explicitly use

- Lack of available probability data.
- Inaccuracies in the assumption.

5.5.2 Decision Trees

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows −

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

Decision Tree Induction Algorithm

- A machine researcher named J. Ross Quinlan in 1980 developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). Later, he presented C4.5, which was the successor of ID3. ID3 and C4.5 adopt a greedy approach. In this algorithm, there is no backtracking; the trees are constructed in a top-down recursive divide-and-conquer manner.

Generating a decision tree form training tuples of data partition D

Algorithm : Generate_decision_tree

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 that the data tuples into individual classes. This criterion includes a splitting_attribute and either a splitting point or splitting subset.

Output: A Decision Tree

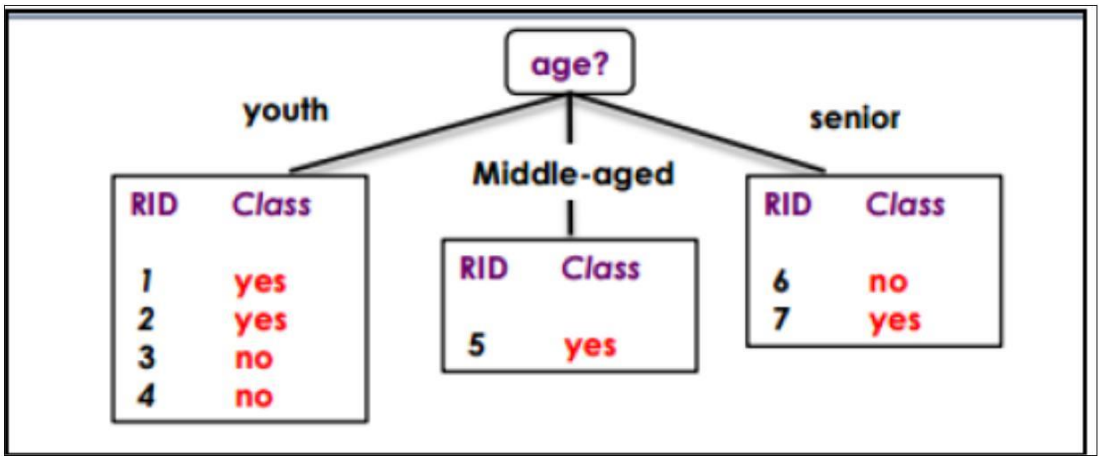
Method

1. create a node N;
2. if tuples in D are all of the same class, C then
3. return N as leaf node labeled with class C;
4. if attribute_list is empty then
5. return N as leaf node with labeled with 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 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;
    end for
15. return N;

Example : customer likely to purchase a computer

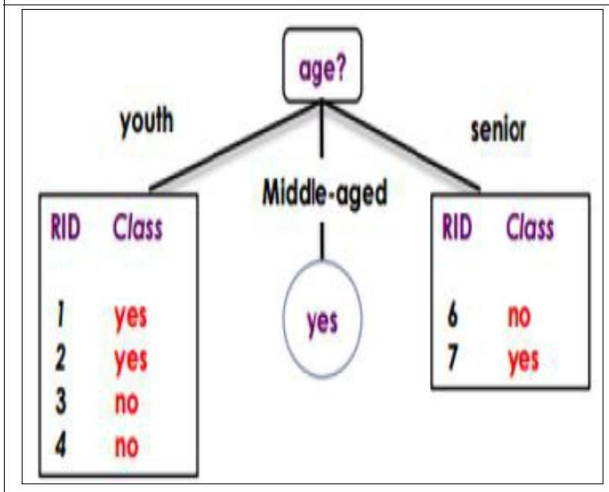
From the training dataset , calculate entropy value, which indicates that splitting attribute is: age

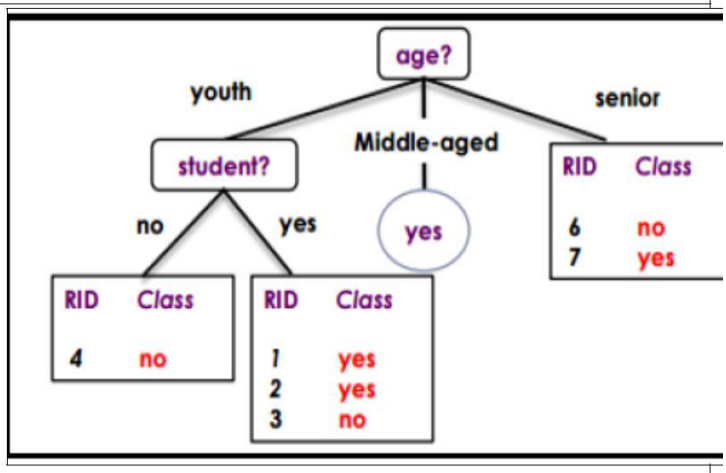
RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



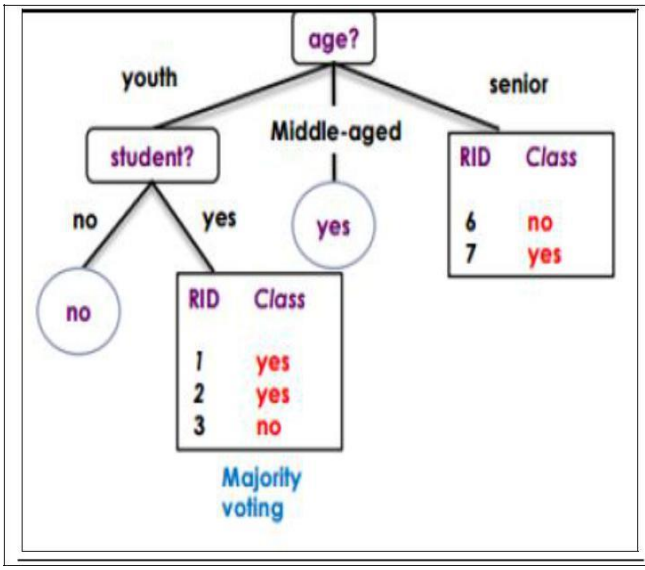
Age has 3 partitions :

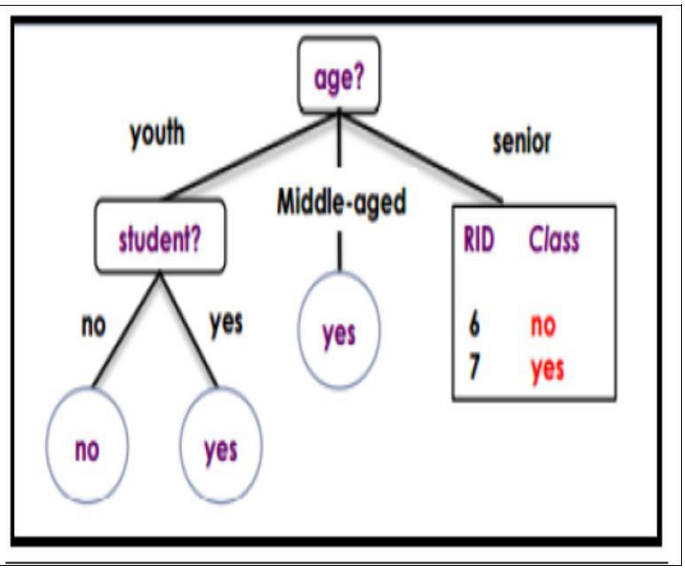
From the training data set , age= youth has 2 classes based on student attribute



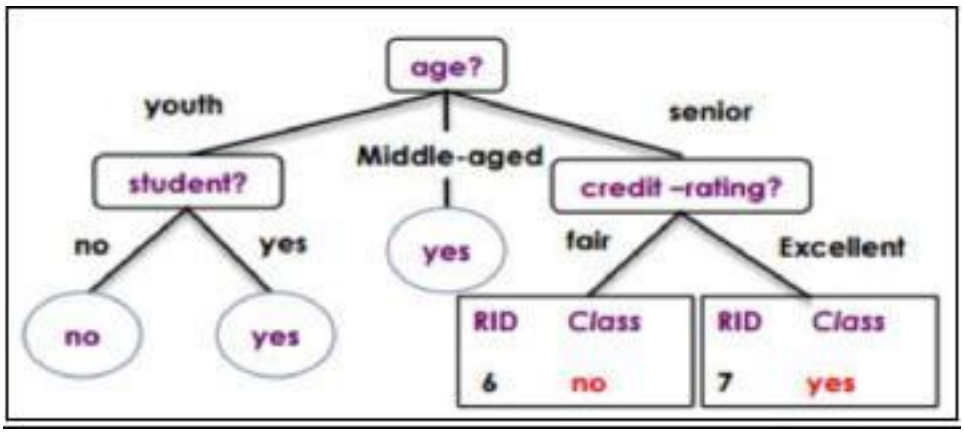


based on majority voting in student attribute , RID=3 is grouped under yes group.

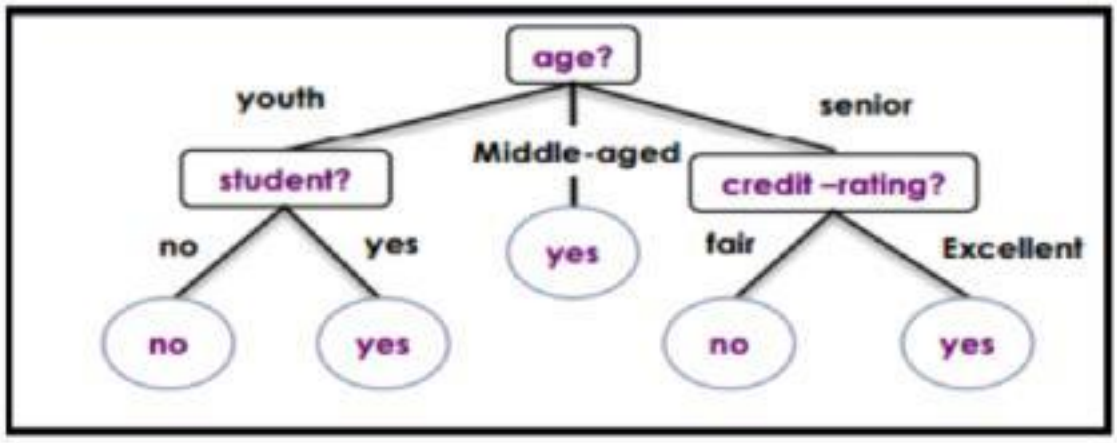




From the training data set , age= senior has 2 classes based on credit rating.

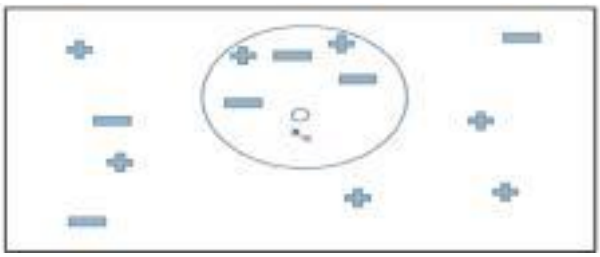


Final Decision Tree



5.5.3 K Nearest Neighbor Classification

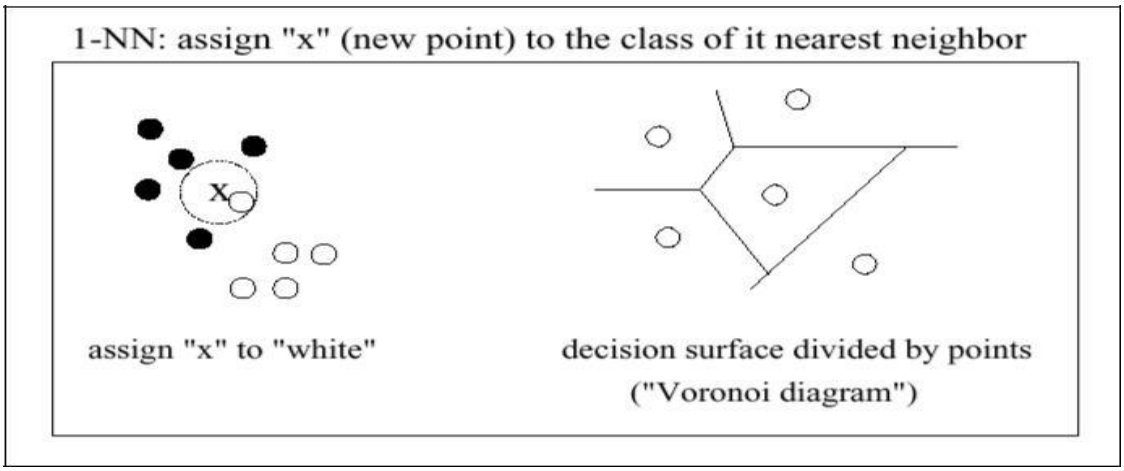
- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). K represents number of nearest neighbors.
- It classify an unknown example with the most common class among k closest examples
- KNN is based on "tell me who your neighbors are, and I'll tell you who you are"
- Example:

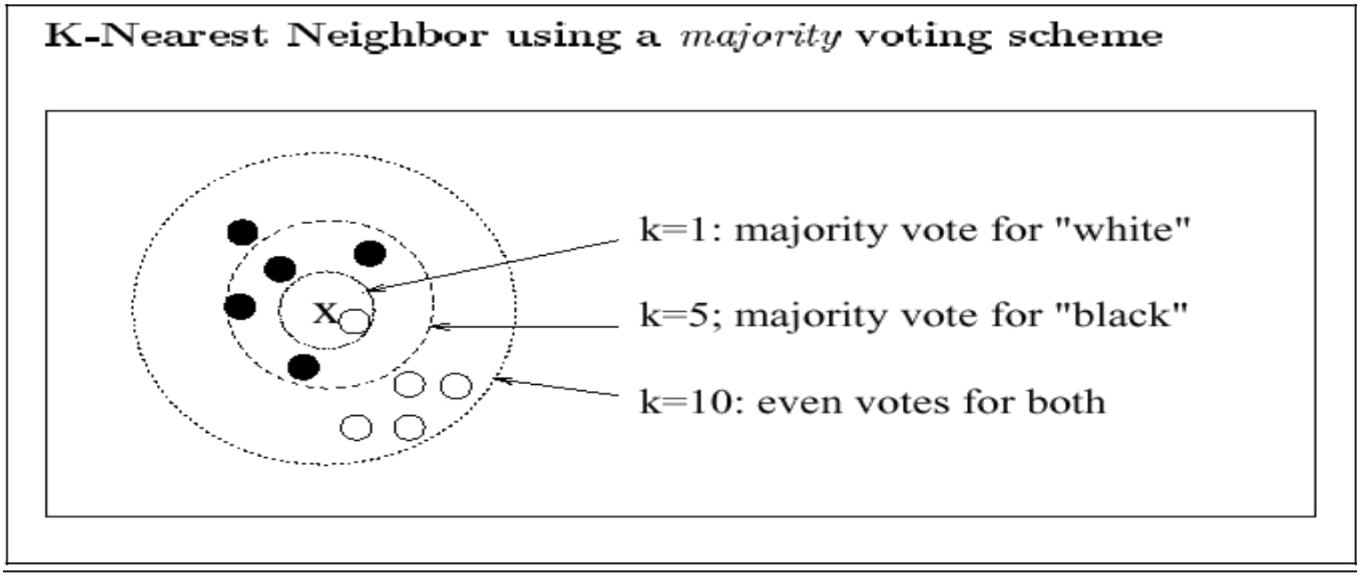


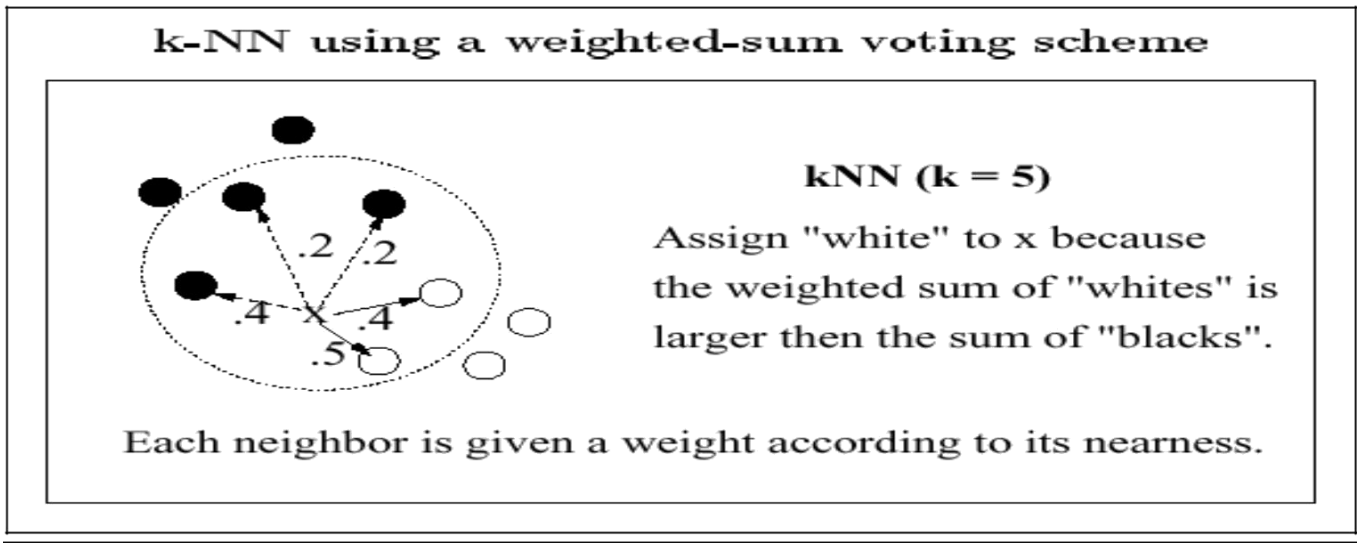
- If K = 5, then in this case query instance x_q will be classified as negative since three of its nearest neighbors are classified as negative.

Different Schemes of KNN

a. 1-Nearest Neighbor
b. K-Nearest Neighbor using a majority voting scheme
c. K-NN using a weighted-sum voting Scheme







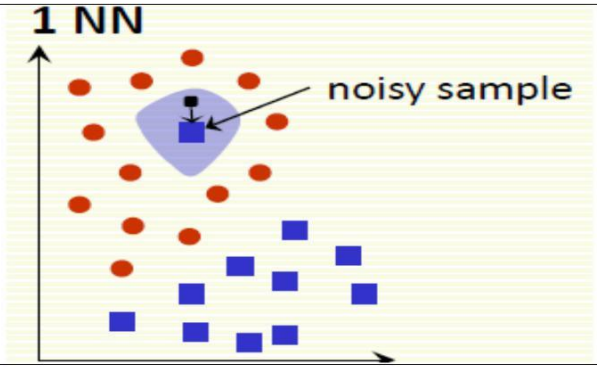
kNN: How to Choose k?

In theory, if infinite number of samples available, the larger is ***k***, the better is classification

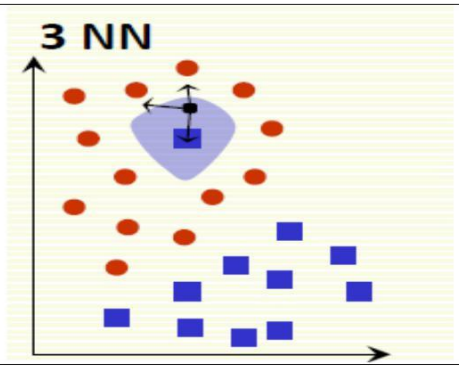
The limitation is that all ***k*** neighbors have to be close

- Possible when infinite no of samples available
- Impossible in practice since no of samples is finite

k = 1 is often used for efficiency, but sensitive to "noise"



every example in the blue shaded area will be misclassified as the blue class(rectangle)



every example in the blue shaded area will be classified correctly as the red class(circle)

Larger k gives smoother boundaries, better for generalization But only if locality is preserved. Locality is not preserved if end up looking at samples too far away, not from the same class.

•Interesting theoretical properties if k < sqrt(n), n is # of examples .

Find a heuristically optimal number k of nearest neighbors, based on RMSE(root-mean-square error). This is done using cross validation.

Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

Distance Measure in KNN

There are three distance measures are valid for continuous variables.

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

It should also be noted that all In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Hamming Distance

$$D_H = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

Simple KNN - Algorithm:

For each training example , add the example to the list of training_examples.

Given a query instance x_q to be classified,

- Let x1 ,x2….xk denote the k instances from training_examples that are nearest to x_q .
- Return the class that represents the maximum of the k instances

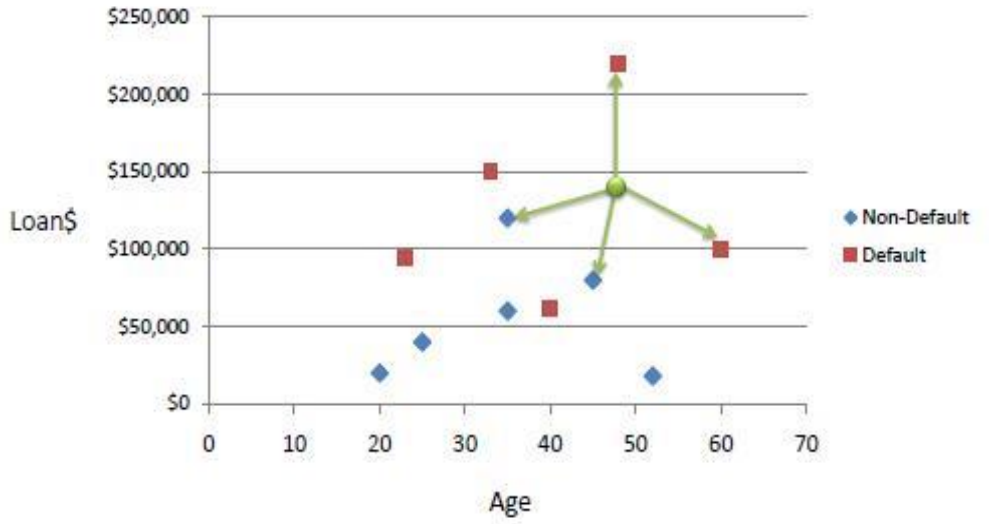
Steps:

1. Determine parameter k= no of nearest neighbor
2. Calculate the distance between the query instance and all the training samples

3. Sort the distance and determine nearest neighbor based on the k –th minimum distance
4. Gather the category of the nearest neighbors
5. Use simple majority of the category of nearest neighbors as the prediction value of the query instance.

Example:

Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



Given Training Data set :

Age	Loan	Default
25	$40,000	N
35	$60,000	N
45	$80,000	N
20	$20,000	N
35	$120,000	N
52	$18,000	N
23	$95,000	Y
40	$62,000	Y
60	$100,000	Y
48	$220,000	Y
33	$150,000	Y

Data to Classify:

to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance.

Step1: Determine parameter k

K=3

Step 2: Calculate the distance

D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y

Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	$142,000	?		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Step 3: Sort the distance (refer above diagram) and mark upto k_{th} rank i.e 1 to 3.

Step 4: Gather the category of the nearest neighbors

Age	Loan	Default	Distance
33	$150000	Y	8000
35	$120000	N	22000
60	$100000	Y	42000

With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is Default=Y.

Standardized Distance (Feature Normalization)

One major drawback in calculating distance measures directly from the training set is in the case where variables have different measurement scales or there is a mixture of numerical and categorical variables. For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated. One solution is to standardize the training set as shown below.

Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

Standardized Variable

$$X_s = \frac{X - Min}{Max - Min}$$

For ex loan , X =$ 40000 ,

$$Xs = \frac{40000 - 20000}{220000 - 20000} = 0.11$$

Same way , calculate the standardized values for age and loan attributes, then apply the KNN algorithm.

Advantages

- Can be applied to the data from any distribution
  - for example, data does not have to be separable with a linear boundary
- Very simple and intuitive
- Good classification if the number of samples is large enough

Disadvantages

- Choosing k may be tricky
- Test stage is computationally expensive
  - No training stage, all the work is done during the test stage
  - This is actually the opposite of what we want. Usually we can afford training step to take a long time, but we want fast test step
- Need large number of samples for accuracy

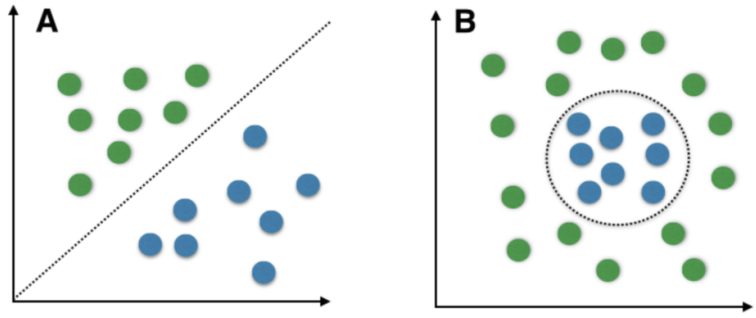
SUPPORT VECTOR MACHINE

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems. **Support Vector Machine for Multi-class Problems** To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :

- The data point belongs to that class OR
- The data point does not belong to that class.

For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the 'mango' class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM.

SVM for complex (Non Linearly Separable) SVM works very well without any modifications for linearly separable data. **Linearly Separable Data** is any data that can be plotted in a graph and can be separated into classes using a straight line.

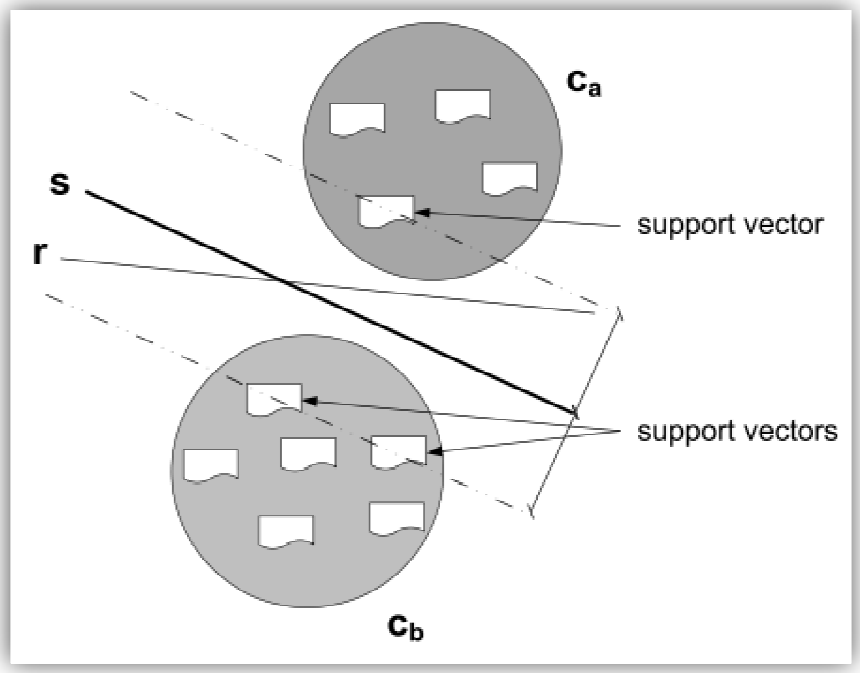


A: Linearly Separable Data B: Non-Linearly Separable Data

SVM CLASSIFIER

- ✓ a vector space method for binary classification problems
- ✓ documents represented in t-dimensional space
- ✓ find a **decision surface (hyperplane)** that best separate
- ✓ documents of two classes
- ✓ new document classified by its position relative to hyperplane.

Simple 2D example: training documents linearly separable

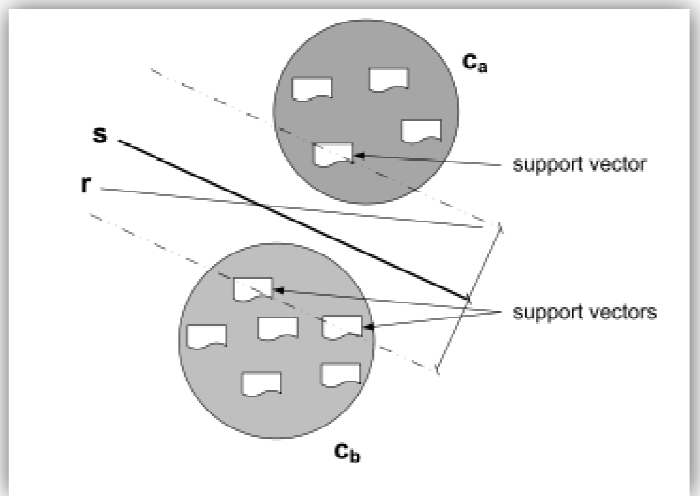


Line s—The Decision Hyperplane

- ✓ maximizes distances to closest docs of each class
- ✓ it is the best separating hyperplane

Delimiting Hyperplanes

- ✓ parallel dashed lines that delimit region where to look for a solution

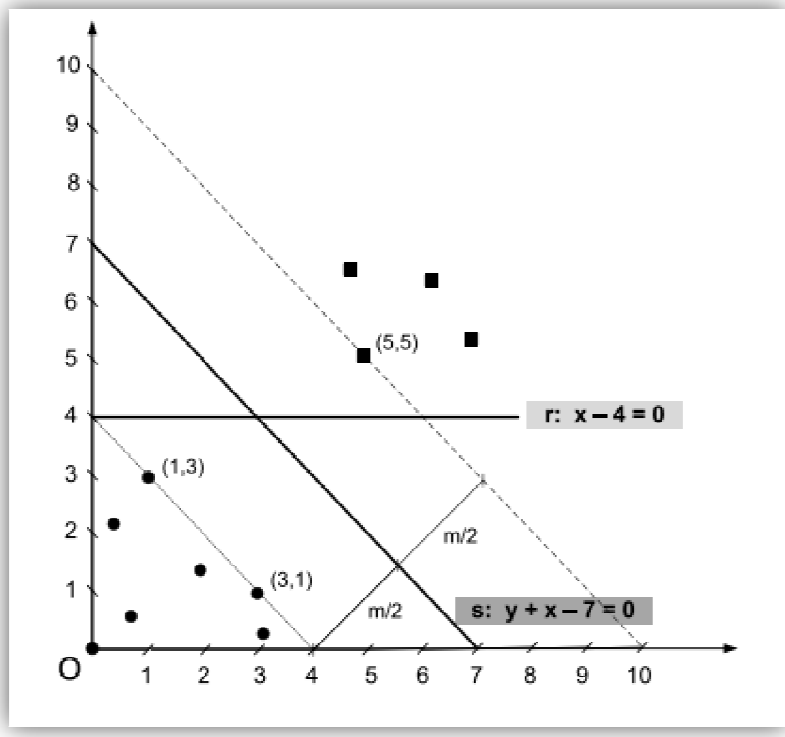


✓ Lines that cross the delimiting hyperplanes.

- candidates to be selected as the decision hyperplane
- lines that are parallel to delimiting hyperplanes: best candidates

Support vectors: documents that belong to, and define, the delimiting hyperplanes

Our example in a 2-dimensional system of coordinates



Let,

- $\mathcal{H}_w$: a hyperplane that separates docs in classes c_a and c_b
- m_a: distance of $\mathcal{H}_w$ to the closest document in class c_a
- m_b: distance of $\mathcal{H}_w$ to the closest document in class c_b
- $m_a + m_b$: **margin** m of the SVM

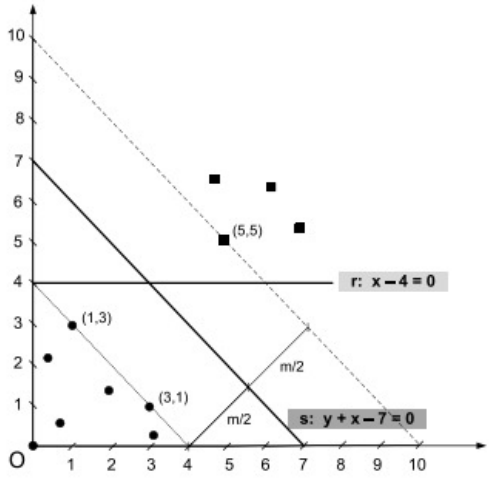
The **decision hyperplane** maximizes the margin m

Hyperplane $r : x - 4 = 0$ separates docs in two sets

- its distances to closest docs in either class is 1
- thus, its margin m is 2

Hyperplane $s : y + x - 7 = 0$ has margin equal to $3\sqrt{2}$

- maximum for this case
- s is the decision hyperplane



FEATURE SELECTION OR DIMENSIONALITY REDUCTION

Feature selection and dimensionality reduction allow us to minimize the number of features in a dataset by only keeping features that are important. In other words, we want to retain features that contain the most useful information that is needed by our model to make accurate predictions while discarding redundant features that contain little to no information. There are several benefits in performing feature selection and dimensionality reduction which include model

interpretability, minimizing overfitting as well as reducing the size of the training set and consequently training time.

Dimensionality Reduction

The number of input variables or features for a dataset is referred to as its dimensionality. Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset. More input features often make a predictive modeling task more challenging to model, more generally referred to as the curse of dimensionality. High-dimensionality statistics and dimensionality reduction techniques are often used for data visualization. Nevertheless these techniques can be used in applied machine learning to simplify a classification or regression dataset in order to better fit a predictive model.

Problem With Many Input Variables

If your data is represented using rows and columns, such as in a spreadsheet, then the input variables are the columns that are fed as input to a model to predict the target variable. Input variables are also called features. We can consider the columns of data representing dimensions on an n-dimensional feature space and the rows of data as points in that space. This is a useful geometric interpretation of a dataset. Having a large number of dimensions in the feature space can mean that the volume of that space is very large, and in turn, the points that we have in that space (rows of data) often represent a small and non-representative sample. This can dramatically impact the performance of machine learning algorithms fit on data with many input features, generally referred to as the "curse of dimensionality."

Therefore, it is often desirable to reduce the number of input features. This reduces the number of dimensions of the feature space, hence the name "*dimensionality reduction*."

Dimensionality Reduction

Dimensionality reduction refers to techniques for reducing the number of input variables in training data.

When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the "essence" of the data. This is called dimensionality reduction.

Fewer input dimensions often mean correspondingly fewer parameters or a simpler structure in the machine learning model, referred to as degrees of freedom. A model with too many degrees of freedom is likely to overfit the training dataset and therefore may not perform well on new data.

It is desirable to have simple models that generalize well, and in turn, input data with few input variables. This is particularly true for linear models where the number of inputs and the degrees of freedom of the model are often closely related.

Techniques for Dimensionality Reduction

There are many techniques that can be used for dimensionality reduction.

- Feature Selection Methods
- Matrix Factorization
- Manifold Learning
- Auto encoder Methods

Feature Selection Methods

Feature selection is also called variable selection or attribute selection.

It is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modeling problem you are working on.

feature selection… is the process of selecting a subset of relevant features for use in model construction

Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction

method do so by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them.

Examples of dimensionality reduction methods include Principal Component Analysis, Singular Value Decomposition and Sammon's Mapping.

Feature selection is itself useful, but it mostly acts as a filter, muting out features that aren't useful in addition to your existing features.

Feature Selection Algorithms

Filter Methods

Filter feature selection methods apply a statistical measure to assign a scoring to each feature. The features are ranked by the score and either selected to be kept or removed from the dataset. The methods are often univariate and consider the feature independently, or with regard to the dependent variable. Some examples of some filter methods include the Chi squared test, information gain and correlation coefficient scores.

Wrapper Methods

Wrapper methods consider the selection of a set of features as a search problem, where different combinations are prepared, evaluated and compared to other combinations. A predictive model us used to evaluate a combination of features and assign a score based on model accuracy. The search process may be methodical such as a best-first search, it may stochastic such as a random hill-climbing algorithm, or it may use heuristics, like forward and backward passes to add and remove features. An example if a wrapper method is the recursive feature elimination algorithm.

Embedded Methods

Embedded methods learn which features best contribute to the accuracy of the model while the model is being created. The most common type of embedded feature selection methods are regularization methods. Regularization methods are also called penalization methods that introduce additional constraints into the

optimization of a predictive algorithm (such as a regression algorithm) that bias the model toward lower complexity (fewer coefficients). Examples of regularization algorithms are the LASSO, Elastic Net and Ridge Regression.

EVALUATION METRICS

Evaluation

- important for any text classification method
- key step to validate a newly proposed classification method

Contingency Table

- Let
  - $\mathcal{D}$: collection of documents
  - $\mathcal{D}_t$: subset composed of training documents
  - N_t: number of documents in $\mathcal{D}_t$
  - $\mathcal{C} = \{c_1, c_2, \ldots, c_L\}$: set of all L classes
- Further let
  - $\mathcal{T} : \mathcal{D}_t \times \mathcal{C} \rightarrow [0, 1]$: training set function
  - n_t: number of docs from training set $\mathcal{D}_t$ in class c_p
  - $\mathcal{F} : \mathcal{D} \times \mathcal{C} \rightarrow [0, 1]$: text classifiier function
  - n_f: number of docs from training set assigned to class c_p by the classifier

- Apply classifier to all documents in training set
- Contingency table is given by

$Case$	$\mathcal{T}(d_j, c_p) = 1$	$\mathcal{T}(d_j, c_p) = 0$	$Total$
$\mathcal{F}(d_j, c_p) = 1$	$n_{f,t}$	$n_f - n_{f,t}$	n_f
$\mathcal{F}(d_j, c_p) = 0$	$n_t - n_{f,t}$	$N_t - n_f - n_t + n_{f,t}$	$N_t - n_f$
$All\ docs$	n_t	$N_t - n_t$	N_t

  - $n_{f,t}$: number of docs that both the training and classifier functions assigned to class c_p
  - $n_t - n_{f,t}$: number of training docs in class c_p that were miss-classified
  - The remaining quantities are calculated analogously

Accuracy and Error

- Accuracy and error metrics, relative to a given class c_p

$$\begin{aligned} Acc(c_p) &= \frac{n_{f,t} + (N_t - n_f - n_t + n_{f,t})}{N_t} \\ Err(c_p) &= \frac{(n_f - n_{f,t}) + (n_t - n_{f,t})}{N_t} \\ Acc(c_p) + Err(c_p) &= 1 \end{aligned}$$

- These metrics are commonly used for evaluating classifiers
- Accuracy and error have disadvantages
  - consider classification with only two categories c_p and c_r
  - assume that out of 1,000 docs, 20 are in class c_p
  - a classifier that assumes all docs not in class c_p
    - accuracy = 98%
    - error = 2%

    which erroneously suggests a very good classifier

- Consider now a second classifier that correctly predicts 50% of the documents in c_p

	$\mathcal{T}(d_j, c_p) = 1$	$\mathcal{T}(d_j, c_p) = 0$	
$\mathcal{F}(d_j, c_p) = 1$	10	0	10
$\mathcal{F}(d_j, c_p) = 0$	10	980	990
all docs	20	980	1,000

- In this case, accuracy and error are given by

$$Acc(c_p) = \frac{10 + 980}{1,000} = 99\%$$

$$Err(c_p) = \frac{10 + 0}{1,000} = 1\%$$

- This classifier is much better than one that guesses that all documents are not in class c_p
- However, its accuracy is just 1% better, it increased from 98% to 99%
- This suggests that the two classifiers are almost equivalent, which is not the case.

Precision and Recall

- Variants of precision and recall metrics in IR
- Precision P and recall R relative to a class c_p

$$P(c_p) = \frac{n_{f,t}}{n_f} \qquad R(c_p) = \frac{n_{f,t}}{n_t}$$

  - Precision is the fraction of all docs assigned to class c_p by the classifier that really belong to class c_p
  - Recall is the fraction of all docs that belong to class c_p that were correctly assigned to class c_p

- Consider again the classifier illustrated below

	$\mathcal{T}(d_j, c_p) = 1$	$\mathcal{T}(d_j, c_p) = 0$	
$\mathcal{F}(d_j, c_p) = 1$	10	0	10
$\mathcal{F}(d_j, c_p) = 0$	10	980	990
all docs	20	980	1,000

- Precision and recall figures are given by

$$P(c_p) = \frac{10}{10} = 100\%$$
$$R(c_p) = \frac{10}{20} = 50\%$$

- Precision and recall
  - computed for every category in set $\mathcal{C}$
  - great number of values
    - makes tasks of comparing and evaluating algorithms more difficult
- Often convenient to combine precision and recall into a single quality measure
  - one of the most commonly used such metric: *F-measure*

F-measure

- F-measure is defined as

$$F_\alpha(c_p) = \frac{(\alpha^2 + 1)P(c_p)R(c_p)}{\alpha^2 P(c_p) + R(c_p)}$$

  - α: relative importance of precision and recall
  - when $\alpha = 0$, only precision is considered
  - when $\alpha = \infty$, only recall is considered
  - when $\alpha = 0.5$, recall is half as important as precision
  - when $\alpha = 1$, common metric called F_1-measure

$$F_1(c_p) = \frac{2P(c_p)R(c_p)}{P(c_p) + R(c_p)}$$

- Consider again the the classifier illustrated below

	$\mathcal{T}(d_j, c_p) = 1$	$\mathcal{T}(d_j, c_p) = 0$	
$\mathcal{F}(d_j, c_p) = 1$	10	0	10
$\mathcal{F}(d_j, c_p) = 0$	10	980	990
all docs	20	980	1,000

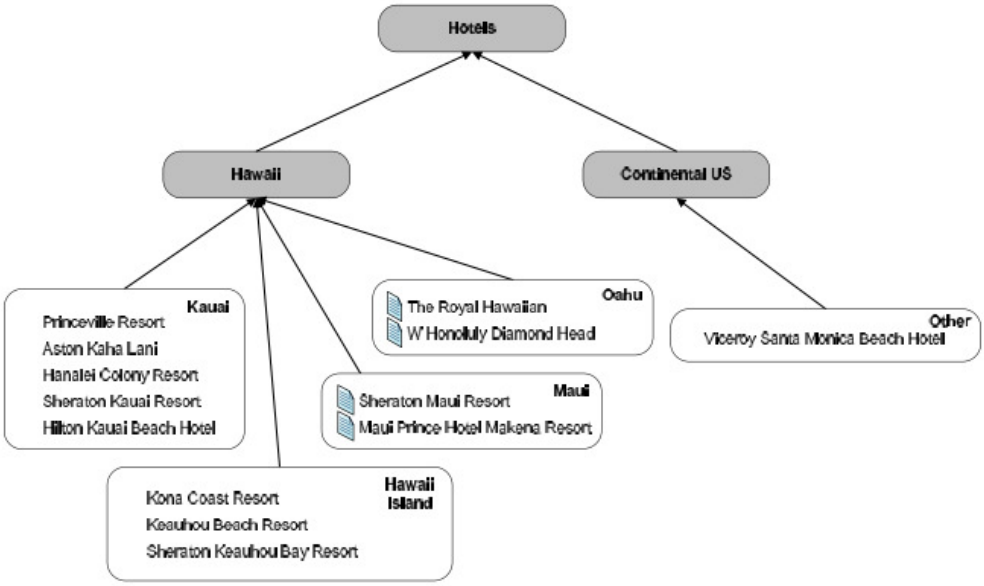
- For this example, we write

$$F_1(c_p) = \frac{2 * 1 * 0.5}{1 + 0.5} \sim 67\%$$

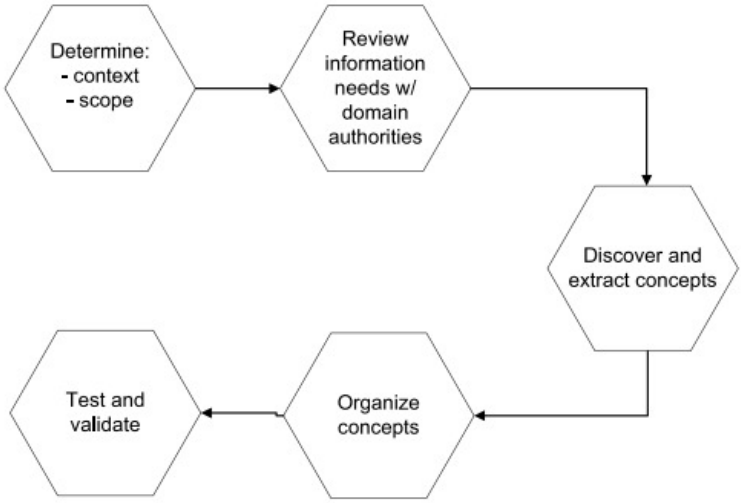
ORGANIZING THE CLASSES TAXONOMIES

Taxonomies

- Labels provide information on semantics of each class
- Lack of organization of classes restricts comprehension and reasoning
- Hierarchical organization of classes
  - most appealing to humans
  - hierarchies allow reasoning with more generic concepts
  - also provide for specialization, which allows breaking up a larger set of entities into subsets
- To organize classes hierarchically use
  - specialization
  - generalization
  - sibling relations
- Classes organized hierarchically compose a **taxonomy**
  - relations among classes can be used to fine tune the classifier
  - taxonomies make more sense when built for a specific domain of knowledge



- Taxonomies are built **manually** or **semi-automatically**
- Process of building a taxonomy:



- Manual taxonomies tend to be of superior quality
  - better reflect the information needs of the users
- Automatic construction of taxonomies
  - needs more research and development
- Once a taxonomy has been built
  - documents can be classified according to its concepts
  - can be done manually or automatically
  - automatic classification is advanced enough to work well in practice

INDEXING AND SEARCHING

Introduction

- Although **efficiency** might seem a secondary issue compared to **effectiveness**, it can rarely be neglected in the design of an IR system
- **Efficiency in IR systems**: to process user queries with minimal requirements of computational resources
- As we move to larger-scale applications, efficiency becomes more and more important
  - For example, in Web search engines that index terabytes of data and serve hundreds or thousands of queries per second

- **Index**: a data structure built from the text to speed up the searches
- In the context of an IR system that uses an index, the efficiency of the system can be measured by:
  - **Indexing time:** Time needed to build the index
  - **Indexing space:** Space used during the generation of the index
  - **Index storage:** Space required to store the index
  - **Query latency:** Time interval between the arrival of the query and the generation of the answer
  - **Query throughput:** Average number of queries processed per second

- When a text is updated, any index built on it must be updated as well
- Current indexing technology is not well prepared to support very frequent changes to the text collection
- **Semi-static collections:** collections which are updated at reasonable regular intervals (say, daily)
- Most real text collections, including the Web, are indeed semi-static
  - For example, although the Web changes very fast, the crawls of a search engine are relatively slow
- For maintaining freshness, incremental indexing is used

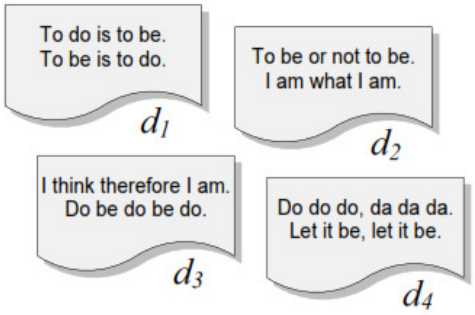
INVERTED INDEXES

Basic Concepts

- **Inverted index**: a word-oriented mechanism for indexing a text collection to speed up the searching task
- The inverted index structure is composed of two elements: the **vocabulary** and the **occurrences**
- The vocabulary is the set of all different words in the text
- For each word in the vocabulary the index stores the documents which contain that word (inverted index)

- **Term-document matrix**: the simplest way to represent the documents that contain each word of the vocabulary

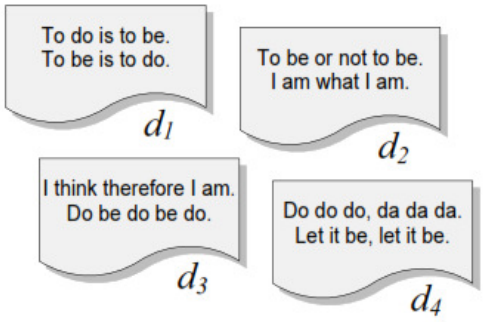
Vocabulary	n_i	d_1	d_2	d_3	d_4
to	2	4	2	-	-
do	3	2	-	3	3
is	1	2	-	-	-
be	4	2	2	2	2
or	1	-	1	-	-
not	1	-	1	-	-
I	2	-	2	2	-
am	2	-	2	1	-
what	1	-	1	-	-
think	1	-	-	1	-
therefore	1	-	-	1	-
da	1	-	-	-	3
let	1	-	-	-	2
it	1	-	-	-	2



- The main problem of this simple solution is that it requires too much space
- As this is a sparse matrix, the solution is to associate a list of documents with each word
- The set of all those lists is called the **occurrences**

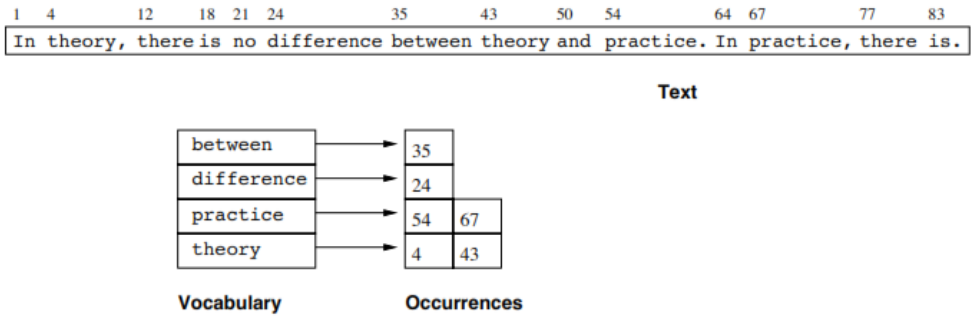
- Basic inverted index

Vocabulary	n_i	Occurrences as inverted lists
to	2	[1,4],[2,2]
do	3	[1,2],[3,3],[4,3]
is	1	[1,2]
be	4	[1,2],[2,2],[3,2],[4,2]
or	1	[2,1]
not	1	[2,1]
I	2	[2,2],[3,2]
am	2	[2,2],[3,1]
what	1	[2,1]
think	1	[3,1]
therefore	1	[3,1]
da	1	[4,3]
let	1	[4,2]
it	1	[4,2]



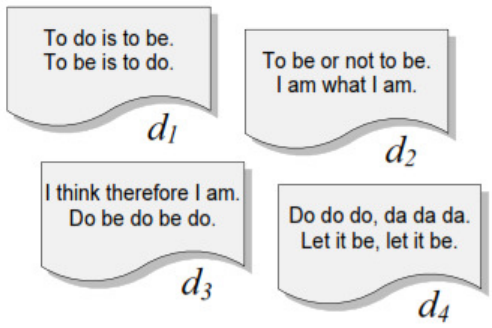
Full Inverted Indexes

- The basic index is not suitable for answering phrase or proximity queries
- Hence, we need to add the positions of each word in each document to the index (full inverted index)



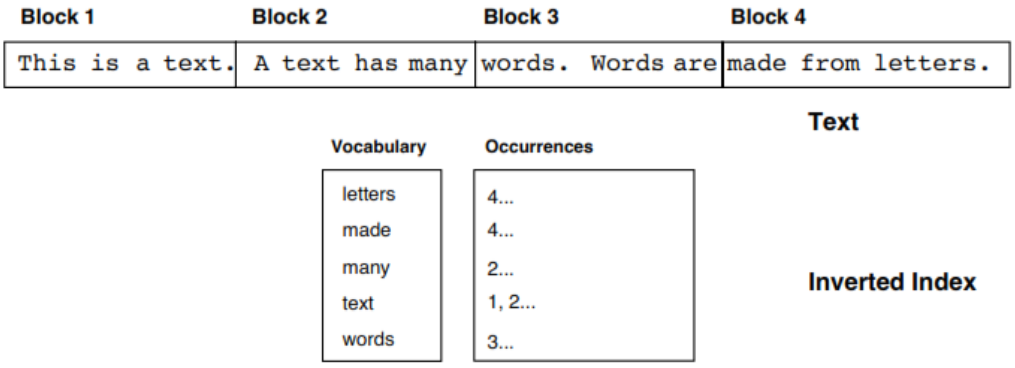
- In the case of multiple documents, we need to store one occurrence list per term-document pair

Vocabulary	n_i	Occurrences as full inverted lists
to	2	[1,4,[1,4,6,9]],[2,2,[1,5]]
do	3	[1,2,[2,10]],[3,3,[6,8,10]],[4,3,[1,2,3]]
is	1	[1,2,[3,8]]
be	4	[1,2,[5,7]],[2,2,[2,6]],[3,2,[7,9]],[4,2,[9,12]]
or	1	[2,1,[3]]
not	1	[2,1,[4]]
I	2	[2,2,[7,10]],[3,2,[1,4]]
am	2	[2,2,[8,11]],[3,1,[5]]
what	1	[2,1,[9]]
think	1	[3,1,[2]]
therefore	1	[3,1,[3]]
da	1	[4,3,[4,5,6]]
let	1	[4,2,[7,10]]
it	1	[4,2,[8,11]]



- The space required for the vocabulary is rather small
- Heaps' law: the vocabulary grows as $O(n^{\beta})$, where
  - n is the collection size
  - β is a collection-dependent constant between 0.4 and 0.6
- For instance, in the TREC-3 collection, the vocabulary of 1 gigabyte of text occupies only 5 megabytes
- This may be further reduced by stemming and other normalization techniques
- The occurrences demand much more space
- The extra space will be $O(n)$ and is around
  - 40% of the text size if stopwords are omitted
  - 80% when stopwords are indexed
- Document-addressing indexes are smaller, because only one occurrence per file must be recorded, for a given word
- Depending on the document (file) size, document-addressing indexes typically require 20% to 40% of the text size

- To reduce space requirements, a technique called **block addressing** is used
- The documents are divided into blocks, and the occurrences point to the blocks where the word appears



- The Table below presents the projected space taken by inverted indexes for texts of different sizes

Index granularity	Single document (1 MB)		Small collection (200 MB)		Medium collection (2 GB)	
Addressing words	45%	73%	36%	64%	35%	63%
Addressing documents	19%	26%	18%	32%	26%	47%
Addressing 64K blocks	27%	41%	18%	32%	5%	9%
Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

- The blocks can be of fixed size or they can be defined using the division of the text collection into documents
- The division into blocks of fixed size improves efficiency at retrieval time
  - This is because larger blocks match queries more frequently and are more expensive to traverse
- This technique also profits from *locality of reference*
  - That is, the same word will be used many times in the same context and all the references to that word will be collapsed in just one reference

SEARCHING

Searching

- Searching a single word is made by comparing its bit mask W with the bit masks B_i of all the text blocks
- Whenever $(W \ \& \ B_i = W)$, where $\&$ is the bit-wise AND, the text block **may** contain the word
- Hence, an online traversal must be performed to verify if the word is actually there
- This traversal cannot be avoided as in inverted indexes (except if the risk of a false match is accepted)

- This scheme is more efficient to search phrases and reasonable proximity queries
  - This is because **all** the words must be present in a block in order for that block to hold the phrase or the proximity query
- Hence, the bit-wise OR of all the query masks is searched, so that **all** their bits must be present
  - This reduces the probability of false drops
- Some care has to be exercised at block boundaries, to avoid missing a phrase which crosses a block limit
- To search phrases of j words or proximities of up to j words, consecutive blocks must overlap in $j-1$ words
- This is the only indexing scheme which improves in phrase searching

SEQUENTIAL SEARCHING

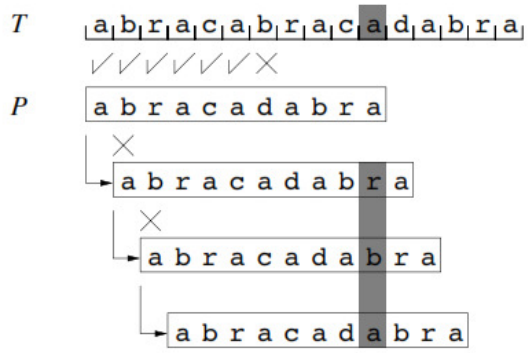
Sequential Searching

- In general the sequential search problem is:
  - Given a text $T = t_1 t_2 \ldots t_n$ and a pattern denoting a set of strings $\mathcal{P}$, find all the occurrences of the strings of $\mathcal{P}$ in T
- **Exact string matching:** the simplest case, where the pattern denotes just a single string $P = p_1 p_2 \ldots p_m$
- This problem subsumes many of the basic queries, such as word, prefix, suffix, and substring search
- We assume that the strings are sequences of characters drawn from an alphabet Σ of size σ

Simple Strings: Brute Force

- The **brute force** algorithm:
  - Try out all the possible pattern positions in the text and checks them one by one
- More precisely, the algorithm slides a **window** of length m across the text, $t_{i+1}t_{i+2}\ldots t_{i+m}$ for $0 \leq i \leq n-m$
- Each window denotes a potential pattern occurrence that must be verified
- Once verified, the algorithm slides the window to the next position

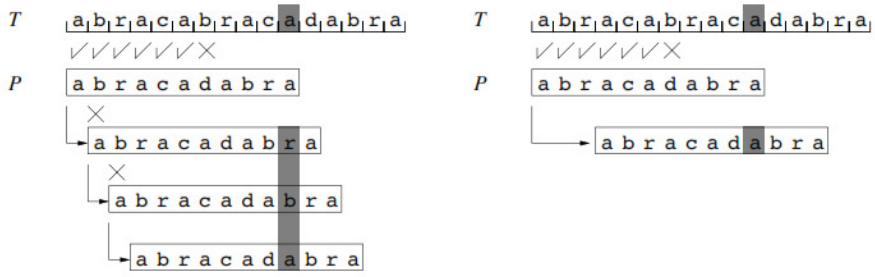
- A sample text and pattern searched for using brute force
  - The first text window is `abracabraca`
  - After verifying that it does not match P, the window is shifted by one position



Simple Strings: Horspool

- **Horspool's algorithm** is in the fortunate position of being very simple to understand and program
- It is the fastest algorithm in many situations, especially when searching natural language texts
- Horspool's algorithm uses the previous idea to shift the window in a smarter way
- A table d indexed by the characters of the alphabet is precomputed:
  - $d[c]$ tells how many positions can the window be shifted if the final character of the window is c
- In other words, $d[c]$ is the distance from the end of the pattern to the last occurrence of c in P, excluding the occurrence of p_m

- The Figure repeats the previous example, now also applying Horspool's shift



- Pseudocode for Horspool's string matching algorithm

Horspool $(T = t_1 t_2 \dots t_n,\ P = p_1 p_2 \dots p_m)$

(1) **for** $c \in \Sigma$ **do** $d[c] \leftarrow m$
(2) **for** $j \leftarrow 1 \dots m-1$ **do** $d[p_j] \leftarrow m - j$
(3) $i \leftarrow 0$
(4) **while** $i \leq n - m$ **do**
(5) $\quad j \leftarrow 1$
(6) $\quad$ **while** $j \leq m \ \wedge\ t_{i+j} = p_j$ **do** $j \leftarrow j + 1$
(7) $\quad$ **if** $j > m$ **then** report an occurrence at text position $i + 1$
(8) $\quad i \leftarrow i + d[t_{i+m}]$

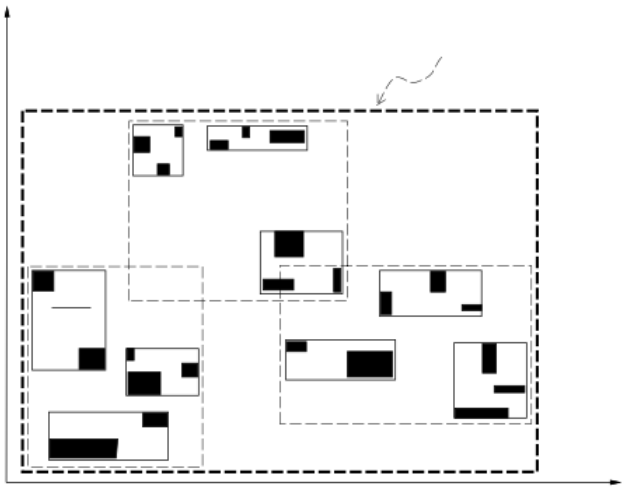
MULTI-DIMENSIONAL INDEXING

Multi-dimensional Indexing

- In multimedia data, we can represent every object by several numerical features
- For example, imagine an image from where we can extract a color histogram, edge positions, etc
- One way to search in this case is to map these object features into points in a multi-dimensional space
- Another approach is to have a distance function for objects and then use a distance based index
- The main mapping methods form three main classes:
  - R^*-trees and the rest of the R-tree family,
  - linear quadtrees,
  - grid-files

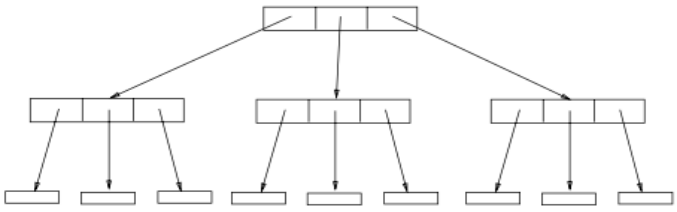
- The R-tree-based methods seem to be most robust for higher dimensions
- The R-tree represents a spatial object by its minimum bounding rectangle (MBR)
- Data rectangles are grouped to form parent nodes, which are recursively grouped, to form grandparent nodes and, eventually, a tree hierarchy
- **Disk pages** are consecutive byte positions on the surface of the disk that are fetched with one disk access
- The goal of the insertion, split, and deletion routines is to give trees that will have good clustering

- Figure below illustrates data rectangles (in black), organized in an R-tree with fanout 3



Multi-dimensional Search

- A range query specifies a region of interest, requiring all the data regions that intersect it
- To answer this query, we first retrieve a superset of the qualifying data regions:
  - We compute the MBR of the query region, and then we recursively descend the R-tree, excluding the branches whose MBRs do not intersect the query MBR
  - Thus, the R-tree will give us quickly the data regions whose MBR intersects the MBR of the query region
- The retrieved data regions will be further examined for intersection with the query region
- The data structure of the R-tree for the previous figure is (fanout = 3)



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.

The Web

World Wide Web, which is also known as a Web, is a collection of websites or web pages stored in web servers and connected to local computers through the internet. These websites contain text pages, digital images, audios, videos, etc. Users can access the content of these sites from any part of the world over the internet using their devices such as computers, laptops, cell phones, etc. The WWW, along with internet, enables the retrieval and display of text and media to your device.

The building blocks of the Web are web pages which are formatted in HTML and connected by links called "hypertext" or hyperlinks and accessed by HTTP. These links are electronic connections that link related pieces of information so that users can access the desired information quickly. Hypertext offers the advantage to select a word or phrase from text and thus to access other pages that provide additional information related to that word or phrase.

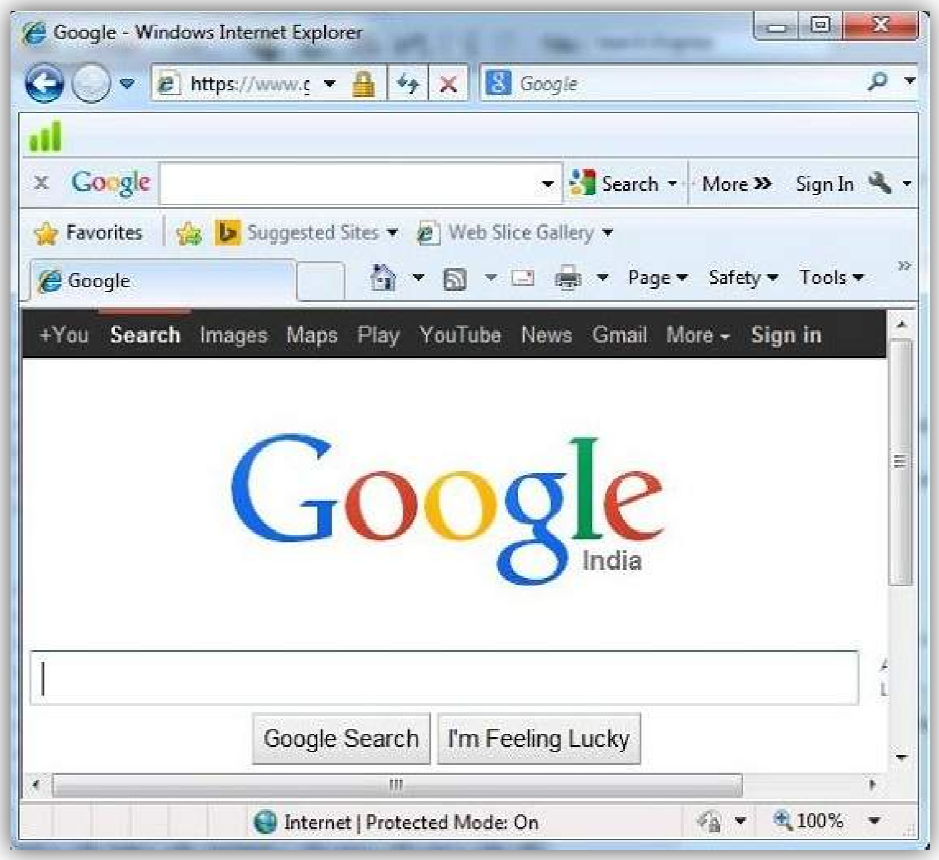
A web page is given an online address called a Uniform Resource Locator (URL). A particular collection of web pages that belong to a specific URL is called a website, e.g., *www.facebook.com*, *www.google.com*, etc. So, the World Wide Web is like a huge electronic book whose pages are stored on multiple servers across the world.

SEARCH ENGINE ARCHITECTURES

Components Of A Search Engine

Search Engine refers to a huge database of internet resources such as web pages, newsgroups, programs, images etc. It helps to locate information on World Wide Web.

User can search for any information by passing query in form of keywords or phrase. It then searches for relevant information in its database and return to the user.



Search Engine Components

Generally there are three basic components of a search engine as listed below:

1. Web Crawler
2. Database
3. Search Interfaces

Web crawler

It is also known as **spider** or **bots.** It is a software component that traverses the web to gather information.

Database

All the information on the web is stored in database. It consists of huge web resources.

Search Interfaces

This component is an interface between user and the database. It helps the user to search through the database.

Search Engine Working

Web crawler, database and the search interface are the major component of a search engine that actually makes search engine to work. Search engines make use of Boolean expression AND, OR, NOT to restrict and widen the results of a search. Following are the steps that are performed by the search engine:

- The search engine looks for the keyword in the index for predefined database instead of going directly to the web to search for the keyword.
- It then uses software to search for the information in the database. This software component is known as web crawler.
- Once web crawler finds the pages, the search engine then shows the relevant web pages as a result. These retrieved web pages generally include title of page, size of text portion, first several sentences etc.

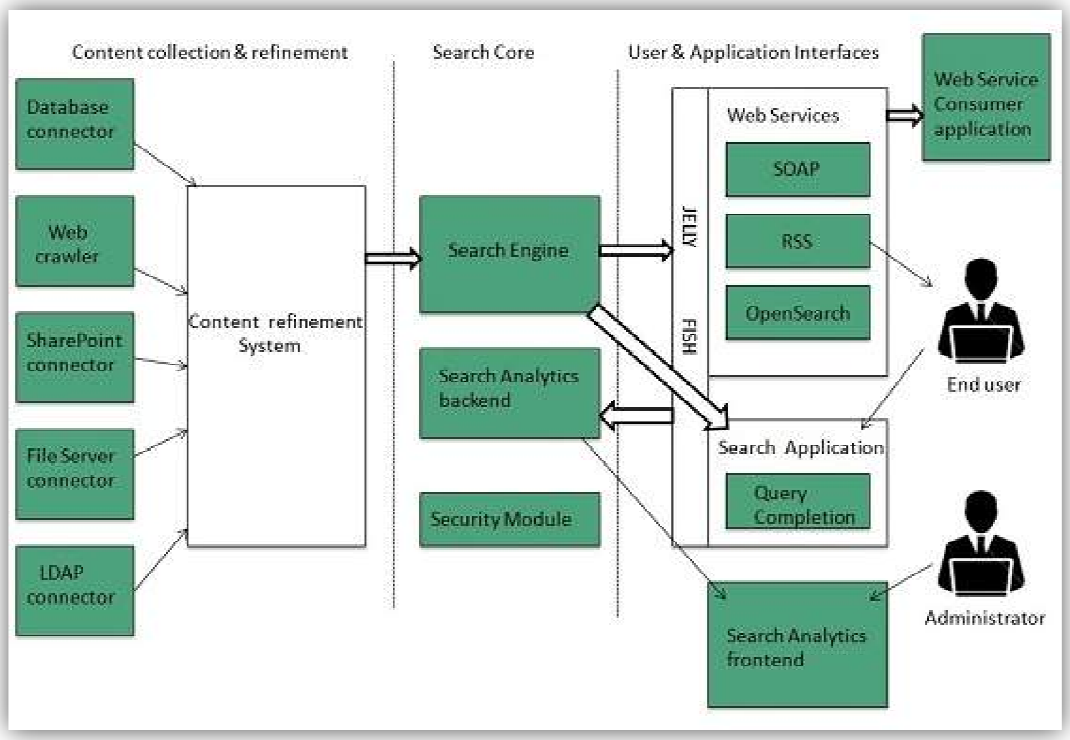
These search criteria may vary from one search engine to the other. The retrieved information is ranked according to various factors such as frequency of keywords, relevancy of information, links etc.

- User can click on any of the search results to open it.

Architecture

The search engine architecture comprises of the three basic layers listed below:

- Content collection and refinement.
- Search core
- User and application interfaces



Search Engine Processing

Indexing Process

Indexing process comprises of the following three tasks:

- Text acquisition
- Text transformation
- Index creation

Text acquisition

It identifies and stores documents for indexing.

Text Transformation

It transforms document into index terms or features.

Index Creation

It takes index terms created by text transformations and create data structures to support fast searching.

Query Process

Query process comprises of the following three tasks:

- User interaction
- Ranking
- Evaluation

User interaction

It supports creation and refinement of user query and displays the results.

Ranking

It uses query and indexes to create ranked list of documents.

Evaluation

It monitors and measures the effectiveness and efficiency. It is done offline.

CLUSTER BASED ARCHITECTURE

- ✓ The task for the retrieval system is to match the query against clusters of documents instead of individual documents, and rank clusters based on their similarity to the query.
- ✓ Any document from a cluster that is ranked higher is considered more likely to be relevant than any document from a cluster ranked lower on the list.
- ✓ This is in contrast to most other cluster search methods that use clusters primarily as a tool to identify a subset of documents that are likely to be relevant, so that at the time of retrieval, only those documents will be matched to the query.
- ✓ This approach has been the most common for cluster-based retrieval.

- ✓ The second approach to cluster-based retrieval is to use clusters as a form of document smoothing.
- ✓ Previous studies have suggested that by grouping documents into clusters, differences between representations of individual documents are, in effect, smoothed out.

Current search engines use a massive parallel and cluster-based architecture. Due to the large size of the document collection, the inverted index does not fit in a single computer and must be distributed across the computers of a cluster. The large volume of queries implies that the basic architecture must be replicated in order to handle the overall query load, and that each cluster must handle a subset of the query load. In addition, as queries originate from all around the world and Internet latency is appreciable across continents, cluster replicas are maintained in different geographical locations to decrease answer time. This allows search engines to be fault-tolerant in most typical worst-case scenarios, such as power outages or natural disasters.

There are many crucial details to be carefully addressed in this type of architecture:

1. It is particularly important to achieve a good balance between the internal (answering queries and indexing) and external (crawling) activities of the search engine. This is achieved by assigning dedicated clusters to crawling, to document serving, to indexing, to user interaction, to query processing, and even to the generation of the result pages.
2. In addition, a good load balancing among the different clusters needs to be maintained. This is achieved by specialized servers called (quite trivially) load balancers.
3. Finally, since hardware breaks often, fault tolerance is handled at the software level. Queries are routed to the most adequate available cluster and CPUs and disks are routinely replaced upon failure, using inexpensive exchangeable hardware components.

Figure 11.7 shows a generic search cluster architecture with its key components.

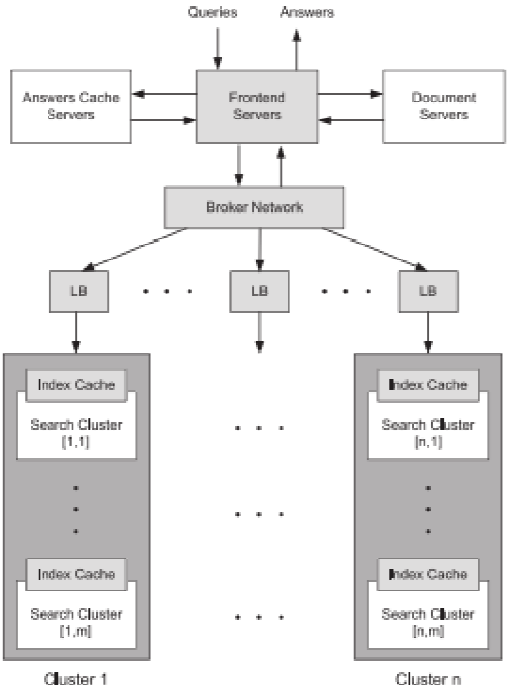


Figure 11.7: Cluster-based architecture for the search module based on [484]. Each cluster contains an index of the whole collection, *i.e.*, the index is partitioned among the m servers in the cluster. The n clusters are used to produce n replicas of the whole index.

The front-end servers receive queries and process them right away if the answer is already in the "answer cache" servers. Otherwise they route the query to the search clusters through a hierarchical broker network. The exact topology of this network can vary but basically, it should be designed to balance traffic so as to reach the search clusters as fast as possible. Each search cluster includes a load balancing server (LB in the figure) that routes the query to all the servers in one replica of the search cluster. In this figure, we show an index partitioned into n clusters with m replicas. Although partitioning the index into a single cluster is conceivable, it is not recommended as the cluster would turn out to be very large and consequently suffer from additional management and fault tolerance problems.

Each search cluster also includes an index cache, which is depicted at the top, as a flat rectangle. The broker network merges the results coming from the

search clusters and sends the merged results to the appropriate front-end server that will use the right document servers to generate the full results pages, including snippet and other search result page artifacts. This is an example of a more general trend to consider a whole data center as a computer.

DISTRIBUTED ARCHITECTURES

There exist several variants of the crawler-indexer architecture and we describe here the most important ones. Among them, the most significant early example is Harvest.

Harvest

Harvest uses a distributed architecture to gather and distribute data, which is more efficient than the standard Web crawler architecture. The main drawback is that Harvest requires the coordination of several Web servers. Interestingly, the Harvest distributed approach does not suffer from some of the common problems of the crawler-indexer architecture, such as:

- increased servers load caused by the reception of simultaneous requests from different crawlers,
- increased Web traffic, due to crawlers retrieving entire objects, while most content is not retained eventually, and
- lack of coordination between engines, as information is gathered independently by each crawler.

Avoiding these issues was achieved by introducing two main components in the architecture: gatherers and brokers. A gatherer collects and extracts indexing information from one or more Web servers. Gathering times are defined by the system and are periodic (i.e., there are harvesting times as the name of the system suggests). A broker provides the indexing mechanism and the query interface to the data gathered. Brokers retrieve information from one or more gatherers or other brokers, updating incrementally their indexes. Depending on the configuration of gatherers and brokers, different improvements on server load and network traffic

can be achieved. For example, a gatherer can run on a Web server, generating no external traffic for that server. Also, a gatherer can send information to several brokers, avoiding work repetition. Brokers can also filter information and send it to other brokers. This design allows the sharing of work and information in a very flexible and generic manner. An example of the Harvest architecture is shown in Figure 11.9.

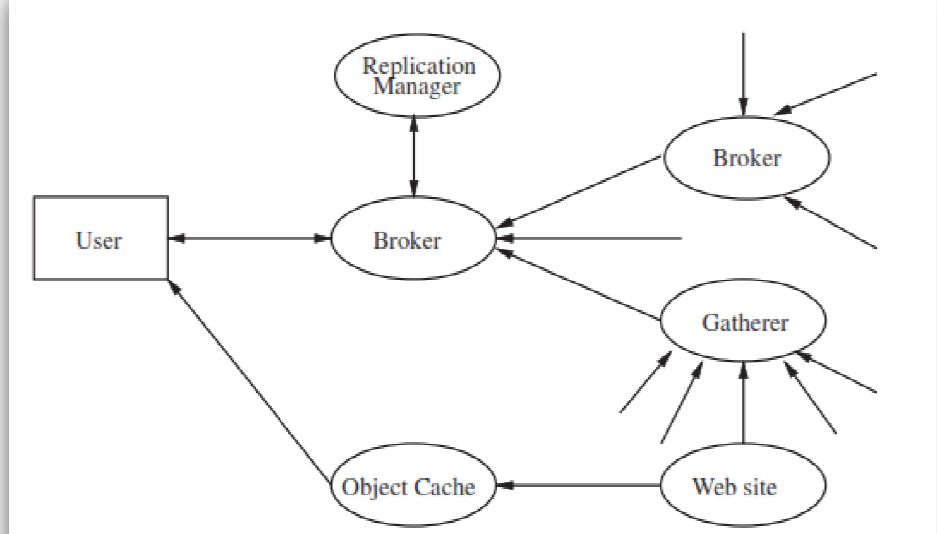


Figure 11.9: Harvest architecture.

One of the goals of Harvest is to build topic-specific brokers, focusing the index contents and avoiding many of the vocabulary and scaling problems of generic indexes. Harvest includes a dedicated broker that allows other brokers to register information about gatherers and brokers. This is mostly useful for identifying an appropriate broker or gatherer when building a new system. The Harvest architecture also provides replicators and object caches. A replicator can be used to replicate servers, enhancing user-base scalability. For example, the registration broker can be replicated in different geographic regions to allow faster access. Replication can also be used to divide the gathering process among many Web servers. Finally, the object cache reduces network and server load, as well as response latency when accessing Web pages.

SEARCH ENGINE RANKING

Ranking is the hardest and most important function search engines have to execute. A first challenge is to devise an adequate evaluation process that allows judging the efficacy of a ranking, in terms of its relevance to the users. Without such evaluation process it is close to impossible to fine tune the ranking function, which basically prevents achieving high quality results. There are many possible evaluation techniques and measures. We cover this topic in the context of the Web, paying particular attention to the exploitation of user's clicks.

A second critical challenge is the identification of quality content in the Web. Evidence of quality can be indicated by several signals such as domain names (i.e., .edu is a positive signal as content originating from academic institutions is more likely to be reviewed), text content and various counts (such as the number of word occurrences), links (like Page Rank), and Web page access patterns as monitored by the search engine. Indeed, as mentioned before, clicks are a key element of quality. The more traffic a search engine has, the more signals it will have available. Additional useful signals are provided by the layout of the Web page, its title, metadata, font sizes, as discussed later.

The economic incentives of the current advertising based business model adopted by search engines have created a third challenge, avoiding Web spam. Spammers in the context of the Web are malicious users who try to trick search engines by artificially inflating the signals mentioned in the previous paragraph. This can be done, for instance, by repeating a term in a page a great number of times, using link farms, hiding terms from the users but keeping them visible to the search engines through weird coloring tricks, or even for the most sophisticated ones, deceiving Javascript code. Finally, the fourth issue lies in defining the ranking function and computing it (which is different from evaluating its quality as mentioned above). While it is fairly difficult to compare different search engines as they evolve and operate on different Web corpora, leading search engines have to

constantly measure and compare themselves, each one using its own measure, so as to remain competitive.

Ranking Signals

We distinguish among different types of signals used for ranking improvements according to their origin, namely content, structure, or usage, as follows. Content signals are related to the text itself, to the distributions of words in the documents as has been traditionally studied in IR. The signal in this case can vary from simple word counts to a full IR score such as BM25. They can also be provided by the layout, that is, the HTML source, ranging from simple format indicators (more weight given to titles/headings) to sophisticated ones such as the proximity of certain tags in the page. Structural signals are intrinsic to the linked structure of the Web. Some of them are textual in nature, such as anchor text, which describe in very brief form the content of the target Web page. In fact, anchor text is usually used as surrogate text of the linked Web page. That implies that Web pages can be found by searching the anchor texts associated with links that point to them, even if they have not been crawled. Other signals pertain to the links themselves, such as the number of in-links to or outlinks from a page.

The next set of signals comes from Web usage. The main one is the implicit feedback of the user through clicks. In our case the main use of clicks are the ones in the URLs of the results set.

LINK-BASED RANKING

Given that there might be thousands or even millions of pages available for any given query, the problem of ranking those pages to generate a short list is probably one of the key problems of Web IR; one that requires some kind of relevance estimation. In this context, the number of hyperlinks that point to a page provides a measure of its popularity and quality. Further, many links in common among pages and pages referenced by a same page are often indicative of page relations with potential value for ranking purposes. Next, we present several

examples of ranking techniques that exploit links, but differ on whether they are query dependent or not.

Early Algorithms

- TF-IDF
- Boolean spread, vector spread, and most-cited
- WebQuery

HITS

A better idea is due to Kleinberg and used in HITS (Hypertext Induced Topic Search). This ranking scheme is query-dependent and considers the set of pages S that point to or are pointed by pages in the answer. Pages that have many links pointing to it in S are called authorities because they are susceptible to contain authoritative and thus, relevant content. Pages that have many outgoing links are called hubs and are susceptible to point to relevant similar content. A positive two-way feedback exists: better authority pages come from incoming edges from good hubs and better hub pages come from outgoing edges to good authorities. Let H(b) and A(p) be the hub and authority values of page p. These values are defined such that the following equations are satisfied for all pages p:

$$H(p) = \sum_{u \in S \mid p \to u} A(u) , \qquad A(p) = \sum_{v \in S \mid v \to p} H(v) \qquad (11.5)$$

where H(p) and A(p) for all pages are normalized (in the original paper, the sum of the squares of each measure is set to one). These values can be determined through an iterative algorithm, and they converge to the principal eigenvector of the link matrix of S. In the case of the Web, to avoid an explosion on the size of S, a maximal number of pages pointing to the answer can be defined. This technique does not work with non-existent, repeated, or automatically generated links. One solution is to weigh each link based on the surrounding content. A second problem is that of topic diffusion, because as a consequence of link weights, the result set might include pages that are not directly related to the query (even if they have got

high hub and authority values). A typical case of this phenomenon is when a particular query is expanded to a more general topic that properly contains the original answer. One solution to this problem is to associate a score with the content of each page, like in traditional IR ranking, and combine this score with the link weight. The link weight and the page score can be included in the previous formula multiplying each term of the summation. Experiments show that the recall and precision for the first ten results increase significantly. The appearance order of the links on the Web page can also be used by dividing the links into subgroups and using the HITS algorithm on those subgroups instead of the original Web pages. In Table 11.2, we show the exponent of the power law of the distribution for authority and hub values for different countries of the globe adapted from.

Country	PageRank	HITS	
		Hubs	Auth.
Brazil	1.83	2.9	1.83
Chile	1.85	2.7	1.85
Greece	1.83	2.6	1.83
South Korea	1.83	3.7	1.83
Spain	1.96	n/a	n/a

Table 11.2: Summary of power-law exponents for link-based measures of various countries.

SIMPLE RANKING FUNCTIONS

The simplest ranking scheme consists of using a global ranking function such as PageRank. In that case, the quality of a Web page is independent of the query and the query only acts as a document filter. That is, for all Web pages that satisfy a query, rank them using their PageRank order.

A more elaborated ranking scheme consists of using a linear combination of different relevance signals. For example, combining textual features, say BM25, and link features, such as PageRank. To illustrate, consider the pages p that satisfy query Q. Then, the rank score R(p, Q) of page p with regard to query Q can be computed as:

$$R(p,Q) = \alpha\ BM25(p,Q)\ +\ (1-\alpha)PR(p) \qquad (11.7)$$

Further, R(p, Q)=0 if p does not satisfies Q. If we assume that all the functions are normalized and a ∈ [0, 1], then R(p, Q) ∈ [0, 1]. Notice that this linear function is convex in a. Also, while the first term depends on the query, the second term does not. If a = 1, we have a pure textual ranking, which was the typical case in the early search engines. If a = 0, we have a pure link-based ranking that is also independent of the query. Thus, the order of the pages is known in advance for pages that do contain q. We can tune the value of a experimentally using labeled data as ground truth or click through data. In fact, a might even be query dependent. For example, for navigational queries a could be made smaller than for informational queries.

Early work on combining text-based and link-based rankings was published by Silva. The authors used a Bayesian network to combine the different signals and showed that the combination leads to far better results than those produced by any of the combining ranking functions in isolation. Subsequent research by Calado discussed the efficacy of a global link-based ranking versus a local link based ranking for computing Web results. The local link-based ranking for a page p is computed considering only the pages that link to and are linked by page p. The authors compare results produced by a combination of a text-based ranking (Vector model) with global HITS, local HITS, and global PageRank, to conclude that a global link-based ranking produces better results at the top of the ranking, while a local link-based ranking produces better results later in the ranking.

LEARNING TO RANK

A rather distinct approach for computing a Web ranking is to apply machine learning techniques for learning to rank. For this, one can use their favorite machine learning algorithm, fed with training data that contains ranking information, to “learn” a ranking of the results, analogously to the supervised

algorithms for text classification. The loss function to minimize in this case is the number of mistakes done by the learned algorithm, which is similar to counting the number of misclassified instances in traditional classification. The evaluation of the learned ranking must be done with another data set (which also includes ranking information) distinct from the one used for training. There exist three types of ranking information for a query Q, that can be used for training:

- point wise: a set of relevant pages for Q.
- Pair wise: a set of pairs of relevant pages indicating the ranking relation between the two pages.

  That is, the pair [p1>p2], implies that the page p1 is more relevant than p2.
- List-wise: a set of ordered relevant pages: p1>p2⋯ >pm.

In any case, we can consider that any page included in the ranking information is more relevant than a page without information, or we can maintain those cases undefined. Also, the ranking information does not need to be consistent (for example, in the pair wise case). The training data may come from the so-called "editorial judgments" made by people or, better, from click through data. Given that users' clicks reflect preferences that agree in most cases with relevance judgments done by human assessors, one can consider using click through information to generate the training data. Then, we can learn the ranking function from click-based preferences. That is, if for query Q, p has more clicks than p2 , then [p1_ p21].

One approach for learning to rank from clicks using the pair wise approach is to use support vector machines (SVMs), to learn the ranking function. In this case, preference relations are transformed into inequalities among weighted term vectors representing the ranked documents. These inequalities are then translated into an SVM optimization problem, whose solution computes optimal weights for the document terms. This approach proposes the combination of different retrieval functions with different weights into a single ranking function.

The point wise approach solves the problem of ranking by means of regression or classification on single documents, while the pairwise approach transforms ranking into a problem of classification on document pairs. The advantage of these two approaches is that they can make use of existing results in regression and classification. However, ranking has intrinsic characteristics that cannot be always solved by the latter techniques. The list wise approach tackles the ranking problem directly, by adopting list wise loss functions, or directly optimizes IR evaluation measures such as average precision. However, this case is in general more complex. Some authors have proposed to use a multi-variant function, also called relational ranking function, to perform list wise ranking, instead of using a single-document based ranking function.

QUALITY EVALUATION

To be able to evaluate quality, Web search engines typically use human judgments that indicate which results are relevant for a given query, or some approximation of a "ground truth" inferred from user's clicks, or finally a combination of both, as follows.

Precision at 5, 10, 20

One simple approach to evaluate the quality of Web search results is to adapt the standard precision-recall metrics to the Web. For this, the following observations are important:

on the Web

it is almost impossible to measure recall, as the number of relevant pages for most typical queries is prohibitive and ultimately unknown. Thus, standard precision-recall figures cannot be applied directly.

Most Web users inspect only the top 10 results and it is relatively uncommon that a user inspects answers beyond the top 20 results. Thus, evaluating the quality of Web results beyond position 20 in the ranking is not indicated as does not reflect common user behavior. Since Web queries tend to be short and vague,

human evaluation of results should be based on distinct relevance assessments for each query-result pair. For instance, if three separate assessments are made for each query-result pair, we can consider that the result is indeed relevant to the query if at least two of the assessments suggest so. The compounding effect of these observations is that

(a) precision of Web results should be measured only at the top positions in the ranking, say P@5, P@10, and P@20 and

(b) each query-result pair should be subjected to 3-5 independent relevant assessments.

Click-through Data as an Evaluation Metric

One major advantage of using click through data to evaluate the quality of answers derives from its scalability. Its disadvantage is that it works less well in smaller corpora, such as countries with little Web presence, Intranet search, or simply in the long tail of queries. Note that users' clicks are not used as a binary signal but in significantly more complex ways such as considering whether the user remained a long time on the page it clicked (a good signal) or jumped from one result to the other (a signal that nothing satisfying was found). These measures and their usage are complex and kept secret by leading search engines.

Evaluating the Quality of Snippets

A related problem is to measure the quality of the snippets in the results. Search snippets are the small text excerpts associated with each result generated by a search engine. They provide a summary of the search result and indicate how it is related to the query (by presenting query terms in boldface, for instance). This provides great value to the users who can quickly inspect the snippets to decide which results are of interest.

Web Spam

The Web contains numerous profit-seeking ventures, so there is an economic incentive from Web site owners to rank high in the result lists of search engines.

All deceptive actions that try to increase the ranking of a page in search engines are generally referred to as Web spam or spamdexing (a portmanteau of "spamming" and "index"). The area of research that relates to spam fighting is called Adversarial Information Retrieval, which has been the object of several publications and workshops.

SEARCH ENGINE RANKING

Assigning Identifiers to Documents

Document identifiers are usually assigned randomly or according to the ordering with which URLs are crawled. Numerical identifiers are used to represent URLs in several data structures. In addition to inverted lists, they are also used to number nodes in Web graphs and to identify documents in search engines repositories.

It has been shown in the literature that a careful ordering of documents leads to an assignment of identifiers from which both index and Web graph storing methods can benefit. Also an assignment based on a global ranking scheme may simplify the ranking of answers (see section 11.5.3). Regarding the compression of inverted lists, a very effective mapping can be obtained by considering the sorted list of URLs referencing the Web documents of the collection. Assigning identifiers in ascending order of lexicographically sorted URLs improves the compression rate. The hypothesis that is empirically validated by Silvestri is that documents sharing correlated and discriminant terms are very likely to be hosted by the same site and will therefore also share a large prefix of their URLs. Experiments validate the hypothesis since compression rates can be improved up to 0.4 by using the URL sorting technique. Furthermore, sorting a few million URLs takes only tens of seconds and takes only a few hundreds megabytes of main memory.

search engine user interaction

Web search engines target hundreds of millions of users, most of which have very little technical background. As a consequence, the design of the interface has been heavily influenced by a extreme simplicity rule, as follows.

Extreme Simplicity Rule. The design of the user search experience, i.e., the patterns of user interaction with the search engine, must assume that the users have only minimal prior knowledge of the search task and must require as little learning on their part as possible. In fact, users are expected to read the "user manual" of a new refrigerator or DVD player more often than the help page of their favorite search engine. One immediate consequence of this state of affairs is that a user that does not "get it", while interacting with a search engine, is very likely to attempt to solve the problem by simply switching to another search engine. In this context, extreme simplicity has become the rule for user interaction in Web search.

In this section, we describe typical user interaction models for the most popular Web Search engines of today, their recent innovations, and the challenges they face to abide by this extreme simplicity rule. But we revisit them here in more depth, in the context of the Web search experience offered by major players such as Ask.com, Bing, Google and Yahoo! Search. We do not discuss here "vertical" search engines, i.e., search engines restricted to a specific domains of knowledge such as Yelp or Netflix, or major search engines verticals, such as Google Image Search or Yahoo! Answers.

The Search Rectangle Paradigm

Users are now accustomed with specifying their information needs by formulating queries in a search "rectangle". This interaction mode has become so popular that many Web homepages now feature a rectangle search box, visible in prominent area of the site, even if the supporting search technology is provided by a third partner. To illustrate, Figure 11.10 displays the search rectangle of the Ask, Bing, Google, and Yahoo! search engines. The rectangle design has remained

pretty much stable for some engines such as Google, whose main homepage has basically not changed in the last ten years. Others like Ask and Bing allow a more fantasy oriented design with colorful skins and beautiful photos of interesting places and objects (notice, for instance, the Golden Gate bridge background of Ask in Figure 11.10).

Figure 11.10: The search rectangle as the central user interface component of four major search engines (from Ask, Bing, Google and Yahoo! Search, respectively Ask screenshot, © IAC Search & Media, Inc. 2010. All rights reserved. ASK.COM, ASK JEEVES, the ASK logo, the ASK JEEVES logo and other trade marks appearing on the Ask.com and Ask Jeeves websites are property of IAC Search & Media, Inc. and/or its licensors.).

Despite these trends, the search rectangle remains the center piece of the action in all engines. While the display of a search rectangle at the center of the page is the favored layout style, there are alternatives:

Some Web portals embed the search rectangle in a privileged area of the homepage. Examples of this approach are provided by yahoo.com or aol.com.

• Many sites include an Advanced Search page, which provides the users with a form composed of multiple search "rectangles" and options (rarely used).

• The search toolbars provided by most search engines as a browser plug-in, or built-in in browsers like Firefox, can be seen as a leaner version of the central search rectangle. By being accessible at all times, they represent a more convenient alternative to the homepage rectangle, yet their requirement of a download for installation prevents wider adoption. Notice that, to compensate this overhead, many search engines negotiate costly OEM deals with PC distributors/manufacturers to preinstall their toolbars.

• The "ultimate" rectangle, introduced by Google's Chrome "omnibox", merges the functionality of the address bar with that of the search box. It becomes then

responsibility of the browser to decide whether the text inputted by the user aims at navigating to a given site or at conducting a search. Prior to the introduction of the Omnibox, Firefox already provided functionality to recognize that certain words cannot be part of a URL and thus, should be treated as part of a query. In these cases, it would trigger Google's "I feel lucky" function to return the top search result. Interestingly enough, this Firefox feature is customizable, allowing users to trigger search engines other than Google or to obtain a full page of results.

The Search Engine Result Page

Result Presentation -The Basic Layout

The classic presentation style of the Search Engine Result Page, often referred to as SERP, consists of a list of "organic" or "algorithmic" results, that appear on the left hand side of the results page, as well as paid/sponsored results (ads) that appear on the right hand side. Additionally, the most relevant paid results might appear on top of the organic results in the North area, as illustrated in Figure 11.12. By default, most search engines show ten results in the first page, even though some engines, such as Bing, allow users to customize the number of results to show on a page. Figure 11.12 illustrates the layout generally adopted by most engines, with common but not uniformly adopted locations being indicated by a dotted line framed box.

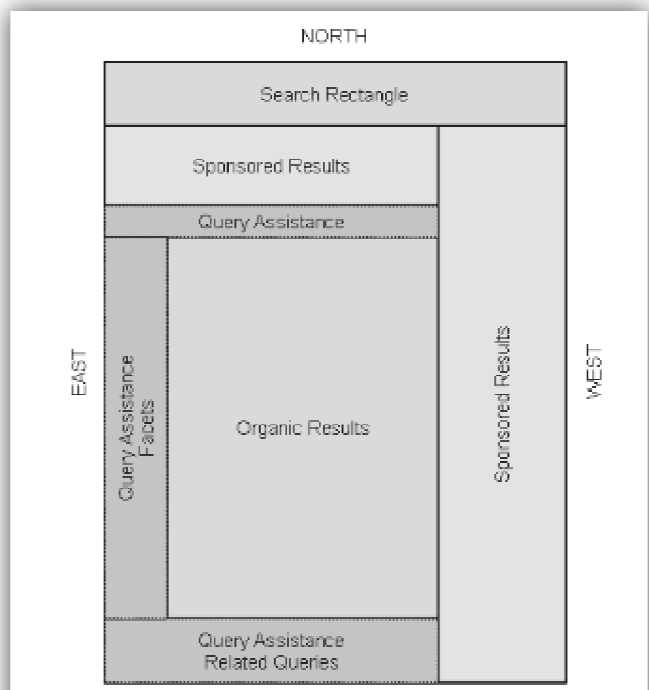


Figure 11.12: A typical SERP layout.

These engines might differ on small details like the "query assistance" features, which might appear in the North, South or West region of the page, the position of the navigational tools, which might or might not be displayed on the West region, or the position of spelling correction recommendations, which might appear before of after the sponsored results in the North region. Search engines constantly experiment with small variations of layout, and it might be the case that drastically different layouts be adopted in the future, as this is a space that calls for innovative features. To illustrate, Cuil introduced a radically different layout that departs from the one dimensional ranking, but this is more the exception than the rule. In contrast, search properties other than the main engines, commonly adopt distinct, such as the image search in both Google and Yahoo! as an example, or Google Ads search results, all of which display results across several columns. In this section, we focus exclusively on the organic part of search results. We will refer to them from now as "search results", note the distinction with paid/sponsored search results.

Major search engines use a very similar format to display individual results composed basically of

(a) a title shown in blue and underlined,

(b) a short snippet consisting of two or three sentences extracted from the result page, and

(c) a URL, that points to the page that contains the full text. In most cases, titles can be extracted directly from the page.

When a page does not have a title, anchor texts pointing to it can be used to generate a title.

More Structured Results

In addition to results fed by the main Web corpus, search engines include additional types of results, as follows.

Oneboxes" results. These are very specific results, produced in response to very precise queries, that are susceptible of having one unique answer. They are displayed above regular Web results, due to their high relevance, and in a distinct format. Oneboxes are triggered by specific terms in the user's query that indicate a clear intent. They aim at either exposing the answer directly or exposing a direct link to the answer, which provides the ultimate search experience but can be achieved only in very specific cases, i.e., when relevance is guaranteed and the answer is short and unambiguous.

As an example, both Google and Yahoo! Search support a weather onebox, which is triggered by entering "weather <a location>" (see example in Figure 11.13).

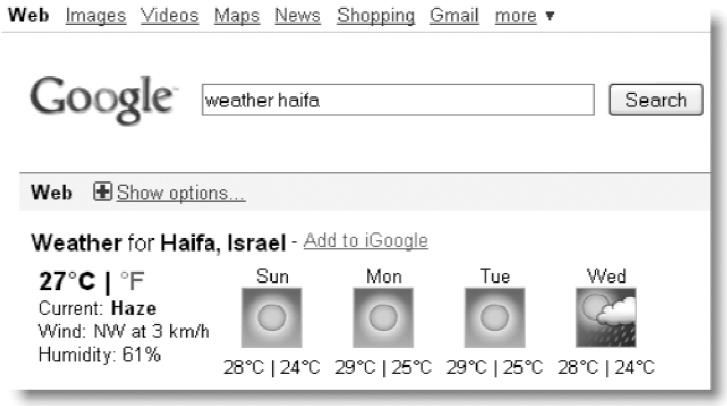


Figure 11.13: Example of weather onebox from Google, where is no need to click to get the full answer.

Universal search results: Most Web search engines offer, in addition to core Web search, other properties, such as Images, Videos, Products, Maps, which come with their own vertical search. While users can go directly to these properties to conduct corpus-specific searches, the "universal" vision states that users should not have to specify the target corpus. The engine should guess their intent and automatically return results from the most relevant sources when appropriate. The key technical challenge here is to select these sources and to decide how many results from each sources to display.

BROWSING

In this section, we cover browsing as an additional discovery paradigm, with special attention to Web directories. Browsing is mostly useful when users have no idea of how to specify a query (which becomes rarer and rarer in the context of the global Web), or when they want to explore a specific collection and are not sure of its scope. Nowadays, browsing is no longer the discovery paradigm of choice on the Web. Despite that, it can still be useful in specific contexts such as that of an Intranet or in vertical domains, as we now discuss. In the case of browsing, users are willing to invest some time exploring the document space, looking for interesting or even unexpected references. Both with browsing and searching, the user is pursuing discovery goals. However, in search, the user's goal

is somewhat crisper. In contrast, with browsing, the user's needs are usually broader. While this distinction is not valid in all cases, we will adopt it here for the sake of simplicity. We first describe the three types of browsing namely, flat, structure driven (with special attention to Web directories), and hypertext driven. Following, we discuss attempts at combining searching and browsing in a hybrid manner.

Flat Browsing

In flat browsing, the user explores a document space that follows a flat organization. For instance, the documents might be represented as dots in a two-dimensional plane or as elements in a single dimension list, which might be ranked by alphabetical or by any other order. The user then glances here and there looking for information within the visited documents. Note that exploring search results is a form of flat browsing. Each single document can also be explored in a flat manner via the browser, using navigation arrows and the scroll bar.

One disadvantage is that in a given page or screen there may not be any clear indication of the context the user is in. For example, while browsing large documents, users might lose track of which part of the document they are looking at. Flat browsing is obviously not available in the global Web due to its scale and distribution, but is still the mechanism of choice when exploring smaller sets. Furthermore, it can be used in combination with search for exploring search results or attributes. In fact, flat browsing conducted after an initial search allows identifying new keywords of interest. Such keywords can then be added to the original query in an attempt to provide better contextualization.

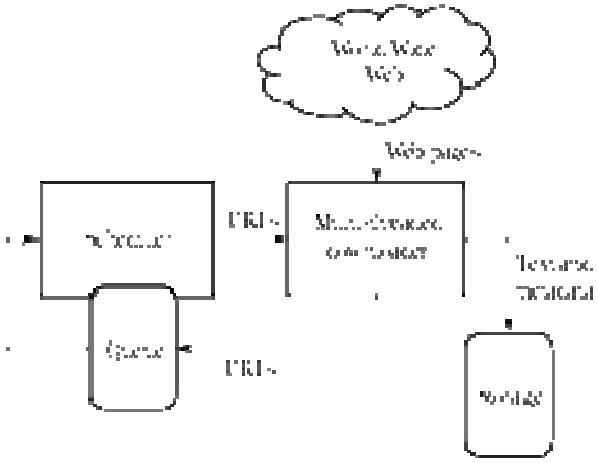
WEB CRAWLING

A **Web Crawler** is a software for downloading pages from the Web. Also known as Web Spider, Web Robot, or simply Bot.

A **Web crawler**, sometimes called a **spider** or **spiderwort** and often shortened to **crawler**, is an Internet bot that systematically browses the World

Wide Web, typically operated by search engines for the purpose of Web indexing (web spidering).

Web search engines and some other websites use Web crawling or spidering software to update their web content or indices of other sites' web content. Web crawlers copy pages for processing by a search engine, which indexes the downloaded pages so that users can search more efficiently. Crawlers consume resources on visited systems and often visit sites without approval. Issues of schedule, load, and "politeness" come into play when large collections of pages are accessed. Mechanisms exist for public sites not wishing to be crawled to make this known to the crawling agent. For example, including a robots.txt file can request bots to index only parts of a website, or nothing at all.



The number of Internet pages is extremely large; even the largest crawlers fall short of making a complete index. For this reason, search engines struggled to give relevant search results in the early years of the World Wide Web, before 2000. Today, relevant results are given almost instantly. Crawlers can validate hyperlinks and HTML code. They can also be used for web scraping and data-driven programming.

APPLICATIONS OF A WEB CRAWLER

A Web Crawler can be used to

- create an index covering broad topics (**general Web search**)
- create an index covering specific topics (**vertical Web search**)
- archive content (**Web archival**)
- analyze Web sites for extracting aggregate statistics (**Web characterization**)
- keep copies or replicate Web sites (**Web mirroring**)
- Web site analysis

Types of Web search

- **General Web search:** done by large search engines
- **Vertical Web search:** the set of target pages is delimited by a topic, a country or a language

Crawler for general Web search must balance coverage and quality.

- **Coverage**: It must scan pages that can be used to answer many different queries
- **Quality**: The pages should have high quality

Vertical Crawler: focus on a particular subset of the Web

✓ This subset may be defined geographically, linguistically, topically, etc. Examples of vertical crawlers

1.Shopbot: designed to download information from on-line shopping catalogs and provide an interface for comparing prices in a centralized way

2.News crawler: gathers news items from a set of pre-defined sources

3.Spambot: crawler aimed at harvesting e-mail addresses inserted on Web pages

Vertical search also includes segmentation by a data format. In this case, the crawler is tuned to collect only objects of a specific type, as image, audio, or video objects. Example

Feed crawler: checks for updates in RSS/RDF files in Web sites.

Focused crawlers: focus on a particular topic

Provides a more efficient strategy to avoid collecting more pages than necessary

A focused crawler receives as input the description of a topic, usually described by

- a driving query
- a set of example documents

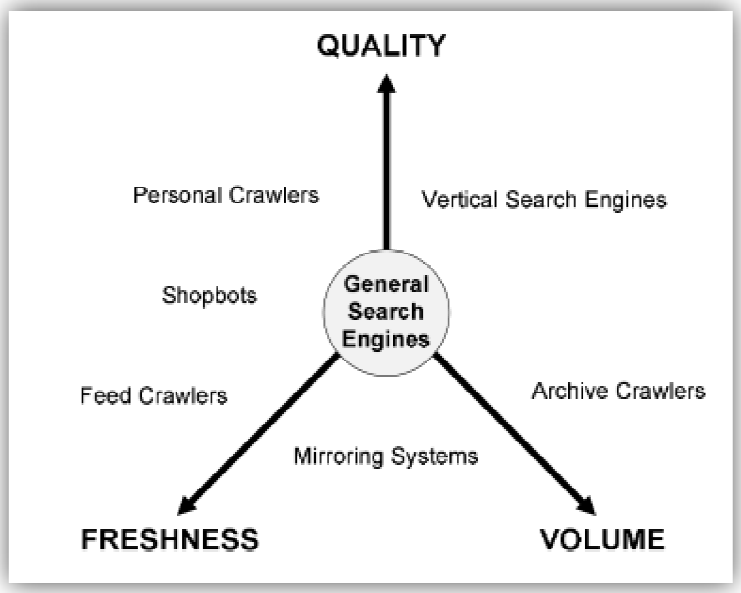
The crawler can operate in

batch mode, collecting pages about the topic periodically

on-demand, collecting pages driven by a user query

TAXONOMY

The crawlers assign different importance to issues such as **freshness**, **quality**, and **volume** The crawlers can be classified according to these three axes



A crawler would like to use all the available resources as much as possible crawling servers, Internet bandwidth). However, crawlers should also fulfill **politeness,** That is, a crawler cannot overload a Web site with HTTP requests. That implies that a crawler should wait a small delay between two requests to the same Web site Later we will detail other aspects of politeness

ARCHITECTURE AND IMPLEMENTATION

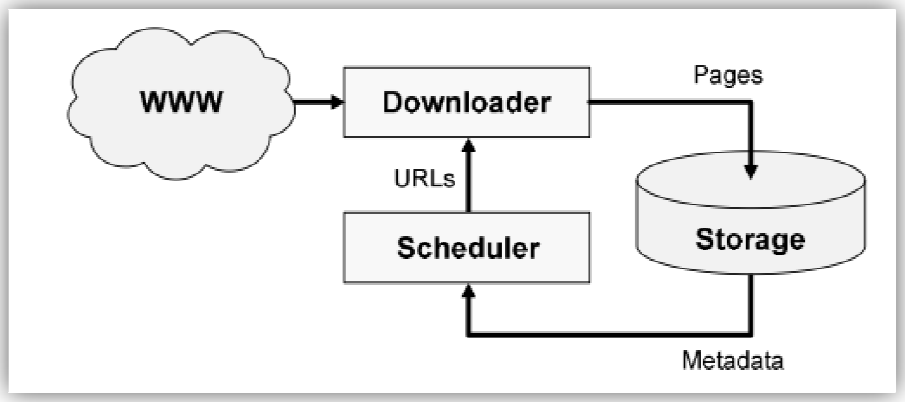
The crawler is composed of three main modules:

- downloader,
- storage, and
- scheduler

Scheduler: maintains a queue of URLs to visit

Downloader: downloads the pages

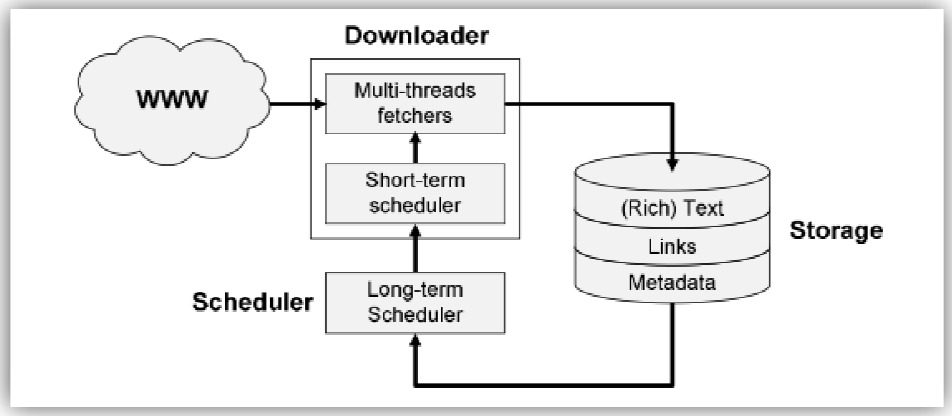
Storage: makes the indexing of the pages, and provides the scheduler with metadata on the pages retrieved.



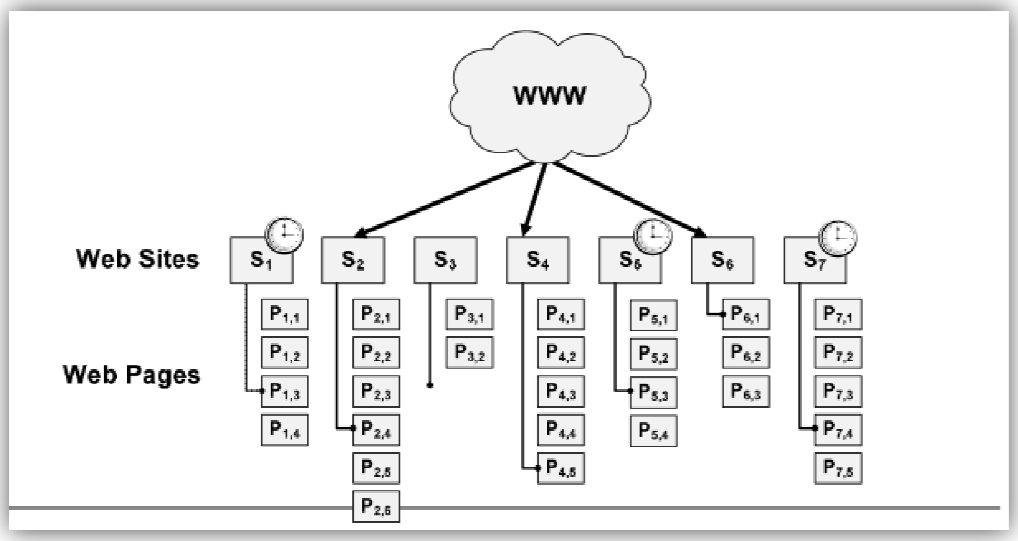
The scheduling can be further divided into two parts:

- **long-term scheduling**: decide which pages to visit next
- **short-term scheduling**: re-arrange pages to fulfill politeness

The storage can also be further subdivided into three parts: (rich) text, metadata, and links.



In the **short-term scheduler**, enforcement of the politeness policy requires maintaining several queues, one for each site, and a list of pages to download in each queue.



The implementation of a crawler involves many **practical issues.**

Most of them is due to the need to interact with many different systems

Example: How to download maintaining the traffic produced as uniform as possible?

The pages are from multiple sources DNS and Web server response times are highly variable Web server up-time cannot be taken for granted.

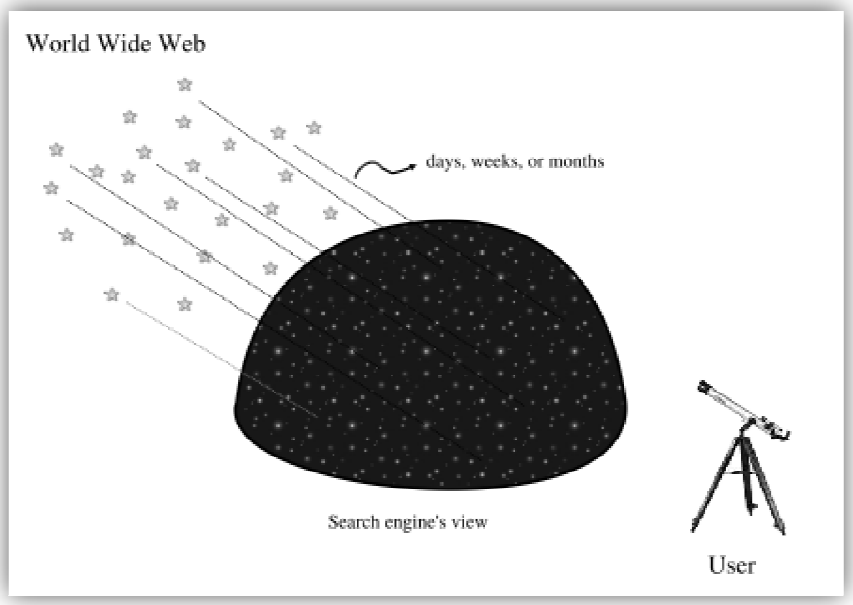
Other practical issues are related with:

types of Web pages, URL canonization, parsing, wrong implementation of HTML standards, and duplicates.

SCHEDULING ALGORITHMS

A Web crawler needs to balance various objectives that contradict each other It must download new pages and seek fresh copies of downloaded pages It must use network bandwidth efficiently avoiding to download bad pages However, the crawler cannot know which pages are good, without first downloading them To further complicate matters there is a huge amount of pages being added, changed and removed every day on the Web

Crawling the Web, in a certain way, resembles watching the sky in a clear night: the star positions that we see reflects the state of the stars at different times.



The simplest crawling scheduling is traversing Web sites in a breadth-first fashion. This algorithm increases the Web site coverage and is good to the politeness policy by not requesting many pages from a site in a row. However, can be useful to consider the crawler's behavior as a combination of a series of policies To illustrate, a crawling algorithm can be viewed as composed of three distinct policies

- **selection policy**: to visit the best quality pages, first
- **re-visit policy**: to update the index when pages change
- **politeness policy**: to avoid overloading Web sites

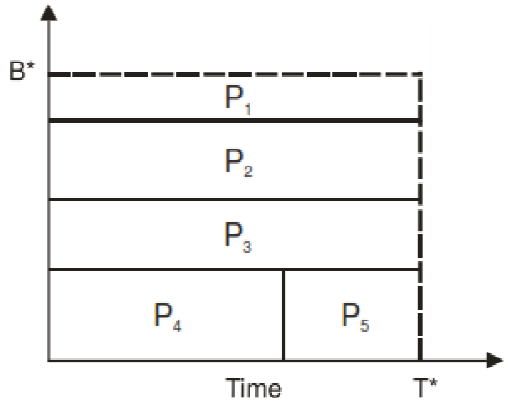
CRAWLING EVALUATION

The diagram below depicts an optimal Web crawling scenario for an hypothetical batch of five pages. The x-axis is time and the y-axis is speed, so the area of each page is its size (in bytes).

The downloader has maximum bandwidth B^* so the crawl can be completed in time:

$$T^* = \frac{\sum_i \text{size}(P_i)}{B^*}$$

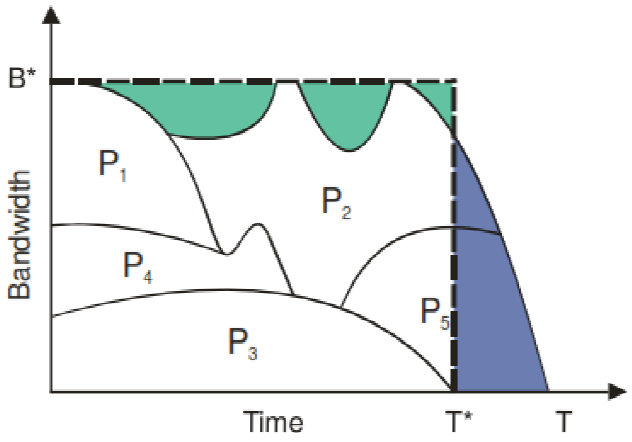
where $\text{size}(P_i)$ is the size of page P_i



Let us consider now a more realistic setting in which:

The speed of download of every page is variable and bounded by the effective bandwidth to a Web site (a fraction of the bandwidth that the crawler would like to use can be lost) Pages from the same site can not be downloaded right away one after the other (politeness policy).

- Under these assumptions, a different crawling timeline may occur, such as the one depicted on the figure below
- In the realistic case, the total time T is larger than the optimal case
- The bandwidth lost can be measured and is given by $B^* \times (T - T^*)$
- In the figure, green and blue areas are equal



By the end of the batch of pages, it is very likely that only a few hosts are active

Once a large fraction of the pages have been downloaded it is reasonable to stop the crawl, if only a few hosts remain at the end. Particularly, if the number of hosts remaining is very small then the bandwidth cannot be used completely.

A short-term scheduler may improve its finishing time by saturating the bandwidth usage, through the use of more threads However, if too many threads are used, the cost of switching control among threads becomes prohibitive The ordering of the pages can also be optimized by avoiding having a few sites to choose from at any point of the batch.

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.

RECOMMENDER SYSTEMS

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read.

"Item" is the general term used to denote what the system recommends to users. A RS normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for that specific type of item.

RSs are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number of alternative items that a Web site

In their simplest form, personalized recommendations are offered as ranked lists of items. In performing this ranking, RSs try to predict what the most suitable products or services are, based on the user's preferences and constraints. In order to complete such a computational task, RSs collect from users their preferences, which are either explicitly expressed, e.g., as ratings for products, or are inferred by interpreting user actions. For instance, a RS may consider the

navigation to a particular product page as an implicit sign of preference for the items shown on that page.

Depending on the recommendation approach, by the user's context and need, RSs generate recommendations using various types of knowledge and data about users, the available items, and previous transactions stored in customized databases. The user can then browse the recommendations. She may accept them or not and may provide, immediately or at a next stage, an implicit or explicit feedback. All these user actions and feedbacks can be stored in the recommender database and may be used for generating new recommendations in the next user-system interactions.

Recommender systems emerged as an independent research area in the mid-1990s. In recent years, the interest in recommender systems has dramatically increased, as the following facts indicate:
Recommender systems play an important role in such highly rated Internet sites as Amazon.com, YouTube, Netflix, Yahoo, Tripadvisor, Last.fm, and IMDb. Moreover many media companies are now developing and deploying RSs as part of the services they provide to their subscribers.

RECOMMENDER SYSTEMS FUNCTION

In the previous section we defined RSs as software tools and techniques providing users with suggestions for items a user may wish to utilize. Now we want to refine this definition illustrating a range of possible roles that a RS can play. First of all, we must distinguish between the role played by the RS on behalf of the service provider from that of the user of the RS. For instance, a travel recommender system is typically introduced by a travel intermediary (e.g., Expedia.com) or a destination management organization (e.g., Visitfinland.com) to increase its turnover (Expedia), i.e., sell more hotel rooms, or to increase the number of tourists to the destination [86]. Whereas, the user's primary motivations for accessing the

two systems is to find a suitable hotel and interesting events/attractions when visiting a destination.

In fact, there are various reasons as to why service providers may want to exploit this technology:

- Increase the number of items sold.
- Sell more diverse items.
- Increase the user satisfaction.
- Increase user fidelity.
- Better understand what the user wants.

DATA AND KNOWLEDGE SOURCES

RSs are information processing systems that actively gather various kinds of data in order to build their recommendations. Data is primarily about the items to suggest and the users who will receive these recommendations. But, since the data and knowledge sources available for recommender systems can be very diverse, ultimately, whether they can be exploited or not depends on the recommendation technique.

In general, there are recommendation techniques that are knowledge poor, i.e., they use very simple and basic data, such as user ratings/evaluations for items. Other techniques are much more knowledge dependent, e.g., using ontological descriptions of the users or the items, or constraints, or social relations and activities of the users. In any case, as a general classification, data used by RSs refers to three kinds of objects: items, users, and transactions, i.e., relations between users and items.

Items.

Items are the objects that are recommended. Items may be characterized by their complexity and their value or utility. The value of an item may be positive if the

item is useful for the user, or negative if the item is not appropriate and the user made a wrong decision when selecting it. We note that when a user is acquiring an item she will always incur in a cost, which includes the cognitive cost of searching for the item and the real monetary cost eventually paid for the item.

For instance, the designer of a news RS must take into account the complexity of a news item, i.e., its structure, the textual representation, and the time-dependent importance of any news item. But, at the same time, the RS designer must understand that even if the user is not paying for reading news, there is always a cognitive cost associated to searching and reading news items. If a selected item is relevant for the user this cost is dominated by the benefit of having acquired a useful information, whereas if the item is not relevant the net value of that item for the user, and its recommendation, is negative. In other domains, e.g., cars, or financial investments, the true monetary cost of the items becomes an important element to consider when selecting the most appropriate recommendation approach.

Items with low complexity and value are: news, Web pages, books, CDs, movies. Items with larger complexity and value are: digital cameras, mobile phones, PCs, etc. The most complex items that have been considered are insurance policies, financial investments, travels, jobs.

RSs, according to their core technology, can use a range of properties and features of the items. For example in a movie recommender system, the genre (such as comedy, thriller, etc.), as well as the director, and actors can be used to describe a movie and to learn how the utility of an item depends on its features. Items can be represented using various information and representation approaches, e.g., in a minimalist way as a single id code, or in a richer form, as a set of attributes, but even as a concept in an ontological representation of the domain.

Users.

Users of a RS, as mentioned above, may have very diverse goals and characteristics. In order to personalize the recommendations and the human-computer interaction, RSs exploit a range of information about the users. This information can be structured in various ways and again the selection of what information to model depends on the recommendation technique.

Users can also be described by their behavior pattern data, for example, site browsing patterns (in a Web-based recommender system), or travel search patterns (in a travel recommender system). Moreover, user data may include relations between users such as the trust level of these relations between users. A RS might utilize this information to recommend items to users that were preferred by similar or trusted users.

Transactions.

We generically refer to a transaction as a recorded interaction between a user and the RS. Transactions are log-like data that store important information generated during the human-computer interaction and which are useful for the recommendation generation algorithm that the system is using. For instance, a transaction log may contain a reference to the item selected by the user and a description of the context (e.g., the user goal/query) for that particular recommendation. If available, that transaction may also include an explicit feedback the user has provided, such as the rating for the selected item.

In fact, ratings are the most popular form of transaction data that a RS collects. These ratings may be collected explicitly or implicitly. In the explicit collection of ratings, the user is asked to provide her opinion about an item on a rating scale. Ratings can take on a variety of forms:

Numerical ratings such as the 1-5 stars provided in the book recommender associated with Amazon.com.

- **Ordinal ratings**, such as "strongly agree, agree, neutral, disagree, strongly disagree"

  where the user is asked to select the term that best indicates her opinion regarding an item (usually via questionnaire).
- **Binary ratings** that model choices in which the user is simply asked to decide if a certain item is good or bad.
- **Unary ratings** can indicate that a user has observed or purchased an item, or otherwise rated the item positively. In such cases, the absence of a rating indicates that we have no information relating the user to the item (perhaps she purchased the item somewhere else).

RECOMMENDATION TECHNIQUES

In order to implement its core function, identifying the useful items for the user, a RS must predict that an item is worth recommending. In order to do this, the system must be able to predict the utility of some of them, or at least compare the utility of some items, and then decide what items to recommend based on this comparison. The prediction step may not be explicit in the recommendation algorithm but we can still apply this unifying model to describe the general role of a RS. Here our goal is to provide the reader with a unifying perspective rather than an account of all the different recommendation approaches that will be illustrated in this handbook.

To provide a first overview of the different types of RSs, we want to quote a taxonomy provided by that has become a classical way of distinguishing between recommender systems and referring to them.

Distinguishes between six different classes of recommendation approaches:

Content-based: The system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. For example, if a user has positively

rated a movie that belongs to the comedy genre, then the system can learn to recommend other movies from this genre.

Collaborative filtering: The simplest and original implementation of this approach recommends to the active user the items that other users with similar tastes liked in the past. The similarity in taste of two users is calculated based on the similarity in the rating history of the users. This is the reason why refers to collaborative filtering as "people-to-people correlation." Collaborative filtering is considered to be the most popular and widely implemented technique in RS.

Demographic: This type of system recommends items based on the demographic profile of the user. The assumption is that different recommendations should be generated for different demographic niches. Many Web sites adopt simple and effective personalization solutions based on demographics. For example, users are dispatched to particular Web sites based on their language or country. Or suggestions may be customized according to the age of the user. While these approaches have been quite popular in the marketing literature, there has been relatively little proper RS research into demographic systems.

Knowledge-based: Knowledge-based systems recommend items based on specific domain knowledge about how certain item features meet users needs and preferences and, ultimately, how the item is useful for the user. Notable knowledge based recommender systems are case-based. In these systems a similarity function estimates how much the user needs (problem description) match the recommendations (solutions of the problem). Here the similarity score can be directly interpreted as the utility of the recommendation for the user.

Community-based: This type of system recommends items based on the preferences of the users friends. This technique follows the epigram "Tell me who your friends are, and I will tell you who you are". Evidence suggests that people tend to rely more on recommendations from their friends than on recommendations

from similar but anonymous individuals. This observation, combined with the growing popularity of open social networks, is generating a rising interest in community-based systems or, as or as they usually referred to, social recommender systems. This type of RSs models and acquires information about the social relations of the users and the preferences of the user's friends. The recommendation is based on ratings that were provided by the user's friends. In fact these RSs are following the rise of social-networks and enable a simple and comprehensive acquisition of data related to the social relations of the users.

Hybrid recommender systems: These RSs are based on the combination of the above mentioned techniques. A hybrid system combining techniques A and B tries to use the advantages of A to fix the disadvantages of B. For instance, CF methods suffer from new-item problems, i.e., they cannot recommend items that have no ratings. This does not limit content-based approaches since the prediction for new items is based on their description (features) that are typically easily available. Given two (or more) basic RSs techniques, several ways have been proposed for combining them to create a new hybrid.

The aspects that apply to the design stage include factors that might affect the choice of the algorithm. The first factor to consider, the application's domain, has a major effect on the algorithmic approach that should be taken. Based on the specific application domains, we define more general classes of domains for the most common recommender systems applications:

- **Entertainment** - recommendations for movies, music, and IPTV.
- **Content** - personalized newspapers, recommendation for documents, recommendations of Web pages, e-learning applications, and e-mail filters.
- **E-commerce** - recommendations for consumers of products to buy such as books, cameras, PCs etc.

- **Services** - recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services.

CONTENT-BASED RECOMMENDER SYSTEMS

Recommender systems have the effect of guiding users in a personalized way to interesting objects in a large space of possible options. Content-based recommendation systems try to recommend items similar to those a given user has liked in the past. Indeed, the basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of a content object (item), in order to recommend to the user new interesting items. This chapter provides an overview of content-based recommender systems, with the aim of imposing a degree of order on the diversity of the different aspects involved in their design and implementation.

The first part of the chapter presents the basic concepts and terminology of content based recommender systems, a high level architecture, and their main advantages and drawbacks. The second part of the chapter provides a review of the state of the art of systems adopted in several application domains, by thoroughly describing both classical and advanced techniques for representing items and user profiles. The most widely adopted techniques for learning user profiles are also presented. The last part of the chapter discusses trends and future research which might lead towards the next generation of systems, by describing the role of User Generated Content as a way for taking into account evolving vocabularies, and the challenge of feeding users with serendipitous recommendations, that is to say surprisingly interesting items that they might not have otherwise discovered.

Content-based recommendation systems try to recommend items similar to those a given user has liked in the past, whereas systems designed according to the

collaborative recommendation paradigm identify users whose preferences are similar to those of the given user and recommend items they have liked.

BASICS OF CONTENT-BASED RECOMMENDER SYSTEMS

Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the user's level of interest in that object. If a profile accurately reflects user preferences, it is of tremendous advantage for the effectiveness of an information access process. For instance, it could be used to filter search results by deciding whether a user is interested in a specific Web page or not and, in the negative case, preventing it from being displayed.

A HIGH LEVEL ARCHITECTURE OF CONTENT-BASED SYSTEMS

Content-based Information Filtering (IF) systems need proper techniques for representing the items and producing the user profile, and some strategies for comparing the user profile with the item representation. The high level architecture of a content based recommender system is depicted in Figure 3.1. The recommendation process is performed in three steps, each of which is handled by a separate component:

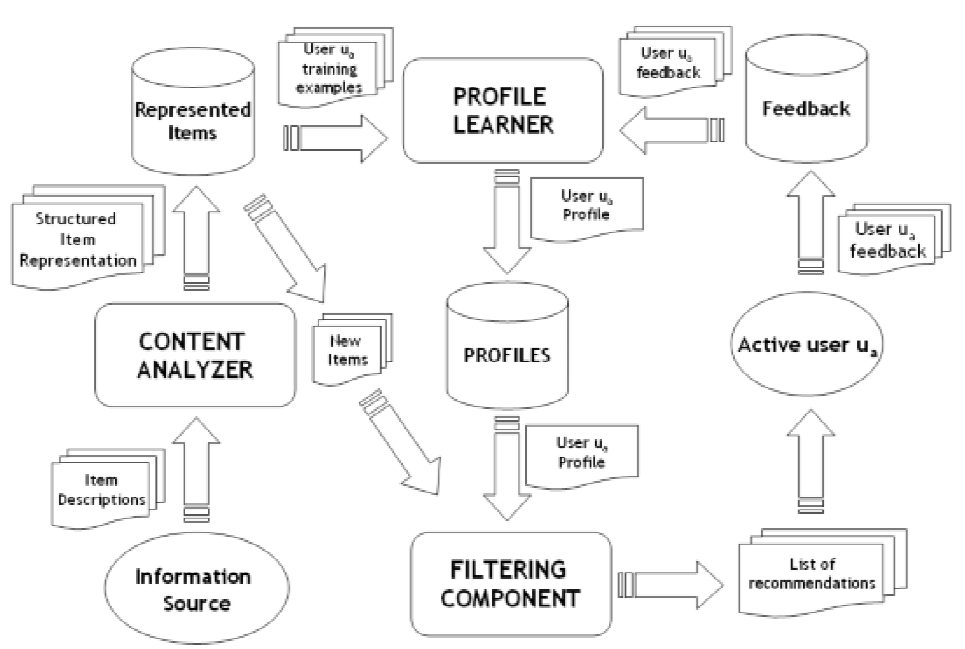


Fig. 3.1: High level architecture of a Content-based Recommender

• **CONTENT ANALYZER** – When information has no structure (e.g. text), some kind of pre-processing step is needed to extract structured relevant information. The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps. Data items are analyzed by feature extraction techniques in order to shift item representation from the original information space to the target one (e.g. Web pages represented as keyword vectors). This representation is the input to the PROFILE LEARNER and FILTERING COMPONENT;

• **PROFILE LEARNER** – This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques, which are able to infer a model of user interests starting from items liked or disliked in the past. For instance, the PROFILE LEARNER of a Web page

recommender can implement a relevance feedback method in which the learning technique combines vectors of positive and negative examples into a prototype vector representing the user profile. Training examples are Web pages on which a positive or negative feedback has been provided by the user;

• **FILTERING COMPONENT** – This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance judgment (computed using some similarity metrics [42]), the latter case resulting in a ranked list of potentially interesting items. In the above mentioned example, the matching is realized by computing the cosine similarity between the prototype vector and the item vectors.

The first step of the recommendation process is the one performed by the CONTENT ANALYZER, that usually borrows techniques from Information Retrieval system. Item descriptions coming from Information Source are processed by the CONTENT ANALYZER, that extracts features (keywords, n-grams, concepts, . . .) from unstructured text to produce a structured item representation, stored in the repository Represented Items. In order to construct and update the profile of the active user u (user for which recommendations must be provided) her reactions to items are collected in some way and recorded in the repository Feedback. These reactions, called annotations or feedback, together with the related item descriptions, are exploited during the process of learning a model useful to predict the actual relevance of newly presented items. Users can also explicitly define their areas of interest as an initial profile without providing any feedback. Typically, it is possible to distinguish between two kinds of relevance feedback:

- positive information (inferring features liked by the user) and
- negative information

Two different techniques can be adopted for recording user's feedback. When a system requires the user to explicitly evaluate items, this technique is usually referred to as "explicit feedback"; the other technique, called "implicit

feedback", a does not require any active user involvement, in the sense that feedback is derived from monitoring and analyzing user's activities. Explicit evaluations indicate how relevant or interesting an item is to the user. There are three main approaches to get explicit relevance feedback:

like/dislike – items are classified as "relevant" or "not relevant" by adopting a simple binary rating scale; ratings – a discrete numeric scale is usually adopted to judge items. Alternatively, symbolic ratings are mapped to a numeric scale, such as in Syskill & Webert, where users have the possibility of rating a Web page as hot, lukewarm, or cold; text comments – Comments about a single item are collected and presented to the users as a means of facilitating the decision-making process. For instance, customer's feedback at Amazon.com or eBay.com might help users in deciding whether an item has been appreciated by the community. Textual comments are helpful, but they can overload the active user because she must read and interpret each comment to decide if it is positive or negative, and to what degree.

The literature proposes advanced techniques from the affective computing research area to make content-based recommenders able to automatically perform this kind of analysis.

ADVANTAGES AND DRAWBACKS OF CONTENT-BASED FILTERING

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

• **USER INDEPENDENCE** - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the "nearest neighbors" of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended;

• **TRANSPARENCY** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;

• **NEW ITEM** - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users' preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Nonetheless, content-based systems have several shortcomings:

• **LIMITED CONTENT ANALYSIS** - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a user's experience.

For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for affective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information.

To sum up, both automatic and manually assignment of features to items could not be sufficient to define distinguishing aspects of items that turn out to be necessary for the elicitation of user interests.

OVER-SPECIALIZATION - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A "perfect" content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful.

NEW USER - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

COLLABORATIVE FILTERING

The collaborative filtering (CF) approach to recommenders has recently enjoyed much interest and progress. The fact that it played a central role within the recently completed Netflix competition has contributed to its popularity. This chapter surveys the recent progress in the field. Matrix factorization techniques, which became a first choice for implementing CF, are described together with recent innovations. We also describe several extensions that bring competitive accuracy into neighborhood methods, which used to dominate the field. The chapter demonstrates how to utilize temporal models and implicit feedback to extend models accuracy. In passing, we include detailed descriptions of some the central methods developed for tackling the challenge of the Netflix Prize competition. Collaborative filtering (CF) methods produce user specific recommendations of items based on patterns of ratings or usage (e.g., purchases) without need for exogenous information about either items or users. While well established methods work adequately for many purposes, we present several recent extensions available

to analysts who are looking for the best possible recommendations.

The Netflix Prize competition that began in October 2006 has fueled much recent progress in the field of collaborative filtering. For the first time, the research community gained access to a large-scale, industrial strength data set of 100 million movie ratings attracting thousands of scientists, students, engineers and enthusiasts to the field. The nature of the competition has encouraged rapid development, where innovators built on each generation of techniques to improve prediction accuracy.

Because all methods are judged by the same rigid yardstick on common data, the evolution of more powerful models has been especially efficient. Recommender systems rely on various types of input. Most convenient is high quality explicit feedback , where users directly report on their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons.

MATRIX FACTORIZATION MODELS

Latent factor models approach collaborative filtering with the holistic goal to uncover latent features that explain observed ratings; examples include pLSA, neural networks, Latent Dirichlet Allocation, and models that are induced by factorization of the user-item ratings matrix (also known as SVD-based models). Recently, matrix factorization models have gained popularity, thanks to their attractive accuracy and scalability. In information retrieval, SVD is well established for identifying latent semantic factors. However, applying SVD to explicit ratings in the CF domain raises difficulties due to the high portion of missing values.

Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works relied on imputation, which fills in missing ratings and makes the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent

works suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model.

In this section we describe several matrix factorization techniques, with increasing complexity and accuracy. We start with the basic model – "SVD". Then, we show how to integrate other sources of user feedback in order to increase prediction accuracy, through the "SVD++ model". Finally we deal with the fact that customer preferences for products may drift over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste. This leads to a factor model that addresses temporal dynamics for better tracking user behavior.

SVD

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f , such that user-item interactions are modeled as inner products in that space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or "quirkiness"; or completely uninterpretable dimensions.

SVD++

Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. This is especially helpful for those users that provided much more implicit feedback than explicit one. As explained earlier, even in cases where independent implicit feedback is absent, one can capture a significant signal by accounting for which items users rate, regardless of their rating value. This led to several methods that modeled a user factor by the identity of the items he/she has rated. Here we focus on the SVD++ method, which was shown to offer accuracy superior to SVD.

Time-aware factor model

The matrix-factorization approach lends itself well to modeling temporal effects, which can significantly improve its accuracy. Decomposing ratings into distinct terms allows us to treat different temporal aspects separately. Specifically, we identify the following effects that each vary over time:

(1) user biases b(t),

(2) item biases bi(t), and

(3) user preferences p(t).

On the other hand, we specify static item characteristics, q_i, because we do not expect significant temporal variation for items, which, unlike humans, are static in nature. We start with a detailed discussion of the temporal effects that are contained within the baseline predictors.

NEIGHBORHOOD MODELS

The most common approach to CF is based on neighborhood models. Its original form, which was shared by virtually all earlier CF systems, is user-user based; see for a good analysis. User-user methods estimate unknown ratings based on recorded ratings of likeminded users. Later, an analogous item-item approach became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-item approach more favorable in many cases. In addition, item-item methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like-minded users. We focus mostly on item-item approaches, but the same techniques can be directly applied within a user-user approach;

A global neighborhood model

In this subsection, we introduce a neighborhood model based on global optimization. The model offers an improved prediction accuracy, by offering the

aforementioned merits of the model. with additional advantages that are summarized as follows:

1. No reliance on arbitrary or heuristic item-item similarities. The new model is cast as the solution to a global optimization problem.
2. Inherent overfitting prevention or "risk control": the model reverts to robust baseline predictors, unless a user entered sufficiently many relevant ratings.
3. The model can capture the totality of weak signals encompassed in all of a user's ratings, not needing to concentrate only on the few ratings for most similar items.
4. The model naturally allows integrating different forms of user input, such as explicit and implicit feedback.
5. A highly scalable implementation allows linear time and space complexity, thus facilitating both item-item and user-user implementations to scale well to very large datasets.
6. Time drifting aspects of the data can be integrated into the model, thereby improving its accuracy.

Neighborhood-based Recommendation Methods

Abstract Among collaborative recommendation approaches, methods based on nearest-neighbors still enjoy a huge amount of popularity, due to their simplicity, their efficiency, and their ability to produce accurate and personalized recommendations. This chapter presents a comprehensive survey of neighborhood-based methods for the item recommendation problem. In particular, the main benefits of such methods, as well as their principal characteristics, are described. Furthermore, this document addresses the essential decisions that are required while implementing a neighborhood-based recommender system, and gives practical information on how to make such decisions. Finally, the problems of sparsity and limited coverage, often observed in large commercial recommender systems, are discussed, and a few solutions to overcome these problems are presented.

Building the Model

We gradually construct the various components of the model, through an ongoing refinement of our formulations. Previous models were centered around *user-specific* interpolation weights – θ_{ij}^u in (5.19) or $s_{ij}/\sum_{j\in S^k(i;u)} s_{ij}$ in (5.17) – relating item i to the items in a user-specific neighborhood $S^k(i;u)$. In order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global item-item weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learned from the data through optimization. An initial sketch of the model describes each rating r_{ui} by the equation

$$\hat{r}_{ui} = b_{ui} + \sum_{j\in R(u)} (r_{uj} - b_{uj}) w_{ij} \,. \tag{5.29}$$

This rule starts with the crude, yet robust, baseline predictors (b_{ui}). Then, the estimate is adjusted by summing over *all* ratings by u.

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, we adopt a different viewpoint, that enables a more flexible usage of the weights. We no longer treat weights as interpolation coefficients. Instead, we take weights as part of adjustments, or *offsets*, added to the baseline predictors. This way, the weight w_{ij} is the extent by which we increase our baseline prediction of r_{ui} based on the observed value of r_{uj}. For two related items i and j, we expect w_{ij} to be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$ is high), we would like to increase our estimate for u's rating of i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline prediction. Likewise, our estimate will not deviate much from the baseline by an item j that u rated just as expected ($r_{uj} - b_{uj}$ is around zero), or by an item j that is not known to be predictive on i (w_{ij} is close to zero).

5.5.2.1 Factoring item-item relationships

We factor item-item relationships by associating each item i with three vectors: $q_i, x_i, y_i \in \mathbb{R}^f$. This way, we confine w_{ij} to be $q_i^T x_i$. Similarly, we impose the structure $c_{ij} = q_i^T y_j$. Essentially, these vectors strive to map items into an f-dimensional latent factor space where they are measured against various aspects that are revealed automatically by learning from the data. By substituting this into (5.34) we get the following prediction rule:

$$\hat{r}_{ui} = \mu + b_u + b_i + |R(u)|^{-\frac{1}{2}} \sum_{j\in R(u)} [(r_{uj} - b_{uj}) q_i^T x_j + q_i^T y_j] \tag{5.37}$$

Computational gains become more obvious by using the equivalent rule

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|R(u)|^{-\frac{1}{2}} \sum_{j\in R(u)} (r_{uj} - b_{uj}) x_j + y_j \right) . \tag{5.38}$$

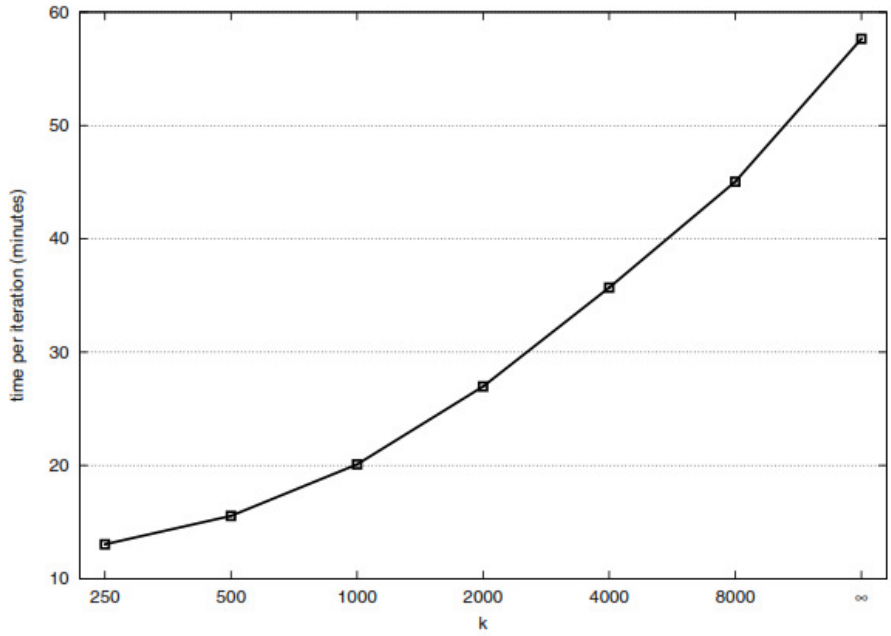


Fig. 5.2: Running time per iteration of the globally optimized neighborhood model, as a function of the parameter k.

Notice that the bulk of the rule $(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + y_j)$ depends only on u while being independent of i. This leads to an efficient way to learn the model parameters. As usual, we minimize the regularized squared error function associated with (5.38)

$$\min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + y_j \right) \right)^2 + \lambda_{11} \left(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathrm{R}(u)} \|x_j\|^2 + \|y_j\|^2 \right). \quad (5.39)$$

Optimization is done by a stochastic gradient descent scheme, which is described in the following pseudo code: