

22.3 INFORMATION RETRIEVAL

INFORMATION
RETRIEVAL

Information retrieval is the task of finding documents that are relevant to a user’s need for information. The best-known examples of information retrieval systems are search engines on the World Wide Web. A Web user can type a query such as [AI book]² into a search engine and see a list of relevant pages. In this section, we will see how such systems are built. An information retrieval (henceforth **IR**) system can be characterized by

IR

1. **A corpus of documents.** Each system must decide what it wants to treat as a document: a paragraph, a page, or a multipage text.

QUERY LANGUAGE

2. **Queries posed in a query language.** A query specifies what the user wants to know. The query language can be just a list of words, such as [AI book]; or it can specify a phrase of words that must be adjacent, as in [“AI book”]; it can contain Boolean operators as in [AI AND book]; it can include non-Boolean operators such as [AI NEAR book] or [AI book site:www.aaii.org].

RESULT SET

RELEVANT

3. **A result set.** This is the subset of documents that the IR system judges to be **relevant** to the query. By *relevant*, we mean likely to be of use to the person who posed the query, for the particular information need expressed in the query.

PRESENTATION

4. **A presentation of the result set.** This can be as simple as a ranked list of document titles or as complex as a rotating color map of the result set projected onto a three-dimensional space, rendered as a two-dimensional display.

BOOLEAN KEYWORD
MODEL

The earliest IR systems worked on a **Boolean keyword model**. Each word in the document collection is treated as a Boolean feature that is true of a document if the word occurs in the document and false if it does not. So the feature “retrieval” is true for the current chapter but false for Chapter 15. The query language is the language of Boolean expressions over

² We denote a search query as [*query*]. Square brackets are used rather than quotation marks so that we can distinguish the query [“two words”] from [two words].

features. A document is relevant only if the expression evaluates to true. For example, the query [information AND retrieval] is true for the current chapter and false for Chapter 15.

This model has the advantage of being simple to explain and implement. However, it has some disadvantages. First, the degree of relevance of a document is a single bit, so there is no guidance as to how to order the relevant documents for presentation. Second, Boolean expressions are unfamiliar to users who are not programmers or logicians. Users find it unintuitive that when they want to know about farming in the states of Kansas *and* Nebraska they need to issue the query [farming (Kansas OR Nebraska)]. Third, it can be hard to formulate an appropriate query, even for a skilled user. Suppose we try [information AND retrieval AND models AND optimization] and get an empty result set. We could try [information OR retrieval OR models OR optimization], but if that returns too many results, it is difficult to know what to try next.

22.3.1 IR scoring functions

Most IR systems have abandoned the Boolean model and use models based on the statistics of word counts. We describe the **BM25 scoring function**, which comes from the Okapi project of Stephen Robertson and Karen Sparck Jones at London's City College, and has been used in search engines such as the open-source Lucene project.

A scoring function takes a document and a query and returns a numeric score; the most relevant documents have the highest scores. In the BM25 function, the score is a linear weighted combination of scores for each of the words that make up the query. Three factors affect the weight of a query term: First, the frequency with which a query term appears in a document (also known as *TF* for term frequency). For the query [farming in Kansas], documents that mention “farming” frequently will have higher scores. Second, the inverse document frequency of the term, or *IDF*. The word “in” appears in almost every document, so it has a high document frequency, and thus a low inverse document frequency, and thus it is not as important to the query as “farming” or “Kansas.” Third, the length of the document. A million-word document will probably mention all the query words, but may not actually be about the query. A short document that mentions all the words is a much better candidate.

The BM25 function takes all three of these into account. We assume we have created an index of the N documents in the corpus so that we can look up $TF(q_i, d_j)$, the count of the number of times word q_i appears in document d_j . We also assume a table of document frequency counts, $DF(q_i)$, that gives the number of documents that contain the word q_i . Then, given a document d_j and a query consisting of the words $q_{1:N}$, we have

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^N IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k + 1)}{TF(q_i, d_j) + k \cdot (1 - b + b \cdot \frac{|d_j|}{L})},$$

where $|d_j|$ is the length of document d_j in words, and L is the average document length in the corpus: $L = \sum_i |d_i| / N$. We have two parameters, k and b , that can be tuned by cross-validation; typical values are $k = 2.0$ and $b = 0.75$. $IDF(q_i)$ is the inverse document

frequency of word q_i , given by

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}.$$

Of course, it would be impractical to apply the BM25 scoring function to every document in the corpus. Instead, systems create an **index** ahead of time that lists, for each vocabulary word, the documents that contain the word. This is called the **hit list** for the word. Then when given a query, we intersect the hit lists of the query words and only score the documents in the intersection.

INDEX
HIT LIST

22.3.2 IR system evaluation

How do we know whether an IR system is performing well? We undertake an experiment in which the system is given a set of queries and the result sets are scored with respect to human relevance judgments. Traditionally, there have been two measures used in the scoring: recall and precision. We explain them with the help of an example. Imagine that an IR system has returned a result set for a single query, for which we know which documents are and are not relevant, out of a corpus of 100 documents. The document counts in each category are given in the following table:

	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

PRECISION
RECALL

Precision measures the proportion of documents in the result set that are actually relevant. In our example, the precision is $30/(30 + 10) = .75$. The false positive rate is $1 - .75 = .25$. **Recall** measures the proportion of all the relevant documents in the collection that are in the result set. In our example, recall is $30/(30 + 20) = .60$. The false negative rate is $1 - .60 = .40$. In a very large document collection, such as the World Wide Web, recall is difficult to compute, because there is no easy way to examine every page on the Web for relevance. All we can do is either estimate recall by sampling or ignore recall completely and just judge precision. In the case of a Web search engine, there may be thousands of documents in the result set, so it makes more sense to measure precision for several different sizes, such as “P@10” (precision in the top 10 results) or “P@50,” rather than to estimate precision in the entire result set.

It is possible to trade off precision against recall by varying the size of the result set returned. In the extreme, a system that returns every document in the document collection is guaranteed a recall of 100%, but will have low precision. Alternately, a system could return a single document and have low recall, but a decent chance at 100% precision. A summary of both measures is the F_1 score, a single number that is the harmonic mean of precision and recall, $2PR/(P + R)$.

22.3.3 IR refinements

There are many possible refinements to the system described here, and indeed Web search engines are continually updating their algorithms as they discover new approaches and as the Web grows and changes.

One common refinement is a better model of the effect of document length on relevance. Singhal *et al.* (1996) observed that simple document length normalization schemes tend to favor short documents too much and long documents not enough. They propose a *pivoted* document length normalization scheme; the idea is that the pivot is the document length at which the old-style normalization is correct; documents shorter than that get a boost and longer ones get a penalty.

The BM25 scoring function uses a word model that treats all words as completely independent, but we know that some words are correlated: “couch” is closely related to both “couches” and “sofa.” Many IR systems attempt to account for these correlations.

CASE FOLDING
STEMMING

For example, if the query is [couch], it would be a shame to exclude from the result set those documents that mention “COUCH” or “couches” but not “couch.” Most IR systems do **case folding** of “COUCH” to “couch,” and some use a **stemming** algorithm to reduce “couches” to the stem form “couch,” both in the query and the documents. This typically yields a small increase in recall (on the order of 2% for English). However, it can harm precision. For example, stemming “stocking” to “stock” will tend to decrease precision for queries about either foot coverings or financial instruments, although it could improve recall for queries about warehousing. Stemming algorithms based on rules (e.g., remove “-ing”) cannot avoid this problem, but algorithms based on dictionaries (don’t remove “-ing” if the word is already listed in the dictionary) can. While stemming has a small effect in English, it is more important in other languages. In German, for example, it is not uncommon to see words like “Lebensversicherungsgesellschaftsangestellter” (life insurance company employee). Languages such as Finnish, Turkish, Inuit, and Yupik have recursive morphological rules that in principle generate words of unbounded length.

SYNONYM

The next step is to recognize **synonyms**, such as “sofa” for “couch.” As with stemming, this has the potential for small gains in recall, but can hurt precision. A user who gives the query [Tim Couch] wants to see results about the football player, not sofas. The problem is that “languages abhor absolute synonyms just as nature abhors a vacuum” (Cruse, 1986). That is, anytime there are two words that mean the same thing, speakers of the language conspire to evolve the meanings to remove the confusion. Related words that are not synonyms also play an important role in ranking—terms like “leather”, “wooden,” or “modern” can serve to confirm that the document really is about “couch.” Synonyms and related words can be found in dictionaries or by looking for correlations in documents or in queries—if we find that many users who ask the query [new sofa] follow it up with the query [new couch], we can in the future alter [new sofa] to be [new sofa OR new couch].

METADATA

As a final refinement, IR can be improved by considering **metadata**—data outside of the text of the document. Examples include human-supplied keywords and publication data. On the Web, hypertext **links** between documents are a crucial source of information.

LINKS

22.3.4 The PageRank algorithm

PAGERANK

PageRank³ was one of the two original ideas that set Google’s search apart from other Web search engines when it was introduced in 1997. (The other innovation was the use of anchor

³ The name stands both for Web pages and for coinventor Larry Page (Brin and Page, 1998).

```

function HITS(query) returns pages with hub and authority numbers

  pages ← EXPAND-PAGES(RELEVANT-PAGES(query))
  for each p in pages do
    p.AUTHORITY ← 1
    p.HUB ← 1
  repeat until convergence do
    for each p in pages do
      p.AUTHORITY ←  $\sum_i \text{INLINK}_i(p).\text{HUB}$ 
      p.HUB ←  $\sum_i \text{OUTLINK}_i(p).\text{AUTHORITY}$ 
    NORMALIZE(pages)
  return pages

```

Figure 22.1 The HITS algorithm for computing hubs and authorities with respect to a query. RELEVANT-PAGES fetches the pages that match the query, and EXPAND-PAGES adds in every page that links to or is linked from one of the relevant pages. NORMALIZE divides each page’s score by the sum of the squares of all pages’ scores (separately for both the authority and hubs scores).

text—the underlined text in a hyperlink—to index a page, even though the anchor text was on a *different* page than the one being indexed.) PageRank was invented to solve the problem of the tyranny of *TF* scores: if the query is [IBM], how do we make sure that IBM’s home page, `ibm.com`, is the first result, even if another page mentions the term “IBM” more frequently? The idea is that `ibm.com` has many in-links (links to the page), so it should be ranked higher: each in-link is a vote for the quality of the linked-to page. But if we only counted in-links, then it would be possible for a Web spammer to create a network of pages and have them all point to a page of his choosing, increasing the score of that page. Therefore, the PageRank algorithm is designed to weight links from high-quality sites more heavily. What is a high-quality site? One that is linked to by other high-quality sites. The definition is recursive, but we will see that the recursion bottoms out properly. The PageRank for a page *p* is defined as:

$$PR(p) = \frac{1 - d}{N} + d \sum_i \frac{PR(in_i)}{C(in_i)},$$

where $PR(p)$ is the PageRank of page *p*, *N* is the total number of pages in the corpus, *in_i* are the pages that link in to *p*, and $C(in_i)$ is the count of the total number of out-links on page *in_i*. The constant *d* is a damping factor. It can be understood through the **random surfer model**: imagine a Web surfer who starts at some random page and begins exploring. With probability *d* (we’ll assume $d = 0.85$) the surfer clicks on one of the links on the page (choosing uniformly among them), and with probability $1 - d$ she gets bored with the page and restarts on a random page anywhere on the Web. The PageRank of page *p* is then the probability that the random surfer will be at page *p* at any point in time. PageRank can be computed by an iterative procedure: start with all pages having $PR(p) = 1$, and iterate the algorithm, updating ranks until they converge.

22.3.5 The HITS algorithm

The Hyperlink-Induced Topic Search algorithm, also known as “Hubs and Authorities” or HITS, is another influential link-analysis algorithm (see Figure 22.1). HITS differs from PageRank in several ways. First, it is a query-dependent measure: it rates pages with respect to a query. That means that it must be computed anew for each query—a computational burden that most search engines have elected not to take on. Given a query, HITS first finds a set of pages that are relevant to the query. It does that by intersecting hit lists of query words, and then adding pages in the link neighborhood of these pages—pages that link to or are linked from one of the pages in the original relevant set.

AUTHORITY

HUB

Each page in this set is considered an **authority** on the query to the degree that other pages in the relevant set point to it. A page is considered a **hub** to the degree that it points to other authoritative pages in the relevant set. Just as with PageRank, we don’t want to merely count the number of links; we want to give more value to the high-quality hubs and authorities. Thus, as with PageRank, we iterate a process that updates the authority score of a page to be the sum of the hub scores of the pages that point to it, and the hub score to be the sum of the authority scores of the pages it points to. If we then normalize the scores and repeat k times, the process will converge.

Both PageRank and HITS played important roles in developing our understanding of Web information retrieval. These algorithms and their extensions are used in ranking billions of queries daily as search engines steadily develop better ways of extracting yet finer signals of search relevance.

22.3.6 Question answering

QUESTION
ANSWERING

Information retrieval is the task of finding documents that are relevant to a query, where the query may be a question, or just a topic area or concept. **Question answering** is a somewhat different task, in which the query really is a question, and the answer is not a ranked list of documents but rather a short response—a sentence, or even just a phrase. There have been question-answering NLP (natural language processing) systems since the 1960s, but only since 2001 have such systems used Web information retrieval to radically increase their breadth of coverage.

The ASKMSR system (Banko *et al.*, 2002) is a typical Web-based question-answering system. It is based on the intuition that most questions will be answered many times on the Web, so question answering should be thought of as a problem in precision, not recall. We don’t have to deal with all the different ways that an answer might be phrased—we only have to find one of them. For example, consider the query [Who killed Abraham Lincoln?] Suppose a system had to answer that question with access only to a single encyclopedia, whose entry on Lincoln said

John Wilkes Booth altered history with a bullet. He will forever be known as the man who ended Abraham Lincoln’s life.

To use this passage to answer the question, the system would have to know that ending a life can be a killing, that “He” refers to Booth, and several other linguistic and semantic facts.

ASKMSR does not attempt this kind of sophistication—it knows nothing about pronoun reference, or about killing, or any other verb. It does know 15 different kinds of questions, and how they can be rewritten as queries to a search engine. It knows that [Who killed Abraham Lincoln] can be rewritten as the query [* killed Abraham Lincoln] and as [Abraham Lincoln was killed by *]. It issues these rewritten queries and examines the results that come back—not the full Web pages, just the short summaries of text that appear near the query terms. The results are broken into 1-, 2-, and 3-grams and tallied for frequency in the result sets and for weight: an n -gram that came back from a very specific query rewrite (such as the exact phrase match query [“Abraham Lincoln was killed by *”]) would get more weight than one from a general query rewrite, such as [Abraham OR Lincoln OR killed]. We would expect that “John Wilkes Booth” would be among the highly ranked n -grams retrieved, but so would “Abraham Lincoln” and “the assassination of” and “Ford’s Theatre.”

Once the n -grams are scored, they are filtered by expected type. If the original query starts with “who,” then we filter on names of people; for “how many” we filter on numbers, for “when,” on a date or time. There is also a filter that says the answer should not be part of the question; together these should allow us to return “John Wilkes Booth” (and not “Abraham Lincoln”) as the highest-scoring response.

In some cases the answer will be longer than three words; since the components responses only go up to 3-grams, a longer response would have to be pieced together from shorter pieces. For example, in a system that used only bigrams, the answer “John Wilkes Booth” could be pieced together from high-scoring pieces “John Wilkes” and “Wilkes Booth.”

At the Text Retrieval Evaluation Conference (TREC), ASKMSR was rated as one of the top systems, beating out competitors with the ability to do far more complex language understanding. ASKMSR relies upon the breadth of the content on the Web rather than on its own depth of understanding. It won’t be able to handle complex inference patterns like associating “who killed” with “ended the life of.” But it knows that the Web is so vast that it can afford to ignore passages like that and wait for a simple passage it can handle.
