CHAPTER 31

Information Retrieval

Textual data are unstructured, unlike the rigidly structured data in relational databases. The term **information retrieval** generally refers to the querying of unstructured textual data. Information-retrieval systems have much in common with database systems, in particular, the storage and retrieval of data on secondary storage. However, the emphasis in the field of information systems is different from that in database systems, concentrating on issues such as querying based on keywords; the relevance of documents to the query; and the analysis, classification, and indexing of documents. web search engines today go beyond the paradigm of retrieving documents and address broader issues such as what information to display in response to a keyword query, to satisfy the information needs of a user.

31.1 Overview

The field of **information retrieval** has developed in parallel with the field of databases. In the traditional model used in the field of information retrieval, information is organized into documents, and it is assumed that there is a large number of documents. Data contained in documents are unstructured, without any associated schema. The process of information retrieval consists of locating relevant documents on the basis of user input, such as keywords or example documents.

The web provides a convenient way to get to, and to interact with, information sources across the internet. However, a persistent problem facing the web is the explosion of stored information, with little guidance to help the user to locate what is interesting. Information retrieval has played a critical role in making the web a productive and useful tool, especially for researchers.

Traditional examples of information-retrieval systems are online library catalogs and online document-management systems such as those that store newspaper articles. The data in such systems are organized as a collection of *documents*; a newspaper article and a catalog entry (in a library catalog) are examples of documents. In the context of the web, usually each HTML page is considered to be a document.

A user of such a system may want to retrieve a particular document or a particular class of documents. The intended documents are typically described by a set of **keywords**—for example, the keywords "database system" may be used to locate books on database systems, and the keywords "stock" and "scandal" may be used to locate articles about stock-market scandals. Documents have associated with them a set of keywords, and documents whose keywords contain those supplied by the user are retrieved.

Keyword-based information retrieval can be used not only for retrieving textual data, but also for retrieving other types of data, such as video and audio data, that have descriptive keywords associated with them. For instance, a video movie may have associated with it keywords such as its title, director, actors, and genre, while an image or video clip may have tags, which are keywords describing the image or video clip, associated with it.

There are several differences between this model and the models used in traditional database systems.

- Database systems deal with several operations that are not addressed in information-retrieval systems. For instance, database systems deal with updates and with the associated transactional requirements of concurrency control and durability. These matters are viewed as less important in information systems. Similarly, database systems deal with structured information organized with relatively complex data models (such as the relational model or object-oriented data models), whereas information-retrieval systems traditionally have used a much simpler model, where the information in the database is organized simply as a collection of unstructured documents.
- Information-retrieval systems deal with several issues that have not been addressed adequately in database systems. For instance, the field of information retrieval has dealt with the issue of querying collections of unstructured documents, focusing on issues such as keyword queries, and of ranking of documents on estimated degree of relevance of the documents to the query.

In addition to simple keyword queries that are just sets of words, information-retrieval systems typically allow query expressions formed using keywords and the logical connectives *and, or*, and *not*. For example, a user could ask for all documents that contain the keywords "motorcycle *and* maintenance," or documents that contain the keywords "computer *or* microprocessor," or even documents that contain the keyword "computer *but not* database." A query containing keywords without any of the above connectives is assumed to have *and*s implicitly connecting the keywords.

In **full text** retrieval, all the words in each document are considered to be keywords. For unstructured documents, full text retrieval is essential since there may be no information about what words in the document are keywords. We shall use the word **term** to refer to the words in a document, since all words are keywords.

In its simplest form, an information-retrieval system locates and returns all documents that contain all the keywords in the query, if the query has no connectives; connectives are handled as you would expect. More sophisticated systems estimate relevance of documents to a query so that the documents can be shown in order of estimated relevance. They use information about term occurrences, as well as hyperlink information, to estimate relevance.

Information-retrieval systems, as exemplified by web search engines, have today evolved beyond just retrieving documents based on a ranking scheme. Today, search engines aim to satisfy a user's information needs by judging what topic a query is about and displaying not only web pages judged as relevant, but also displaying other kinds of information about the topic. For example, given a query term *cricket*, a search engine may display scores from ongoing or recent cricket matches, rather than just display top-ranked documents related to cricket. As another example, in response to a query "New York", a search engine may show a map of New York, and images of New York, in addition to web pages related to New York.

31.2 Relevance Ranking Using Terms

The set of all documents that satisfy a query expression may be very large; in particular, there are billions of documents on the web, and most keyword queries on a web search engine find hundreds of thousands of documents containing the keywords. Full text retrieval makes this problem worse: each document may contain many terms, and even terms that are mentioned only in passing are treated equivalently with documents where the term is indeed relevant. Irrelevant documents may be retrieved as a result.

Information-retrieval systems therefore estimate the relevance of documents to a query and return only highly ranked documents as answers. **Relevance ranking** is not an exact science, but there are some well-accepted approaches.

31.2.1 Ranking Using TF-IDF

The first question to address is, given a particular term t, how relevant is a particular document d to the term. One approach is to use the the number of occurrences of the term in the document as a measure of its relevance, on the assumption that relevant terms are likely to be mentioned many times in a document. Just counting the number of occurrences of a term is usually not a good indicator: first, the number of occurrences depends on the length of the document, and second, a document containing 10 occurrences of a term may not be 10 times as relevant as a document containing one occurrence.

One way of measuring $TF(d, t)$, the relevance of a document d to a term t, is

$$TF(d, t) = \log\left(1 + \frac{n(d, t)}{n(d)}\right)$$

where $n(d)$ denotes the number of term occurrences in the document and $n(d, t)$ denotes the number of occurrences of term t in the document d. Observe that this metric takes the length of the document into account. The relevance grows with more occurrences of a term in the document, although it is not directly proportional to the number of occurrences.

Many systems refine the preceding metric by using other information. For instance, if the term occurs in the title, or the author list, or the abstract, the document would be considered more relevant to the term. Similarly, if the first occurrence of a term is late in the document, the document may be considered less relevant than if the first occurrence is early in the document. These notions can be formalized by extensions of the formula we have shown for $TF(d, t)$. In the information retrieval community, the relevance of a document to a term is referred to as **term frequency** (**TF**), regardless of the exact formula used.

A query Q may contain multiple keywords. The relevance of a document to a query with two or more keywords is estimated by combining the relevance measures of the document to each keyword. A simple way of combining the measures is to add them up. However, not all terms used as keywords are equal. Suppose a query uses two terms, one of which occurs frequently, such as *database*, and another that is less frequent, such as *Silberschatz*. A document containing *Silberschatz* but not *database* should be ranked higher than a document containing the term *database* but not *Silberschatz*.

To fix the problem, weights are assigned to terms using the **inverse document frequency** (**IDF**), defined as

$$IDF(t) = \frac{1}{n(t)}$$

where $n(t)$ denotes the number of documents (among those indexed by the system) that contain the term t. The **relevance** of a document d to a set of terms Q is then defined as

$$r(d, Q) = \sum_{t \in Q} TF(d, t) * IDF(t)$$

This measure can be further refined if the user is permitted to specify weights $w(t)$ for terms in the query, in which case the user-specified weights are also taken into account by multiplying $TF(t)$ by $w(t)$ in the above formula.

The above approach of using term frequency and inverse document frequency as a measure of the relevance of a document is called the **TF–IDF** approach.

Almost all text documents (in English) contain words such as *and*, *or*, *a*, and so on, and hence these words are useless for querying purposes since their inverse document frequency is extremely low. Information-retrieval systems define a set of words, called **stop words**, containing 100 or so of the most common words, and ignore these words when indexing a document. Such words are not used as keywords and are discarded if present in the keywords supplied by the user.

Another factor taken into account when a query contains multiple terms is the **proximity** of the terms in the document. If the terms occur close to each other in the document, the document will be ranked higher than if they occur far apart. The formula for $r(d, Q)$ can be modified to take proximity of the terms into account.

Given a query Q, the job of an information-retrieval system is to return documents in descending order of their relevance to Q. Since there may be a very large number of documents that are relevant, information-retrieval systems typically return only the first few documents with the highest degree of estimated relevance, and they permit users to interactively request further documents.

31.2.2 Similarity-Based Retrieval

Certain information-retrieval systems permit **similarity-based retrieval**. Here, the user can give the system document A, and ask the system to retrieve documents that are "similar" to A. The similarity of a document to another may be defined, for example, on the basis of common terms. One approach is to find k terms in A with highest values of $TF(A, t) * IDF(t)$, and to use these k terms as a query to find the relevance of other documents. The terms in the query are themselves weighted by $TF(A, t) * IDF(t)$.

More generally, the similarity of documents is defined by the **cosine similarity** metric. Let the terms occurring in either of the two documents be $t_1, t_2, \ldots, t_n$. Let $r(d, t) = TF(d, t) * IDF(t)$. Then the cosine similarity metric between documents d and e is defined as:

$$\frac{\sum_{i=1}^{n} r(d, t_i) r(e, t_i)}{\sqrt{\sum_{i=1}^{n} r(d, t_i)^2} \sqrt{\sum_{i=1}^{n} r(e, t_i)^2}}$$

You can easily verify that the cosine similarity metric of a document with itself is 1, while that between two documents that do not share any terms is 0.

The name "cosine similarity" comes from the fact that the above formula computes the cosine of the angle between two vectors, one representing each document, defined as follows: Let there be n words overall across all the documents being considered. An n-dimensional space is defined, with each word as one of the dimensions. A document d is represented by a point in this space, with the value of the ith coordinate of the point being $r(d, t_i)$. The vector for document d connects the origin (all coordinates = 0) to the point representing the document. The model of documents as points and vectors in an n-dimensional space is called the **vector space model**.

If the set of documents similar to a query document A is large, the system may present the user a few of the similar documents, allow the user to choose the most relevant few, and start a new search based on similarity to A *and* to the chosen documents. The resultant set of documents is likely to be what the user intended to find. This idea is called **relevance feedback**.

Relevance feedback can also be used to help users find relevant documents from a large set of documents matching the given query keywords. In such a situation, users

may be allowed to identify one or a few of the returned documents as relevant; the system then uses the identified documents to find other similar ones. The resultant set of documents is likely to be what the user intended to find. An alternative to the relevance feedback approach is to require users to modify the query by adding more keywords; relevance feedback can be easier to use, in addition to giving a better final set of documents as the answer.

In order to show the user a representative set of documents when the number of documents is very large, a search system may cluster the documents based on their cosine similarity. Then a few documents from each cluster may be shown, so that more than one cluster is represented in the set of answers.

As a special case of similarity, there are often multiple copies of a document on the web; this could happen, for example, if a web site mirrors the contents of another web site. In this case, it makes no sense to return multiple copies of a highly ranked document as separate answers; duplicates should be detected, and only one copy should be returned as an answer.

31.3 Relevance Using Hyperlinks

Early web-search engines ranked documents by using only TF-IDF–based relevance measures like those described in Section 31.2. However, these techniques had some limitations when used on very large collections of documents, such as the set of all web pages. In particular, many web pages have all the keywords specified in a typical search engine query; further, some of the pages that users want as answers often have just a few occurrences of the query terms and would not get a very high TF-IDF score.

However, researchers soon realized that web pages have very important information that plain text documents do not have, namely hyperlinks. These can be exploited to get better relevance ranking; in particular, the relevance ranking of a page is influenced greatly by hyperlinks that point *to* the page. In this section, we study how hyperlinks are used for ranking of web pages.

31.3.1 Popularity Ranking

The basic idea of **popularity ranking** (also called **prestige ranking**) is to find pages that are popular and to rank them higher than other pages that contain the specified keywords. Since most searches are intended to find information from popular pages, ranking such pages higher is generally a good idea. For instance, the term *google* may occur in vast numbers of pages, but the page google.com is the most popular among the pages that contain the term *google*. The page google.com should therefore be ranked as the most relevant answer to a query consisting of the term *google*.

Traditional measures of relevance of a page such as the TF-IDF-based measures that we saw in Section 31.2 can be combined with the popularity of the page to get an overall measure of the relevance of the page to the query. Pages with the highest overall relevance value are returned as the top answers to a query.

This raises the question of how to define and how to find the popularity of a page. One way would be to find how many times a page is accessed and use the number as a measure of the site's popularity. However, getting such information is impossible without the cooperation of the site, and while a few sites may be persuaded to reveal this information, it is difficult to get it for all sites. Further, sites may lie about their access frequency in order to get ranked higher.

A very effective alternative is to use hyperlinks to a page as a measure of its popularity. Many people have bookmark files that contain links to sites that they use frequently. Sites that appear in a large number of bookmark files can be inferred to be very popular sites. Bookmark files are usually stored privately and are not accessible on the web. However, many users do maintain web pages with links to their favorite web pages. Many web sites also have links to other related sites, which can also be used to infer the popularity of the linked sites. A web search engine can fetch web pages (by a process called crawling, which we describe in Section 31.7) and analyze them to find links between the pages.

A first solution to estimating the popularity of a page is to use the number of pages that link to the page as a measure of its popularity. However, this by itself has the drawback that many sites have a number of useful pages, yet external links often point only to the root page of the site. The root page in turn has links to other pages in the site. These other pages would then be wrongly inferred to be not very popular and would have a low ranking in answering queries.

One alternative is to associate popularity with sites, rather than with pages. All pages at a site then get the popularity of the site, and pages other than the root page of a popular site would also benefit from the site's popularity. However, the question of what constitutes a site then arises. In general, the internet address prefix of a page URL would constitute the site corresponding to the page. However, there are many sites that host a large number of mostly unrelated pages, such as home page servers in universities and web portals such as groups.yahoo.com or groups.google.com. For such sites, the popularity of one part of the site does not imply popularity of another part of the site.

A simpler alternative is to allow **transfer of prestige** from popular pages to pages to which they link. Under this scheme, in contrast to the one-person one-vote principles of democracy, a link from a popular page x to a page y is treated as conferring more prestige to page y than a link from a not-so-popular page z.[1]

This notion of popularity is in fact circular, since the popularity of a page is defined by the popularity of other pages, and there may be cycles of links between pages. However, the popularity of pages can be defined by a system of simultaneous linear equations, which can be solved by matrix manipulation techniques. The linear equations can be defined in such a way that they have a unique and well-defined solution.

[1]This is similar in some sense to giving extra weight to endorsements of products by celebrities (such as film stars), so its significance is open to question, although it is effective and widely used in practice.

It is interesting to note that the basic idea underlying popularity ranking is actually quite old and first appeared in a theory of social networking developed by sociologists in the 1950s. In the social-networking context, the goal was to define the prestige of people. For example, the president of the United States has high prestige since a large number of people know him. If someone is known by multiple prestigious people, then she also has high prestige, even if she is not known by as large a number of people. The use of a set of linear equations to define the popularity measure also dates back to this work.

31.3.2 PageRank

The web search engine Google introduced **PageRank**, which is a measure of popularity of a page based on the popularity of pages that link to the page. Using the PageRank popularity measure to rank answers to a query gave results so much better than previously used ranking techniques that Google became the most widely used search engine in a rather short period of time.

PageRank can be understood intuitively using a **random walk model**. Suppose a person browsing the web performs a random walk (traversal) on web pages as follows: the first step starts at a random web page, and in each step, the random walker does one of the following: With a probability δ the walker jumps to a randomly chosen web page, and with a probability of $1 - \delta$ the walker randomly chooses one of the outlinks from the current web page and follows the link. The PageRank of a page is then the probability that the random walker is visiting the page at any given point in time.

Note that pages that are pointed to from many web pages are more likely to be visited and thus will have a higher PageRank. Similarly, pages pointed to by web pages with a high PageRank will also have a higher probability of being visited, and thus will have a higher PageRank.

PageRank can be defined by a set of linear equations, as follows: First, web pages are given integer identifiers. The jump probability matrix T is defined with $T[i,j]$ set to the probability that a random walker who is following a link out of page i follows the link to page j. Assuming that each link from i has an equal probability of being followed, $T[i,j] = 1/N_i$, where N_i is the number of links out of page i. Most entries of T are 0, and it is best represented as an adjacency list. Then the PageRank $P[j]$ for each page j can be defined as

$$P[j] = \delta/N + (1 - \delta) * \sum_{i=1}^{N} (T[i,j] * P[i])$$

where δ is a constant between 0 and 1 and N the number of pages; δ represents the probability of a step in the random walk being a jump.

The set of equations generated as above are usually solved by an an iterative technique, starting with each $P[i]$ set to $1/N$. Each step of the iteration computes new values for each $P[i]$ using the P values from the previous iteration. Iteration stops when the maximum change in any $P[i]$ value in an iteration goes below some cutoff value.

31.3.3 Other Measures of Popularity

Basic measures of popularity such as PageRank play an important role in ranking of query answers, but they are by no means the only factor. The TF-IDF scores of a page are used to judge its relevance to the query keywords, and they must be combined with the popularity ranking. Other factors must also be taken into account, to handle limitations of PageRank and related popularity measures.

Information about how often a site is visited would be a useful measure of popularity, but as mentioned earlier it is hard to obtain in general. However, search engines do track what fraction of times users click on a page when it is returned as an answer. This fraction can be used as a measure of the site's popularity. To measure the click fraction, instead of providing a direct link to the page, the search engine provides an indirect link through the search engine's site, which records the page click and transparently redirects the browser to the original link.[2]

One drawback of the PageRank algorithm is that it assigns a measure of popularity that does not take query keywords into account. For example, the page google.com is likely to have a very high PageRank because many sites contain a link to it. Suppose it contains a word mentioned in passing, such as *Stanford* (the advanced search page at Google did in fact contain this word at one point several years ago). A search on the keyword Stanford would then return google.com as the highest-ranked answer, ahead of a more relevant answer such as the Stanford University web page.

One widely used solution to this problem is to use keywords in the anchor text of links to a page to judge what topics the page is highly relevant to. The anchor text of a link consists of the text that appears within the HTML a href tag. For example, the anchor text of the link:

```
<a href="http://stanford.edu"> Stanford University</a>
```

is *Stanford University*. If many links to the page stanford.edu have the word *Stanford* in their anchor text, the page can be judged to be very relevant to the in keyword "Stanford." Text near the anchor text may also be taken into account; for example, a web site may contain the text "Stanford's home page is here" but may have used only the word *here* as anchor text in the link to the Stanford web site.

Popularity based on anchor text is combined with other measures of popularity, and with TF-IDF measures, to get an overall ranking for query answers, as we discuss in Section 31.3.5. As an implementation trick, the words in the anchor text are often treated as part of the page, with a term frequency based on the the popularity of the

[2]Sometimes this indirection is hidden from the user. For example, when you point the mouse at a link (such as http://db-book.com) in a Google query result, the link appears to point directly to the site. However, at least as of mid-2009, when you actually click on the link, Javascript code associated with the page actually rewrites the link to go indirectly through Google's site. If you use the back button of the browser to go back to the query result page, and point to the link again, the change in the linked URL becomes visible.

pages where the anchor text appears. Then TF-IDF ranking automatically takes anchor text into account.

An alternative approach to taking keywords into account when defining popularity is to compute a measure of popularity using *only* pages that contain the query keywords, instead of computing popularity using all available web pages. This approach is more expensive, since the computation of popularity ranking has to be done dynamically when a query is received, whereas PageRank is computed statically once and reused for all queries. web search engines handling billions of queries per day cannot afford to spend so much time answering a query. As a result, although this approach can give better answers, it is not very widely used.

The HITS algorithm was based on the above idea of first finding pages that contain the query keywords, and then computing a popularity measure using just this set of related pages. In addition, it introduced a notion of *hubs* and *authorities*. A **hub** is a page that stores links to many related pages; it may not in itself contain actual information on a topic, but it points to pages that contain actual information. In contrast, an **authority** is a page that contains actual information on a topic, although it may not store links to many related pages. Each page then gets a prestige value as a hub (*hub-prestige*) and another prestige value as an authority (*authority-prestige*). The definitions of prestige, as before, are cyclic and are defined by a set of simultaneous linear equations. A page gets higher hub-prestige if it points to many pages with high authority-prestige, while a page gets higher authority-prestige if it is pointed to by many pages with high hub-prestige. Given a query, pages with highest authority-prestige are ranked higher than other pages. See the bibliographical notes for references giving further details.

31.3.4 Search Engine Spamming

Search engine spamming refers to the practice of creating web pages, or sets of web pages, designed to get a high relevance rank for some queries, even though the sites are not actually popular sites. For example, a travel site may want to be ranked high for queries with the keyword "travel". It can get high TF-IDF scores by repeating the word *travel* many times in its page.[3] Even a site unrelated to travel, such as a pornographic site, could do the same thing, and would get highly ranked for a query on the word *travel*. In fact, this sort of spamming of TF-IDF was common in the early days of web search, and there was a constant battle between such sites and search engines that tried to detect spamming and deny them a high ranking.

Popularity ranking schemes such as PageRank make the job of search engine spamming more difficult, since just repeating words to get a high TF-IDF score is no longer sufficient. However, even these techniques can be spammed by creating a collection of web pages that point to each other, increasing their popularity rank. Techniques such as using sites instead of pages as the unit of ranking (with appropriately normalized jump

[3]Repeated words in a web page may confuse users; spammers can tackle this problem by delivering different pages to search engines than to other users, for the same URL, or by making the repeated words invisible, for example, by formatting the words in small white font on a white background.

probabilities) have been proposed to avoid some spamming techniques, but they are not fully effective against other spamming techniques. The war between search engine spammers and the search engines continues even today.

The hubs and authorities approach of the HITS algorithm is more susceptible to spamming. A spammer can create a web page containing links to good authorities on a topic, and gains a high hub score as a result. In addition, the spammer's web page includes links to pages that they wish to popularize, which may not have any relevance to the topic. Because these linked pages are pointed to by a page with a high hub score, they get a high but undeserved authority score.

31.3.5 Combining TF-IDF and Popularity Ranking Measures

We have seen two broad kinds of features used in ranking, TF-IDF and popularity scores such as PageRank. TF-IDF itself reflects a combination of several factors including raw term frequency and inverse document frequency, occurrence of a term in anchor text linking to the page, and a variety of other factors such as occurrence of the term in the title, occurrence of the term early in the document, and larger font size for the term, among other factors.

How to combine the scores of a page on each these factors, to generate an overall page score is a major problem that must be addressed by any information retrieval system. In the early days of search engines, humans created functions to combine scores into an overall score. But today, search engines use machine-learning techniques to decide how to combine scores. Typically, a score-combining formula is fixed, but the formula takes as parameters weights for different scoring factors. By using a training set of query results ranked by humans, a machine-learning algorithm can come up with an assignment of weights for each scoring factor that results in the best ranking performance across multiple queries.

We note that most search engines do not reveal how they compute relevance rankings; they believe that revealing their ranking techniques would allow competitors to catch up and would make the job of search engine spamming easier, resulting in poorer quality results.

31.4 Synonyms, Homonyms, and Ontologies

Consider the problem of locating documents about motorcycle maintenance, using the query "motorcycle maintenance". Suppose that the keywords for each document are the words in the title and the names of the authors. The document titled *Motorcycle Repair* would not be retrieved, since the word *maintenance* does not occur in its title.

We can solve that problem by making use of **synonyms**. Each word can have a set of synonyms defined, and the occurrence of a word can be replaced by the *or* of all its synonyms (including the word itself). Thus, the query "motorcycle *and* repair" can be replaced by "motorcycle *and* (repair *or* maintenance)." This query would find the desired document.

Keyword-based queries also suffer from the opposite problem, of **homonyms**, that is, single words with multiple meanings. For instance, the word *object* has different meanings as a noun and as a verb. The word *table* may refer to a dinner table or to a table in a relational database.

In fact, a danger even with using synonyms to extend queries is that the synonyms may themselves have different meanings. For example, "allowance" is a synonym for one meaning of the word *maintenance* but has a different meaning than what the user intended in the query "motorcycle maintenance". Documents that use the synonyms with an alternative intended meaning would be retrieved. The user is then left wondering why the system thought that a particular retrieved document (e.g., using the word *allowance*) is relevant, if it contains neither the keywords the user specified, nor words whose intended meaning in the document is synonymous with specified keywords! It is therefore a bad idea to use synonyms to extend a query without first verifying the synonyms with the user.

A better approach to this problem is for the system to understand what **concept** each word in a document represents, and similarly to understand what concepts a user is looking for, and to return documents that address the concepts that the user is interested in. A system that supports **concept-based querying** has to analyze each document to disambiguate each word in the document and replace it with the concept that it represents; disambiguation is usually done by looking at other surrounding words in the document. For example, if a document contains words such as *database* or *query*, the word *table* probably should be replaced by the concept "table: data," whereas if the document contains words such as *furniture*, *chair*, or *wood* near the word *table*, the word *table* should be replaced by the concept "table: furniture." Disambiguation based on nearby words is usually harder for user queries, since queries contain very few words, so concept-based query systems would offer several alternative concepts to the user, who picks one or more before the search continues.

Concept-based querying has several advantages; for example, a query in one language can retrieve documents in other languages, so long as they relate to the same concept. Automated translation mechanisms can be used subsequently if the user does not understand the language in which the document is written. However, the overhead of processing documents to disambiguate words is very high when billions of documents are being handled. Internet search engines therefore generally did not support concept-based querying initially, but interest in concept-based approaches is growing rapidly. However, concept-based querying systems have been built and used for other large collections of documents.

Querying based on concepts can be extended further by exploiting concept hierarchies. For example, suppose a person issues a query "flying animals"; a document containing information about "flying mammals" is certainly relevant, since a mammal is an animal. However, the two concepts are not the same, and just matching concepts would not allow the document to be returned as an answer. Concept-based querying systems can support retrieval of documents based on concept hierarchies.

Ontologies are hierarchical structures that reflect relationships between concepts. The most common relationship is the **is-a** relationship; for example, a leopard *is-a* mammal, and a mammal *is-a* animal. Other relationships, such as *part-of*, are also possible; for example, an airplane wing is *part-of* an airplane.

The WordNet system defines a large variety of concepts with associated words (called a *synset* in WordNet terminology). The words associated with a synset are synonyms for the concept; a word may be a synonym for several different concepts. In addition to synonyms, WordNet defines homonyms and other relationships. In particular, the *is-a* and *part-of* relationships that it defines connect concepts, and in effect define an ontology. The Cyc project is another effort to create an ontology.

In addition to language-wide ontologies, ontologies have been defined for specific areas to deal with terminology relevant to those areas. For example, ontologies have been created to standardize terms used in businesses; this is an important step in building a standard infrastructure for handling order processing and other interorganization flow of data. As another example, consider a medical insurance company that needs to get reports from hospitals containing diagnosis and treatment information. An ontology that standardizes the terms helps hospital staff to understand the reports unambiguously. This can greatly help in analysis of the reports—for example, to track how many cases of a particular disease occurred in a particular time frame.

It is also possible to build ontologies that link multiple languages. For example, WordNets have been built for different languages, and common concepts between languages can be linked to each other. Such a system can be used for translation of text. In the context of information retrieval, a multilingual ontology can be used to implement a concept-based search across documents in multiple languages.

The largest effort in using ontologies for concept-based queries is the **Semantic web**. The Semantic web is led by the World Wide web Consortium and consists of a collection of tools, standards, and languages that permit data on the web to be connected based on their semantics, or meaning. Instead of being a centralized repository, the Semantic web is designed to permit the same kind of decentralized, distributed growth that has made the World Wide web so successful. Key to this is the ability to integrate multiple, distributed ontologies. As a result, anyone with access to the internet can add to the Semantic web.

31.5 Indexing of Documents

An effective index structure is important for efficient processing of queries in an information-retrieval system. Documents that contain a specified keyword can be located efficiently by using an **inverted index** that maps each keyword K_i to a list S_i of (identifiers of) the documents that contain K_i. For example, if documents d_1, d_9, and d_{21} contain the term *Silberschatz*, the inverted list for the keyword "Silberschatz" would be "$d_1; d_9; d_{21}$". To support relevance ranking based on proximity of keywords, such an index may provide not just identifiers of documents, but also a list of locations within

the document where the keyword appears. For example, if "Silberschatz" appeared at position 21 in d_1, positions 1 and 19 in d_2, and positions 4, 29, and 46 in d_3, the inverted list with positions would be "$d_1/21; d_9/1, 19; d_{21}/4, 29, 46$". The inverted lists may also include with each document the term frequency of the term.

Such indices must be stored on disk, and each list S_i can span multiple disk pages. To minimize the number of I/O operations to retrieve each list S_i, the system would attempt to keep each list S_i in a set of consecutive disk pages, so the entire list can be retrieved with just one disk seek. A B^+-tree index can be used to map each keyword K_i to its associated inverted list S_i.

The *and* operation finds documents that contain all of a specified set of keywords $K_1, K_2, \ldots, K_n$. We implement the *and* operation by first retrieving the sets of document identifiers $S_1, S_2, \ldots, S_n$ of all documents that contain the respective keywords. The intersection, $S_1 \cap S_2 \cap \cdots \cap S_n$, of the sets gives the document identifiers of the desired set of documents. The *or* operation gives the set of all documents that contain at least one of the keywords $K_1, K_2, \ldots, K_n$. We implement the *or* operation by computing the union, $S_1 \cup S_2 \cup \cdots \cup S_n$, of the sets. The *not* operation finds documents that do not contain a specified keyword K_i. Given a set of document identifiers S, we can eliminate documents that contain the specified keyword K_i by taking the difference $S - S_i$, where S_i is the set of identifiers of documents that contain the keyword K_i.

Given a set of keywords in a query, many information-retrieval systems do not insist that the retrieved documents contain all the keywords (unless an *and* operation is used explicitly). In this case, all documents containing at least one of the words are retrieved (as in the *or* operation) but are ranked by their relevance measure.

To use term frequency for ranking, the index structure should additionally maintain the number of times terms occur in each document. To reduce this effort, they may use a compressed representation with only a few bits that approximates the term frequency. The index should also store the document frequency of each term (i.e., the number of documents in which the term appears).

If the popularity ranking is independent of the index term (as is the case for Page Rank), the list S_i can be sorted on the popularity ranking (and secondarily, for documents with the same popularity ranking, on document-id). Then a simple merge can be used to compute *and* and *or* operations. For the case of the *and* operation, if we ignore the TF-IDF contribution to the relevance score and merely require that the document should contain the given keywords, merging can stop once K answers have been obtained, if the user requires only the top K answers. In general, the results with the highest final score (after including TF-IDF scores) are likely to have high popularity scores and would appear near the front of the lists. Techniques have been developed to estimate the best possible scores of remaining results, and these can be used to recognize that answers not yet seen cannot be part of the top K answers. Processing of the lists can then terminate early.

However, sorting on popularity score is not fully effective in avoiding long inverted list scans, since it ignores the contribution of the TF-IDF scores. An alternative in such cases is to break up the inverted list for each term into two parts. The first part contains

documents that have a high TF-IDF score for that term (e.g., documents where the term occurs in the document title, or in anchor text referencing the document). The second part contains all documents. Each part of the list can be sorted in order of (popularity, document-id). Given a query, merging the first parts of the list for each term is likely to give several answers with an overall high score. If sufficient high-scoring answers are not found using the first parts of the lists, the second parts of the lists are used to find all remaining answers. If a document scores high on TF-IDF, it is likely to be found when merging the first parts of the lists. See the bibliographical notes for related references.

31.6 Measuring Retrieval Effectiveness

Each keyword may be contained in a large number of documents; hence, a compact representation is critical to keep space usage of the index low. Thus, the sets of documents for a keyword are maintained in a compressed form. So that storage space is saved, the index is sometimes stored such that the retrieval is approximate; a few relevant documents may not be retrieved (called a **false drop** or **false negative**), or a few irrelevant documents may be retrieved (called a **false positive**). A good index structure will not have *any* false drops, but it may permit a few false positives; the system can filter them away later by looking at the keywords that they actually contain. In web indexing, false positives are not desirable either, since the actual document may not be quickly accessible for filtering.

Two metrics are used to measure how well an information-retrieval system is able to answer queries. The first, **precision**, measures what percentage of the retrieved documents are actually relevant to the query. The second, **recall**, measures what percentage of the documents relevant to the query were retrieved. Ideally both should be 100 percent.

Precision and recall are also important measures for understanding how well a particular document-ranking strategy performs. Ranking strategies can result in false negatives and false positives, but in a more subtle sense.

- False negatives may occur when documents are ranked, as a result of relevant documents receiving a low ranking. If the system fetched all documents down to those with very low ranking there would be very few false negatives. However, humans would rarely look beyond the first few tens of returned documents, and they may thus miss relevant documents because they are not ranked highly. Exactly what is a false negative depends on how many documents are examined. Therefore, instead of having a single number as the measure of recall, we can measure the recall as a function of the number of documents fetched.
- False positives may occur because irrelevant documents get higher rankings than relevant documents. This too depends on how many documents are examined. One option is to measure precision as a function of number of documents fetched.

A better and more intuitive alternative for measuring precision is to measure it as a function of recall. With this combined measure, both precision and recall can be computed as a function of number of documents, if required.

For instance, we can say that with a recall of 50 percent the precision was 75 percent, whereas at a recall of 75 percent the precision dropped to 60 percent. In general, we can draw a graph relating precision to recall. These measures can be computed for individual queries, then averaged out across a suite of queries in a query benchmark.

Yet another problem with measuring precision and recall lies in how to define which documents are really relevant and which are not. In fact, it requires an understanding of natural language, and understanding of the intent of the query, to decide if a document is relevant or not. Researchers therefore have created collections of documents and queries and have manually tagged documents as relevant or irrelevant to the queries. Different ranking systems can be run on these collections to measure their average precision and recall across multiple queries.

31.7 Crawling and Indexing the web

web crawlers are programs that locate and gather information on the web. They recursively follow hyperlinks present in known documents to find other documents. Crawlers start from an initial set of URLs, which may be created manually. Each of the pages identified by these URLs is fetched from the web. The web crawler then locates all URL links in these pages and adds them to the set of URLs to be crawled, if they have not already been fetched, or added to the to-be-crawled set. This process is repeated by fetching all pages in the to-be-crawled set and processing the links in these pages in the same fashion. By repeating the process, all pages that are reachable by any sequence of links from the initial set of URLs would be eventually fetched.

Since the number of documents on the web is very large, it is not possible to crawl the whole web in a short period of time; and in fact, all search engines cover only some portions of the web, not all of it, and their crawlers may take weeks or months to perform a single crawl of all the pages they cover. There are usually many processes, running on multiple machines, involved in crawling. A database stores a set of links (or sites) to be crawled; it assigns links from this set to each crawler process. New links found during a crawl are added to the database and may be crawled later if they are not crawled immediately. Pages have to be refetched (i.e., links recrawled) periodically to obtain updated information and to discard sites that no longer exist, so that the information in the search index is kept reasonably up-to-date.

See the references in the bibliography for a number of practical details in performing a web crawl, such as infinite sequences of links created by dynamically generated pages (called a **spider trap**), prioritization of page fetches, and ensuring that web sites are not flooded by a burst of requests from a crawler.

Pages fetched during a crawl are handed over to a prestige computation and indexing system, which may be running on a different machine. The prestige computation

and indexing systems themselves run on multiple machines in parallel. Pages can be discarded after they are used for prestige computation and added to the index; however, they are usually cached by the search engine to give search engine users fast access to a cached copy of a page, even if the original web site containing the page is not accessible.

It is not a good idea to add pages to the same index that is being used for queries, since doing so would require concurrency control on the index and would affect query and update performance. Instead, one copy of the index is used to answer queries while another copy is updated with newly crawled pages. At periodic intervals the copies switch over, with the old one being updated while the new copy is being used for queries.

To support very high query rates, the indices may be kept in main memory, and there are multiple machines; the system selectively routes queries to the machines to balance the load among them. Popular search engines often have tens of thousands of machines carrying out the various tasks of crawling, indexing, and answering user queries.

web crawlers depend on all relevant pages being reachable through hyperlinks. However, many sites containing large collections of data may not make all the data available as hyperlinked pages. Instead, they provide search interfaces, where users can enter terms or select menu options and get results. As an example, a database of flight information is usually made available using such a search interface, without any hyperlinks to the pages containing flight information. As a result, the information inside such sites is not accessible to a normal web crawler. The information in such sites is often referred to as **deep web** information.

Deep web crawlers extract some such information by guessing what terms would make sense to enter, or what menu options to choose, in such search interfaces. By entering each possible term/option and executing the search interface, they are able to extract pages with data that they would not have been able to find otherwise. The pages extracted by a deep web crawl may be indexed just like regular web pages. The Google search engine, for example, includes results from deep web crawls.

31.8 Information Retrieval: Beyond Ranking of Pages

Information-retrieval systems were originally designed to find textual documents related to a query, and they were later extended to finding pages on the web that are related to a query. People use search engines for many different tasks, from simple tasks such as locating a web site that they want to use, to a broader goal of finding information on a topic of interest. web search engines have become extremely good at the task of locating web sites that a user wants to visit. The task of providing information on a topic of interest is much harder, and we study some approaches in this section.

There is also an increasing need for systems that try to understand documents (to a limited extent) and answer questions based on that (limited) understanding. One approach is to create structured information from unstructured documents and to answer questions based on the structured information. Another approach applies natural lan-

guage techniques to find documents related to a question (phrased in natural language) and return relevant segments of the documents as an answer to the question.

31.8.1 Diversity of Query Results

Today, search engines do not just return a ranked list of web pages relevant to a query. They also return image and video results relevant to a query. Further, there are a variety of sites providing dynamically changing content such as sports scores, or stock market tickers. To get current information from such sites, users would have to first click on the query result. Instead, search engines have created "gadgets," which take data from a particular domain, such as sports updates, stock prices, or weather conditions, and format them in a nice graphical manner, to be displayed as results for a query. Search engines have to rank the set of gadgets available in terms of relevance to a query and display the most relevant gadgets, along with web pages, images, videos, and other types of results. Thus, a query result has a diverse set of result types.

Search terms are often ambiguous. For example, a query "eclipse" may be referring to a solar or lunar eclipse, or to the integrated development environment (IDE) called Eclipse. If all the highly ranked pages for the term *eclipse* are about the IDE, a user looking for information about solar or lunar eclipses may be very dissatisfied. Search engines therefore attempt to provide a set of results that are *diverse* in terms of their topics, to minimize the chance that a user would be dissatisfied. To do so, at indexing time the search engine must disambiguate the sense in which a word is used in a page; for example, it must decide whether the use of the word *eclipse* in a page refers to the IDE or the astronomical phenomenon. Then, given a query, the search engine attempts to provide results that are relevant to the most common senses in which the query words are used.

The results obtained from a web page need to be summarized as a **snippet** in a query result. Traditionally, search engines have provided a few words surrounding the query keywords as a snippet that helps indicate what the page contains. However, there are many domains where the snippet can be generated in a much more meaningful manner. For example, if a user queries about a restaurant, a search engine can generate a snippet containing the restaurant's rating, a phone number, and a link to a map, in addition to providing a link to the restaurant's home page. Such specialized snippets are often generated for results retrieved from a database, for example, a database of restaurants.

31.8.2 Information Extraction

Information-extraction systems convert information from textual form to a more structured form. For example, a real-estate advertisement may describe attributes of a home in textual form, such as "two-bedroom, three-bath house in Queens, $1 million," from which an information extraction system may extract attributes such as number of bedrooms, number of bathrooms, cost, and neighborhood. The original advertisement could have used various terms, such as *2BR*, or two *BR*, or *two bed*, to denote two bedrooms. The extracted information can be used to structure the data in a standard

way. Thus, a user could specify that he is interested in two-bedroom houses, and a search system would be able to return all relevant houses based on the structured data, regardless of the terms used in the advertisement.

An organization that maintains a database of company information may use an information-extraction system to extract information automatically from newspaper articles; the information extracted would relate to changes in attributes of interest, such as resignations, dismissals, or appointments of company officers.

As another example, search engines designed for finding scholarly research articles, such as Citeseer and Google Scholar, crawl the web to retrieve documents that are likely to be research articles. They examine some features of each retrieved document, such as the presence of words such as *bibliography*, *references*, and *abstract*, to judge if a document is in fact a scholarly research article. They then extract the title, list of authors, and the citations at the end of the article by using information extraction techniques. The extracted citation information can be used to link each article to articles that it cites or to articles that cite it; such citation links between articles can be very useful for a researcher.

Several systems have been built for information extraction for specialized applications. They use linguistic techniques, page structure, and user-defined rules for specific domains such as real estate advertisements or scholarly publications. For limited domains, such as a specific web site, it is possible for a human to specify patterns that can be used to extract information. For example, on a particular web site, a pattern such as "Price: <number> $", where <number> indicates any number, may match locations where the price is specified. Such patterns can be created manually for a limited number of web sites.

However, on the web scale with millions of web sites, manual creation of such patterns is not feasible. Machine-learning techniques, which can learn such patterns given a set of training examples, are widely used to automate the process of information extraction.

Information extraction usually has errors in some fraction of the extracted information; typically this is because some page had information in a format that syntactically matched a pattern but did not actually specify a value (such as the price). Information extraction using simple patterns, which separately match parts of a page, is relatively error prone. Machine-learning techniques can perform much more sophisticated analysis, based on interactions between patterns, to minimize errors in the information extracted while maximizing the amount of information extracted. See the references in the bibliographical notes for more information.

31.8.3 Question Answering

Information retrieval systems focus on finding documents relevant to a given query. However, the answer to a query may lie in just one part of a document, or in small parts of several documents. **Question answering** systems attempt to provide direct answers to questions posed by users. For example, a question of the form "Who killed Lincoln?"

may best be answered by a line that says "Abraham Lincoln was shot by John Wilkes Booth in 1865." Note that the answer does not actually contain the words *killed* or *who,* but the system infers that "who" can be answered by a name, and "killed" is related to "shot."

Question answering systems targeted at information on the web typically generate one or more keyword queries from a submitted question, execute the keyword queries against web search engines, and parse returned documents to find segments of the documents that answer the question. A number of linguistic techniques and heuristics are used to generate keyword queries and to find relevant segments from the document.

An issue in answering questions is that different documents may indicate different answers to a question. For example, if the question is "How tall is a giraffe?" different documents may give different numbers as an answer. These answers form a distribution of values, and a question answering system may choose the average, or median value of the distribution as the answer to be returned; to reflect the fact that the answer is not expected to be precise, the system may return the average along with the standard deviation (e.g., average of 16 feet, with a standard deviation of 2 feet), or a range based on the average and the standard deviation (e.g., between 14 and 18 feet).

Current-generation question answering systems are limited in power, since they do not really understand either the question or the documents used to answer the question. However, they are useful for a number of simple question answering tasks.

31.8.4 Querying Structured Data

Structured data are primarily represented in either relational or XML form. Several systems have been built to support keyword querying on relational and XML data (see Chapter 30). A common theme between these systems lies in finding nodes (tuples or XML elements) containing the specified keywords and finding connecting paths (or common ancestors, in the case of XML data) between them.

For example, a query "Zhang Katz" on a university database may find the *name* "Zhang" occurring in a *student* tuple, and the *name* "Katz" in an *instructor* tuple, and a path through the *advisor* relation connecting the two tuples. Other paths, such as student "Zhang" taking a course taught by "Katz," may also be found in response to this query. Such queries may be used for ad hoc browsing and querying of data when the user does not know the exact schema and does not wish to take the effort to write an SQL query defining what she is searching for. Indeed, it is unreasonable to expect lay users to write queries in a structured query language, whereas keyword querying is quite natural.

Since queries are not fully defined, they may have many different types of answers, which must be ranked. A number of techniques have been proposed to rank answers in such a setting, based on the lengths of connecting paths, and on techniques for assigning directions and weights to edges. Techniques have also been proposed for assigning popularity ranks to tuples and XML elements, based on links such as foreign

key and IDREF links. See the bibliographical notes for more information on keyword searching of relational and XML data.

31.9 Directories and Categories

A typical library user may use a catalog to locate a book for which she is looking. When she retrieves the book from the shelf, however, she is likely to *browse* through other books that are located nearby. Libraries organize books in such a way that related books are kept close together. Hence, a book that is physically near the desired book may be of interest as well, making it worthwhile for users to browse through such books.

To keep related books close together, libraries use a **classification hierarchy**. Books on science are classified together. Within this set of books, there is a finer classification, with computer-science books organized together, mathematics books organized together, and so on. Since there is a relation between mathematics and computer science, relevant sets of books are stored close to each other physically. At yet another level in the classification hierarchy, computer-science books are broken down into subareas, such as operating systems, languages, and algorithms. Figure 31.1 illustrates a classification hierarchy that may be used by a library. Because books can be kept at only one place, each book in a library is classified into exactly one spot in the classification hierarchy.

In an information-retrieval system, there is no need to store related documents close together. However, such systems need to *organize documents logically* to permit browsing. Thus, such a system could use a classification hierarchy similar to one that

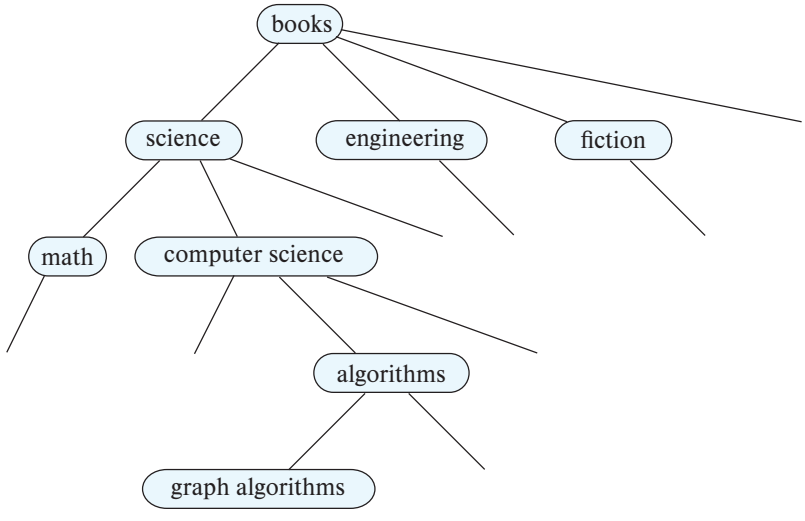

Figure 31.1 A classification hierarchy for a library system.

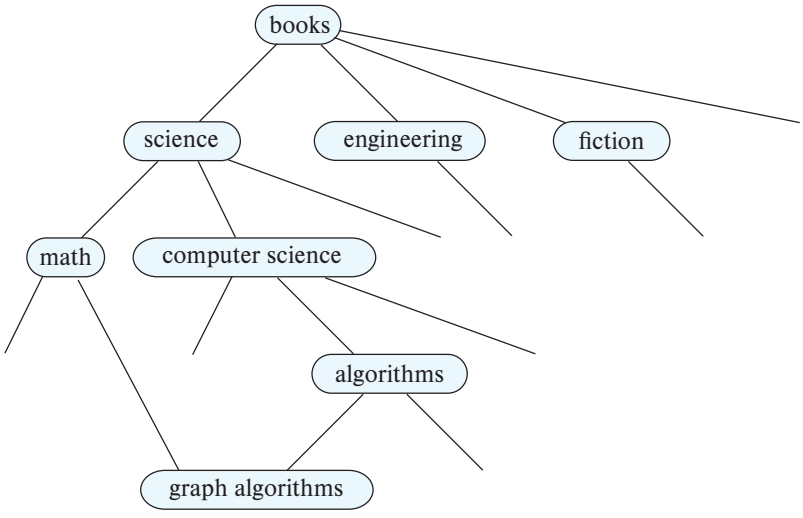

Figure 31.2 A classification DAG for a library information-retrieval system.

libraries use, and, when it displays a particular document, it can also display a brief description of documents that are close in the hierarchy.

In an information-retrieval system, there is no need to keep a document in a single spot in the hierarchy. A document that talks of mathematics for computer scientists could be classified under mathematics as well as under computer science. All that is stored at each spot is an identifier of the document (i.e., a pointer to the document), and it is easy to fetch the contents of the document by using the identifier.

As a result of this flexibility, not only can a document be classified under two locations, but also a subarea in the classification hierarchy can itself occur under two areas. The class of "graph algorithm" documents can appear both under mathematics and under computer science. Thus, the classification hierarchy is now a directed acyclic graph (DAG), as shown in Figure 31.2. A graph-algorithm document may appear in a single location in the DAG but can be reached via multiple paths.

A **directory** is simply a classification DAG structure. Each leaf of the directory stores links to documents on the topic represented by the leaf. Internal nodes may also contain links, for example, to documents that cannot be classified under any of the child nodes.

To find information on a topic, a user would start at the root of the directory and follow paths down the DAG until reaching a node representing the desired topic. While browsing down the directory, the user can find not only documents on the topic he is interested in, but also find related documents and related classes in the classification hierarchy. The user may learn new information by browsing through documents (or subclasses) within the related classes.

Organizing the enormous amount of information available on the web into a directory structure is a daunting task.

- The first problem is determining what exactly the directory hierarchy should be.
- The second problem is, given a document, deciding which nodes of the directory are categories relevant to the document.

To tackle the first problem, portals such as Yahoo! have teams of "Internet librarians" who come up with the classification hierarchy and continually refine it.

The second problem can also be tackled manually by librarians, or web site maintainers may be responsible for deciding where their sites should lie in the hierarchy. There are also techniques for deciding automatically the location of documents based on computing their similarity to documents that have already been classified.

Wikipedia, the online encyclopedia, addresses the classification problem in the reverse direction. Each page in Wikipedia has a list of **categories** to which it belongs. For example, as of 2009, the Wikipedia page on giraffes had several categories including "Mammals of Africa." In turn, the "Mammals of Africa" category itself belongs to the category "Mammals by geography," which in turn belongs to the category "Mammals", which in turn has a category "Vertebrates," and so on. The category structure is useful for browsing other instances of the same category, for example, to find other mammals of Africa, or other mammals. Conversely, a query that looks for mammals can use the category information to infer that a giraffe is a mammal. The Wikipedia category structure is not a tree but is almost a DAG; it is not actually a DAG since it has a few instances of loops, which probably reflect categorization errors.

31.10 Summary

- Information-retrieval systems are used to store and query textual data such as documents. They use a simpler data model than do database systems but provide more powerful querying capabilities within the restricted model.
- Queries attempt to locate documents that are of interest by specifying, for example, sets of keywords. The query that a user has in mind usually cannot be stated precisely; hence, information-retrieval systems order answers on the basis of potential relevance.
- Relevance ranking makes use of several types of information, such as:
  - Term frequency: how important each term is to each document.
  - Inverse document frequency.
  - Popularity ranking.
- Similarity of documents is used to retrieve documents similar to an example document. The cosine metric is used to define similarity and is based on the vector space model.

- PageRank and hub/authority rank are two ways to assign prestige to pages on the basis of links to the page. The PageRank measure can be understood intuitively using a random-walk model. Anchor text information is also used to compute a per-keyword notion of popularity. Information-retrieval systems need to combine scores on multiple factors, such as TF-IDF and PageRank, to get an overall score for a page.
- Search engine spamming attempts to get (an undeserved) high ranking for a page.
- Synonyms and homonyms complicate the task of information retrieval. Concept-based querying aims at finding documents that contain specified concepts, regardless of the exact words (or language) in which the concept is specified. Ontologies are used to relate concepts using relationships such as is-a or part-of.
- Inverted indices are used to answer keyword queries.
- Precision and recall are two measures of the effectiveness of an information-retrieval system.
- web search engines crawl the web to find pages, analyze them to compute prestige measures, and index them.
- Techniques have been developed to extract structured information from textual data, to perform keyword querying on structured data, and to give direct answers to simple questions posed in natural language.
- Directory structures and categories are used to classify documents with other similar documents.

Review Terms

- Information-retrieval systems
- Keyword search
- Full text retrieval
- Term
- Relevance ranking
  - Term frequency
  - Inverse document frequency
  - Relevance
  - Proximity
- Similarity-based retrieval
  - Vector space model
  - Cosine similarity metric
  - Relevance feedback
- Stop words
- Relevance using hyperlinks
  - Popularity/prestige
  - Transfer of prestige
- PageRank
  - Random walk model
- Anchor-text – based relevance
- Hub/authority ranking

- Search engine spamming
- Synonyms
- Homonyms
- Concepts
- Concept-based querying
- Ontologies
- Semantic web
- Inverted index
- False drop
- False negative
- False positive
- Precision
- Recall
- web crawlers
- Deep web
- Query result diversity
- Information extraction
- Question answering
- Querying structured data
- Directories
- Classification hierarchy
- Categories

Practice Exercises

31.1 Compute the relevance (using appropriate definitions of term frequency and inverse document frequency) of each of the Practice Exercises in this chapter to the query "SQL relation".

31.2 Suppose you want to find documents that contain at least k of a given set of n keywords. Suppose also you have a keyword index that gives you a (sorted) list of identifiers of documents that contain a specified keyword. Give an efficient algorithm to find the desired set of documents.

31.3 Suggest how to implement the iterative technique for computing PageRank given that the T matrix (even in adjacency list representation) does not fit in memory.

31.4 Suggest how a document containing a word (such as *leopard*) can be indexed such that it is efficiently retrieved by queries using a more general concept (such as "carnivore" or "mammal"). You can assume that the concept hierarchy is not very deep, so each concept has only a few generalizations (a concept can, however, have a large number of specializations). You can also assume that you are provided with a function that returns the concept for each word in a document. Also suggest how a query using a specialized concept can retrieve documents using a more general concept.

31.5 Suppose inverted lists are maintained in blocks, with each block noting the largest popularity rank and TF-IDF scores of documents in the remaining blocks in the list. Suggest how merging of inverted lists can stop early if the user wants only the top K answers.

Exercises

31.6 Using a simple definition of term frequency as the number of occurrences of the term in a document, give the TF-IDF scores of each term in the set of documents consisting of this and the next exercise.

31.7 Create a small example of four small documents, each with a PageRank, and create inverted lists for the documents sorted by the PageRank. You do not need to compute PageRank, just assume some values for each page.

31.8 Suppose you wish to perform keyword querying on a set of tuples in a database, where each tuple has only a few attributes, each containing only a few words. Does the concept of term frequency make sense in this context? And that of inverse document frequency? Explain your answer. Also suggest how you can define the similarity of two tuples using TF-IDF concepts.

31.9 Web sites that want to get some publicity can join a web ring, where they create links to other sites in the ring in exchange for other sites in the ring creating links to their site. What is the effect of such rings on popularity ranking techniques such as PageRank?

31.10 The Google search engine provides a feature whereby Web sites can display advertisements supplied by Google. The advertisements supplied are based on the contents of the page. Suggest how Google might choose which advertisements to supply for a page, given the page contents.

31.11 One way to create a keyword-specific version of PageRank is to modify the random jump such that a jump is only possible to pages containing the keyword. Thus, pages that do not contain the keyword but are close (in terms of links) to pages that contain the keyword also get a nonzero rank for that keyword.

a. Give equations defining such a keyword-specific version of PageRank.

b. Give a formula for computing the relevance of a page to a query containing multiple keywords.

31.12 The idea of popularity ranking using hyperlinks can be extended to relational and XML data, using foreign key and IDREF edges in place of hyperlinks. Suggest how such a ranking scheme may be of value in the following applications:

a. A bibliographic database that has links from articles to authors of the articles and links from each article to every article that it references.

b. A sales database that has links from each sales record to the items that were sold.

Also suggest why prestige ranking can give less than meaningful results in a movie database that records which actor has acted in which movies.

31.13 What is the difference between a false positive and a false drop? If it is essential that no relevant information be missed by an information-retrieval query, is it acceptable to have either false positives or false drops? Why?

Tools

Google (http://www.google.com) is currently the most popular search engine, but there are a number of other search engines, such as Microsoft Bing (http://www.bing.com) and Yahoo! search (search.yahoo.com). The site searchenginewatch.com provides a wealth of information about search engines. Yahoo! (dir.yahoo.com) and the Open Directory Project (dmoz.org) provide classification hierarchies for web sites.

Further Reading

[Manning et al. (2008)], [Chakrabarti (2002)], [Grossman and Frieder (2004)], [Witten et al. (1999)], and [Baeza-Yates and Ribeiro-Neto (1999)] provide textbook descriptions of information retrieval. In particular, [Chakrabarti (2002)] and [Manning et al. (2008)] provide detailed coverage of web crawling, ranking techniques, and mining techniques related to information retrieval such as text classification and clustering.

Bibliography

[Baeza-Yates and Ribeiro-Neto (1999)] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley (1999).

[Chakrabarti (2002)] S. Chakrabarti, *Mining the Web: Discovering Knowledge from HyperText Data*, Morgan Kaufmann (2002).

[Grossman and Frieder (2004)] D. A. Grossman and O. Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd edition, Springer Verlag (2004).

[Manning et al. (2008)] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press (2008).

[Witten et al. (1999)] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd edition, Morgan Kaufmann (1999).

Credits

The photo of the sailboats in the beginning of the chapter is due to ©Pavel Nesvadba/Shutterstock.