Chapter 18
Information Retrieval

Abstract Information retrieval (IR) is the identification of documents or other units of information in a collection that are relevant to a particular information need—a set of questions to which someone would like to find an answer. This chapter presents the basic strategies of IR in length along with their analysis, particularly emphasizing the vector space and probabilistic models of IR, with worked examples in each category; gives the detailed coverage to construction and maintenance of index, and its parallel processing. The fuzzy logic-based retrieval, concept-based retrieval techniques, their algorithms, and worked examples are presented; and Automatic Query Expansion has been dealt with at length. Application of Bayesian networks, and inferences using these have been demonstrated for IR. The newly emerged semantic web for futuristic IR and its applications have been introduced; and the design aspects of distributed IR suited for currently distributed information resources are treated in depth. The chapter ends with the summary and a set of practice exercises.

Keywords Information retrieval · IR · Vector space model · Boolean model · Probabilistic model · Fuzzy-based IR · Concept-based IR · Automatic Query Expansion · Indexing · Parallel index · Distributed index · Semantic web · Parallel IR · Distributed IR · Query expansion · Bayesian networks

18.1 Introduction

The problem of Information Retrieval in the present context is highly relevant when the volume of information generated is much more than the individuals can easily digest. Consequently, it is extremely difficult to search, locate, and disseminate the precisely desired information from the storage media, irrespective of whether it is local or globally distributed. Ever-increasing information needs to be continually added to the storage systems. This process has been fueled further by the arrival of the Internet and the World Wide Web, as well as digital libraries, research publications repositories, electronic editions of newspapers, journals, and magazines.

Given a set of documents or information collection and information need, Information Retrieval (IR) identifies the documents or other units of information in the

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_18

collection, which are relevant to that particular information need. The information need is in the form of a set of questions one is interested to find an answer to. The aim of IR is to locate the document or file which holds the desired information. Sometimes, it is also required to locate the actual position of the required information in the document selected when the document is of a large size.

Here are some examples of IR tasks: finding an article in the Times of India that discusses the freedom struggle of India; searching the recent postings in Twitter that are related to a particular model of smart phones; finding the entries referring to butterflies in an online encyclopedia, etc. [2].

The early IR methods were based on manually assigned keywords to the documents and required complicated Boolean queries. These methods were primarily used by retrieval experts. However, in the 1970s, automatic indexing and natural language queries gained popularity, and the IR facility became more and more available to non-experts. The documents were commonly indexed by automatically considering all the terms in them as independent words. The documents were represented using the set of these keywords, known as Bag-of-Words (BOW). The query formatting also became simplified in the form of short natural language formulation. This, however, added noise in the documents' representation, but the basic methodology for indexing remained the same. Due to this, the non-expert users faced increasingly more troubles, due to the "vocabulary problem". This was because the keywords chosen were often different than those used by the authors of the relevant documents. Due to this, systems' *recall*[1] rates came down. In other cases, the contextual differences between ambiguous keywords were not properly resolved through the BOW's approach, which reduced the *precision* of the results. These two problems are generally referred to as *synonymy* and *polysemy*, respectively [16].

To solve the problem of polysemy,[2] automatic Word Sense Disambiguation algorithms helped to disambiguate the documents' contents as well as the query. The disambiguation algorithms make use of resources like WordNet Thesaurus, corpora text, or co-occurrence data to find the possible senses of a word and map word occurrences to the correct sense. The disambiguated senses are used in indexing and in query processing, which would help to retrieve only the documents that match the correct sense. Unfortunately, sometimes the inaccuracy and errors of automatic disambiguation are more dangerous than not using the disambiguation at all [5].

The models of Information retrieval assume each document described by a set of representative keywords, called *index* terms. An index term is a word from a document, which is supposed to represent the semantics of the document. In general the index terms are *nouns*, as nouns carry maximum meaning of a sentence. Each of the index terms in a document is not equally important; some of the index terms describe the given document better than others. An index term which appears in every document in a given set of documents is of no use. However, a term which appears in only, say, five documents out of thousands of documents should distinctly identify

[1] *Precision* and *recall* are parameters which represent the performance of any IR system.

[2] *Polysemy* is an ambiguity in an individual word or phrase, such that the word can be used (in different contexts) to express two or more different meanings.

those documents. Accordingly, an index term should be assigned some numerical *weight* to capture this effect. If k_i is an index term, d_j is a document then $w_{i,j} \geq 0$ is the weight associated with the (keyword, document) pair (k_i, d_j).

Consider that there are a number of documents, and a query is put to retrieve the documents relevant to this query. Ideally, the retrieval algorithm must return all the documents that are relevant to that query, i.e., retrieved documents (say Ret) are the same set as the relevant documents (say, Rel) in the entire set of documents. However, in a realistic situation, systems may retrieve many non-relevant documents also along with the relevant ones. To measure the effectiveness of retrieval, two ratios, *precision* and *recall*, are used. The precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved, and it provides the quality of the answer set. However, this does not consider the total number of relevant documents available in the original set. A system may have 90% precision if nine documents are relevant out of total ten that it has retrieved, and there are a total of 100 relevant documents in the store. Hence, the total relevant documents available also matters. The *recall* is ratio of number of relevant documents retrieved to the total number of relevant documents in the entire collection (see Fig. 18.1).

Intuitively, to keep precision high, one need to be overly careful (assuming that human is itself an algorithm). In that case too a few documents will get retrieved. But, in that process many useful documents would be left behind, making recall very low. On the other hand, if one puts efforts to achieve high recall, i.e., trying to retrieve the maximum number of relevant documents out of the total relevant count (a case of too much aggressiveness), many non-relevant documents may also be retrieved, resulting in poor precision. A typical such scenario is shown in Fig. 18.2.

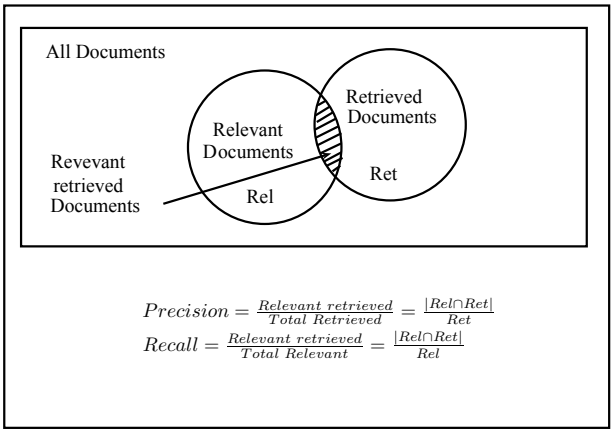

Fig. 18.1 Precision and Recall

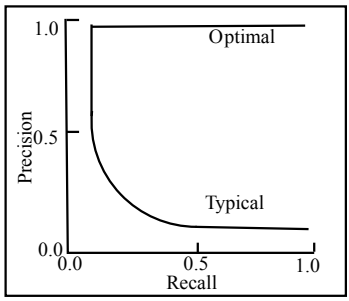

Fig. 18.2 Typical relation between Precision and Recall

Learning Outcomes of This Chapter:

1. Explain basic information storage and retrieval concepts. [Familiarity]
2. Describe what issues are specific to efficient information retrieval. [Familiarity]
3. Give applications of alternative search strategies and explain why the particular search strategy is appropriate for the application. [Assessment]
4. Design and implement a small to medium size information storage and retrieval system, or digital library. [Usage]
5. Describe some of the technical solutions to the problems related to archiving and preserving information in a digital library. [Familiarity]
6. Generate an index file for a collection of resources. [Usage]
7. Explain the role of an inverted index in locating a document in a collection. [Familiarity]
8. Explain how stemming and stop words affect indexing. [Familiarity]
9. Describe key challenges in web crawling, e.g., detecting duplicate documents, determining the crawling frontier. [Familiarity]

18.2 Retrieval Strategies

Consider a set of documents $D_1, D_2, \ldots, D_n$ and query Q, a retrieval strategy is an algorithm for information retrieval that assigns a similarity measure $sim(Q, D_i)$ for $i = 1, 2, \ldots, n$, between query Q and document set D. Following are the most common retrieval strategies [14].

- *Boolean Indexing*. A Boolean query results in ranking based on some score assigned to the terms. This can be achieved by associating a weight with each query term, which can be used to compute the similarity coefficient.
- *Vector Space Model*. In this model, the document D_i and query Q are each represented as vectors in the *term* space. The model computes the similarity $sim(Q, D_i)$ between two vectors.
- *Probabilistic Model*. A probability of relevance of a document to a query is based on the *likelihood* that a term (i.e., query term) will appear in a relevant document. This

probability is computed for each term in the collection of documents. For terms that match between a query and document, the similarity measure is computed as a combination of probabilities of each of the matching terms. This similarity is a measure of relevance of the query to the document.

- *Inference networks*. A Bayesian network is used to infer the relevance of a document to a query, the later is a measure of similarity between the query and the document.
- *Fuzzy set-based retrieval*. In this IR approach, a document is mapped to a *fuzzy set*.[3] For example, *rain*/0.9 versus *rain*/0.2 shows that in the first case, the meaning of the keyword *rain* indicates heavy rain, while in the second set it shows a very mild rain [7].

18.3 Boolean Model of IR System

The Boolean model for IR is a simple information retrieval model, based on set theory and Boolean algebra. Since the concept of set is quite intuitive, the Boolean model provides the framework that is easy to grasp by a common user of an IR system, as well as simple to implement. Also, the Boolean queries in the form of Boolean expressions, are precise in their semantics [14].

The Boolean model considers that the *index terms*, which are the representation of documents, are either present or absent in a document. Consequently, the index terms' weights are taken as binary (1 for presence and 0 for absence), i.e., for a document D_i the index term t_j is either present or absent in D_i, i.e., weight $w_{ij} \in \{0, 1\}$. A query Q is composed of index terms connected through the binary operators $\wedge$, $\vee$, $\neg$ (and, or, not). A query can be expressed as *conjunctive normal form* (CNF) or *disjunctive normal form* (DNF). For example the query "$Q = t_a \wedge (t_b \vee \neg t_c)$" = "$(t_a \wedge t_b) \vee (t_a \wedge \neg t_c)$" can be written in DNF as $\overrightarrow{Q} = (1, 0, 0) \vee (1, 1, 0) \vee (1, 1, 1)$, where each term is a binary weighted vector corresponding to the tuple (t_a, t_b, t_c). Figure 18.3 represent this query using a Venn diagram.

IR is a process of matching the *patterns* in the user *inquiry* with the patterns in the prospective *text documents*. If the inquiry words are taken as a set of words X, and the text document words are taken as a set of words Y, then IR is nothing but finding the binary relation $X \times Y$. However, words may appear in different morphological forms. It is, therefore, necessary that—before the matching is performed, the words in the inquiry as well as the words in the text document are reduced to their basic form—called *stem words*, by a process called *stemming*.

Consider that X is a Boolean set inquiry (or query) keywords, Y is a Boolean set representing the keywords in the document. With this, a binary relation from X to Y can be represented as

[3] A set that contained not only the elements but a number associated with each element that indicates the strength of the membership of the term.

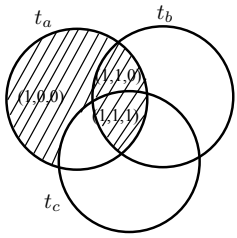

Fig. 18.3 Conjunctive components of query $Q = t_a \wedge (t_b \vee \neg t_c)$

$$R : X \times Y \in \{0, 1\}. \tag{18.1}$$

If for every $x \in X$, there is a corresponding $y \in Y$, then we say $R(x, y) = 1$, i.e., x is related to y, otherwise $R(x, y) = 0$.

Since in a Boolean model a query term x is either related to a document term y (i.e., $R(x, y) = 1$), or not related to y ($R(x, y) = 0$), there are sharp boundaries of relations, hence a Boolean-based model is also called *Crisp set* -based model. The following example demonstrates crisp set-based IR.

Example 18.1 Crisp set-based Information Retrieval.

Consider a set of text documents Y consisting of documents $y_1, y_2, ..., y_n$, as potential documents to be searched for the keywords $x_1, x_2, ..., x_m$ in the inquiry set X. Assume that $n = 2$, and y_1, y_2 are as given below: (quoted from, "Einstein—The Life and Times," by Ronald W. Clark).

y_1: "Thus the new concept of the subatomic world was even by 1920 beginning to produce a gulf. Bohr, Born, and a number of Einstein's other contemporaries, as well as the many of younger men who were in great part responsible for the new idea readily jumped the gap. Einstein stayed where he was. Therefore, the scene in many ways paralleled that into which he has launched his theory of relativity two decades earlier. But then he had been in the iconoclastic vanguard; now he took up station with the small conservative rearguard."

y_2: "Plank, the man of honor who had yet not signed the manifesto of 93, had in fact for the first time done as much to keep Einstein in Berlin as he had done to bring him there in 1914. His letter, which, in Einstein's words, had induced him ...."

Let the inquiry be "Einstein's Scientific Theory of Relativity"; and therefore the corresponding set of keywords in the inquiry is $X = \{x_1 =$ "Einstein", $x_2 =$ "Scientific", $x_3 =$ "Theory of Relativity" $\}$, and document set Y consists of documents y_1, y_2, therefore, $Y = \{y_1, y_2\}$.

In crisp set-based IR, it is required to find R—a subset of $X \times Y$, i.e., R ("Einstein", y_1), R ("Scientific", y_1), R ("Theory of Relativity", y_1), R ("Einstein", y_2), R ("Scientific", y_2), R ("Theory of Relativity", y_2). The Boolean relation R for the example under consideration is given in Table 18.1.

It may be seen that while y_1 has two matchings, y_2 has only one matching. Hence, Table 18.1 shows that the relative relevance of y_1 with respect to y_2, given by the sum of matching counts in y_1 divided by the sum of matching counts in y_2 is 2. For larger

Table 18.1 Crisp set relation between query and document

Query (X)	Document (Y)	
	y_1	y_2
x_1	1	1
x_2	0	0
x_3	1	0

number of documents: y_1, y_2, …, y_n, the relevance can be computed in the similar manner for a specified inquiry. Once the relevance is computed, the IR system lists the documents in the order of their relevance.

In the classical binary logic, there is either 100% match for an index term in the inquiry and corresponding term in the document, so $R(x_i, y_i) = 1$; or there is no match at all with $R(x_i, y_i) = 0$. However, the real-life situations are often different. It may happen that the two terms from index and document which are being matched are two forms of the same word, e.g., *real* and *reality*, *phrase* and *phrases*, *exact* and *exactness*, etc. In all these cases the crisp logic returns zero value of relevance. □

18.4 Vector Space Model

The vector space model determines the measure of similarity ($sim(Q, D_i)$) between a query and a document using a vector that represents query Q, and document D_i. The model is based on the intuitive notion that meaning of a document is conveyed by the words present in the document. Figure 18.4 represents the concepts of vectors for a query Q and documents $D_1, D_2, \ldots, D_n$. Here t is the total number of terms in the collection [12].

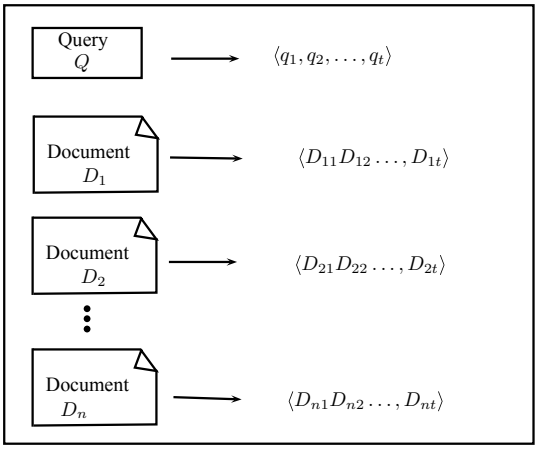

Fig. 18.4 Vector space model

The vector space model is based on a method which compares how close the query vector is to the document vector. The traditional method of determining the closeness between two vectors is the angle between them, where the angle is computed using dot products of two vectors. In our context here, the similarity coefficient is used instead of the angle. If a term is present in the vector, 1 is placed else 0 is placed in the corresponding position in the vector. To account for multiple occurrences of a term in the document, frequency of that term is used instead of merely its presence/absence. Thus, for a query $\langle a, b \rangle$, two documents' vectors, $\langle 1, 0 \rangle$, and $\langle 5, 1 \rangle$ indicate that in the first terms a and b have frequency of 1 and 0, respectively, and for the second these frequencies are 5 and 1, respectively.

The similarity between query and documents can be computed as the distance between the query and each of the document vectors. If a document has the same vector as the query, their distance is minimum and have the highest similarity.

Instead of specifying the list of terms in a query, a user is often interested to indicate that certain terms are more important than others. One approach to this is that user indicates a higher weight to specific terms by manually specifying the weight. The other approach is automatically assigning the weight equal to the number of times (frequency of the term) a term appears in a document. But, if a common term appears in every document, it cannot be used to distinguish a rare document from the rest. For example, if a term appears only in very few documents, say two documents comprising such a term out of a thousand documents, then that term is an important criterion to distinguish those two documents as relevant to that term. In this case, the similarity is proportional to the ratio of the total number of documents (d) to the documents count (say df_j, the document frequency) which have the occurrence of this rare term. The ratio is called *inverse document frequency*, idf.

The important terms encountered in the above discussion are defined and listed below as

t : *Terms*. It is the number of distinct terms (keywords) in the entire collection of documents.
tf_{ij} : *Term frequency*. It is the number of occurrences of the term t_i in the document D_j.
df_j : *Document frequency*. It is the number of documents that contain the term.
idf_j : *Inverse document frequency*. It is equal to $\log \frac{d}{df_j}$, where d is the total number of documents.

Each document is represented as a vector, with a total t number of components, and each entry in the vector corresponds to a distinct term from the entire collection set. In addition, each component in a vector is filled with *weights* computed for the corresponding term. This weight is automatically assigned based on how frequently the term has occurred in the *entire documents collection*, and another weight is based on how often the term has appeared in the *particular document*. The weight of a term in a document increases, the more often it appears in that document, and decreases, the more often it appears in other documents.

The weight of a term in a document is defined as a combination of *term frequency* tf, and the inverse document frequency idf. Each term has a position in the vector,

if it is present in the document, that position is marked with its weight, else marked as weight 0. To compute the weight (dw_{ij}) of a term t_i in a document D_j, the following equation of $tf.idf$ is used:

$$dw_{ij} = tf_{ij} \times idf_j. \tag{18.2}$$

When an information retrieval system (actually it retrieves the document carrying the needed information) is used to query a document collection using a query of n number of terms, the system computes one *document vector*, $\langle dw_{1j}, dw_{2j}, \ldots, dw_{nj} \rangle$ of size n for each of the documents j, and these vectors' components are filled with weights as discussed above. Similarly, a *query vector*, $\langle qw_1, qw_2, \ldots, qw_n \rangle$ is computed for the terms found in the query. Note that, size of the query vector is also n. The similarity measure between query Q and a document D_j is defined as the dot product of two vectors.

$$sim(Q, D_j) = \sum_{i=1}^{n} qw_i \times dw_{ij}. \tag{18.3}$$

18.5 Indexing

The vector space model and other retrieval strategies make use of an inverted index file structure to avoid the length of search in the keywords of every document for which relevancy is to be established. Instead of searching into a document, an inverted index is generated in advance for all the keywords in the document set. An entry for each of the n terms (t_1, ..., t_n) is stored in an index structure, like the one shown in Fig. 18.5. For each term t_i, a pointer references to a linked list, which contains an entry for each document containing this term as well as the term frequency in that document is present. For example in row i, there are entries for (d_j, tf_{ij}) forming a connected list for term t_i. The d_j is a document in which term frequency is tf_{ij} for the term t_i. The figure shows that the term t_i has a frequency of 7 in document d_3. This indexing structure has the advantage that the retrieval system can search the term quickly, as well as the documents in which the term appears [18].

18.5.1 Index Construction

A key challenge in the construction of an index is the size of the data involved in the index itself (see Fig. 18.5), which is a dynamic data structure typically used for cross-reference generation, but cannot be kept together in the memory of a typical system due to its size. The task to be performed here is a *matrix transposition*, given that the documents' terms' matrix is very sparse. Such matrices are not so easy to manipulate directly as an array. Therefore, the index construction makes use

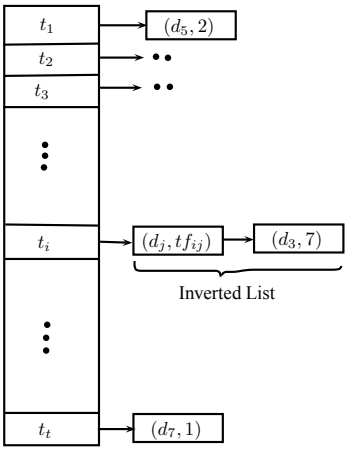

Fig. 18.5 Inverted index file

of index compression techniques, together with distribute-comparison-based sorting techniques.

Algorithm 18.1 is a simple in-memory inversion algorithm. The key idea in this algorithm is that the first pass through the documents to be indexed collects terms frequency (tf_{ij}) information, which is sufficient for the construction of an inverted index. The index is stored in the memory in the form of a template. The second pass places the pointers, shown by arrows, at their correct positions in the template as shown in Fig. 18.5.

Algorithm 18.1 To build an inverted index using the *in-memory technique*

1: **Pass I**: Make an initial pass over the collection of documents.
2: For each term t_i, count its term frequency tf_{ij} in each document d_j, and determine the upper bound u_{t_i} in bytes, on the length of the inverted list for t_i.
3: Allocate an in-memory array of $\sum_{t_i} u_{t_i}$ bytes, and, for each term t_i, create a pointer c_{t_i} to the start of a corresponding block of u_{t_i} bytes.
4: **Pass II**. Process the collection of documents a second time.
5: For each document d_j, and for each term $t_i \in d_j$, append a code representing $\langle d_j, tf_{ij} \rangle$ at c_{t_i}, and update c_{t_i}.
6: **Pass over in memory Index**: Make sequential pass over these index, for each term t_i, copy the tf_{ij} representations of the $\langle d_j, tf_{ij} \rangle$ pointers from the allocated u_{t_i} bytes to the inverted file, and compress if required.

It is possible to extend the in-memory technique to *data collection* technique. In this technique, the index size may exceed the memory size. This is possible by laying off the index skeleton on the disk, while partial sequences of indexes are created in the memory, and each one is transferred to the disk in a skip-sequential manner into a template in a large file. Using this extended method, and using compression, it is possible to create indexes of the size of terabytes using a memory of a moderate size of 4–8 GB—a size common in the present time.

Parallel Processing of Index

For parallel processing of indexes, they can be constructed in parallel and can be merged after regular intervals (see Algorithm 18.2). The merge-based inversion technique reads the documents and indexes them in the memory until a fixed capacity is achieved. Every inverted list is represented using a structure, which can grow as more information about the index terms become available. For this structure, dynamic resizable arrays are most appropriate. As soon as the memory is full to its predefined capacity, the indexes are flushed out to disk in a single run, such that the inverted lists are stored in the disk file in a lexicographic order. This lexicographic order later becomes useful for the sequential merging of the indexes. Since the runs of these subindexes are never used to answer a query, there is no need to store their vocabulary in an explicit structure, hence each run can be written at the head of its inverted list. Once a run is written into the hard disk, it is fully deleted from the memory so that the construction of the next run begins with initially empty vocabulary.

Algorithm 18.2 To build an inverted index using *Merge-based technique*

1: **while** all the documents are not processed **do**
2: Initialize an in-memory index, using a dynamic structure for the vocabularies and a static coding scheme for inverted lists; store lists either in dynamically resized array or in linked blocks.
3: Read documents and insert $\langle d, tf \rangle$ pointers into the in-memory index, continuing until all allocated memory is consumed.
4: Flush this temporary index to disk, including its vocabulary.
5: **end while**
6: Merge all the set of partial indexes to form a single index, compressing the inverted lists if required.

When all the documents have been processed, the runs available in the disk are merged to get the final index. The merging process builds the final vocabulary on the fly. Since the runs from the disk are read (into a buffer) for merging, a sufficiently large size of the buffer will reduce the disk accesses, hence making the process further faster. However, if the free disk space is limited, the final index can be written back into the RAM at the space occupied by the runs, progressively as they are processed, which is helpful in representing the final inverted lists more efficiently. The latter becomes possible because the final index is typically smaller than the runs, hence the vocabulary information is not duplicated.

The index construction using the merge-based approach is common and practical for data collections of all sizes. It is scalable, and operates efficiently in a memory size as small as 100 MB. The overheads of a disk space can be limited to a small fraction of the final index, as it requires only one parsing pass over the data, and the method can be extended from keyword indexing to phrase indexing.

18.5.2 Index Maintenance

Inserting a new document into the text collections amounts to inserting a few bytes to the end of every inverted list depending on how much the terms in that document are, i.e., size of the document. Since a document may consist of 100s–1000s of terms, such insertion of terms requires fetching and extending 100s of inverted lists, and it may require 10–20 s in the worst cases, to rebuild the inverted list for the new collection of documents. However, the merge-based inversion approach can index thousands of documents per second, making this method almost 10,000-times faster [12].

For fast insertion of terms for indexing into the inverted list, the disk resident part of the list be not accessed frequently, else it will reduce the overall speed. The practical solution is to amortize the update cost over a sequence of insertions. There are many properties of text databases, which allow strategies for cost amortization. The new documents are not immediately available for searching, if they are searchable they can be made available through a temporary in-memory index—that is, the last subindex in the merging.

There are three broad categories available for updating the index as the new documents get added into the collection, they are *rebuild* from scratch, *merge* an existing index with an index of new documents, and *incremental update*.

Rebuild

There are applications where the index is not updated at all, but it is rebuilt from scratch at some regular intervals. The presence of new documents is established through crawling, and an update is not immediately needed for these documents. This approach is considered economical even for gigabytes of data, where rebuilding takes just a few minutes, rather than updating the existing index, which anyway will require fetching the document as well as the index to be updated.

Intermittent Merge

Number of inverted lists can be maintained for the documents collection, in the memory of a system. Having the lists in memory, it is easy (less complex) to insert new documents into these lists when the new documents are discovered. This is carried out using Algorithm 18.2 discussed on page 567, through the process of merging the indexes. Alternatively, the existing documents' index remains in a standard inverted file, and new documents are indexed as an inverted file data structure in the memory. With this, the two indexes can share a common vocabulary. In this process, both the indexes are made available to queries, and the result of any query is the union of the query results from both of these inverted indexes. When the size of the in-memory index crosses a threshold size, it is merged into the index file on the disk.

Other approaches for indexing use *incremental update*, or *choice of alternative strategies* for indexing as a combination, suffix arrays, wavelet trees, Bayesian inferences, predicate-based indexing, and probabilistic indexing [14].

18.6 Probabilistic Retrieval Model

The probabilistic model of retrieval computes the *similarity measure* ($sim(q, d_i)$) between the query $q \in Q$ and a document $d_i \in D$ as the probability that d_i is relevant to q, where D is the set of documents in the collection, and Q is the set of queries. The probabilistic retrieval method estimates the term weight based on how often the term appears in the *relevant* but does not appear in the *non-relevant* documents. The term weight is calculated using the *probability ranking principle*, which is based on the assumption that optimal performance is observed when documents are ranked on their relevance to the query. In the approach used, probabilities are first assigned to components of the query and then each of these is used as evidence in computing the probability that a given document is relevant to the query [8].

Each term in the query is also assigned a weight corresponding to the probability that the document term matched with the query will retrieve a relevant document. The weights of the query terms are aggregated to obtain the final measure of relevance. A probability-based information retrieval system ranks the documents in decreasing order of probability of relevance to the user's information needs. Following are essential preconditions for this probabilistic retrieval model:

- Retrieval accuracy is dependent on how accurately the query and document have been represented, and does not directly depend on the documents and the queries,
- The representation of documents and queries may not be accurate due to a variety of uncertainties prevailing in the method of representation itself.

Figure 18.6 shows the conceptual probabilistic model for IR, where event space is represented by $Q \times D$, such that $Q = \{q_1, q_2, q_3, \ldots\}$ is a set of queries representing the information need, and $D = \{d_1, d_2, d_3, \ldots, \}$ is a finite set of documents. Each query q_i and document d_j is in the form of *descriptors*, where q_i and d_j are set of terms (i.e., keywords). The descriptor is a binary-valued vector, and each element in that corresponds to a term. Every query is taken as a unique event, i.e., two identical queries at different times are treated as different events. We assume that G is a set of possible *relevance judgments* for the Cartesian product of documents' set D and queries set Q. Let the relevance relationship be r, between the query set and document set, in the form of a mapping $r : Q \times D \rightarrow G$. In case of Boolean IR, a document is either relevant to a query or not, hence for any query q_i and document d_j, there is $r(q_i, d_j) \rightarrow \{0, 1\}$.

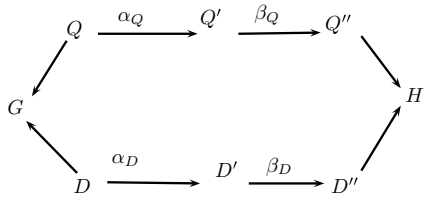

Fig. 18.6 Conceptual probabilistic model of IR

In fact, an IR system does not directly handle the documents and the queries, but handles them indirectly, in the forms of their representations. For example, a document is represented in the form of *index terms*, and a query is represented as a Boolean expression comprising terms and Boolean operators (see page 562 for more detail). Let us assume that Q' and D' are representations of queries and documents, respectively. These representations have a mapping from the original query and documents through some functions, which are expressed as $\alpha_Q : Q \rightarrow Q'$ and $\alpha_D : D \rightarrow D'$. Hence, if there are two different documents, but represented with the identical set of index terms, then they will be mapped onto the identical representation.

Further mapping is introduced from the representation (Q', and D') to *object descriptions*, to make the models more general. This is done by supplementing a *weight* to the index term forms of the queries' and documents' representations. The weight is a real number. Let us assume that these object descriptions, for query set and document set, respectively, are Q'' and D'', and the mappings as $\beta_Q : Q' \rightarrow Q''$ and $\beta_D : D' \rightarrow D''$, respectively. Due to the introduction of weight the new mapping shows a more accurate relevance relation between the query and its descriptor set, and similar is the case for the document set to their descriptors. Therefore, the more correct value of the relevance function r is $r : Q'' \times D'' \rightarrow H$.

For a submitted query $q_i \in Q$ to the IR system, the documents $d_j \in D$ are ranked according to the decreasing order of $r(q_i'', d_j'')$, such that the document with the highest rank (of relevance) is at the top. The job of an IR system that ranks the documents in the order of their relevancy for a query q_i'' is to calculate relevance and rank every document $d_j \in D''$. Often, for the sake of simplicity, the description and representation are treated the same, and both are represented as the form of set of terms [4].

18.7 Fuzzy Logic-Based IR

The fuzzy retrieval technique is based on the fuzzy set theory and fuzzy logic—an extension of the classical set theory. This fuzzy retrieval technique is based on the concept that the word matching between the inquiry word set and the text word set should not be limited to the perfect matching with the stem words. But, since the words in inquiry also match with their synonyms in the text documents, the matching should be graded, depending on the degree or level of matching in the range from 0 to 1. The extremes of this range, a special case in fuzzy logic, corresponds to the Boolean matching. Fuzzy logic is more realistic than the Boolean logic, simply due to the fact that it considers the exact as well as vague matching, the latter being more frequently encountered in the real world [7].

In the relations over fuzzy sets, the elements of two sets have a degree of association as a form of relation rather than simply—related (binary 1) or not related (binary 0). The degree of association ranges from 0 to 1, where 0 indicates the total

absence of relation and a 1 indicates the total presence of the relation; therefore, a fuzzy relation between query keywords set and of document's keywords is

$$R : X \times Y \in [0, 1] \tag{18.4}$$

and the range of $R(x, y)$ varies from 0 to 1 depending on how close the y, $(y \in Y)$ is associated with x, $(x \in X)$.

Information Retrieval Using Fuzzy Sets

The membership value $R(x_i, y_j)$ specifies, for each $x_i \in X$, $y_j \in Y$, the grade of relevance of index term x_i with the document y_j. The grade of relevance depends on many factors [3]:

1. position of term y_j in the text document, if the document is a research article, and y_j appears in the list of keywords, the abstract, or in the conclusion part, the relevance is higher;
2. frequency of occurrence of y_j in the document;
3. x_i and y_j are terms formed from the same basic stem word; and
4. y_j is synonym of x_i—the proximity of the meaning of x_i and y_j decide value of $R(x_i, y_j)$'s closeness with 1.

The criteria (1) and (2) above are user-defined and they can be programmed in the implementation according to the user needs. The stem word criteria (3) requires a data structure similar to the one shown below in Fig. 18.7, which helps to locate the stem word for a given word, and then the stem word is substituted in the original text before the retrieval technique is applied to it [4].

Other, more often used approach for stemming is through some stemming algorithm, e.g., Porter's stemmer, which takes the benefit of certain patterns in the words to obtain the stem word. For example, for the words with "ing" at the end, the stem word can be obtained by removing the "ing" part, like "book" from "booking". Also, there are other features, like removing "ed" at the end of the past tense of a verb; we obtain its stem word, say "book" and "look" from "booked" and "looked", respectively. Along with this, there are some more complex patterns, like "goose" from "geese", etc.

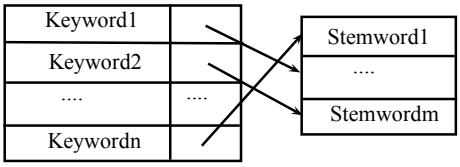

Fig. 18.7 Data Structure for finding stem words

Another important relation for IR based on criteria (iv) above is the fuzzy thesaurus, which plays a pivotal role for FIR. The fuzzy thesaurus shows the relationship between the pairs of words based on their centrality or degree of relevance. The structure of a *fuzzy thesaurus* (T) is

$$\langle WC1\rangle\langle WC2\rangle\langle RD\rangle$$

where WC stands for *word category* and RD is the degree of relationship between the words $WC1$ and $WC2$. A typical examples for this can be as follows.

attraction,	love,	0.8
studious,	hardworking,	0.9
war,	crime,	0.7.

A relation $\langle x_i\rangle, \langle x_j\rangle, \langle 1.0\rangle$ shows that x_i is a perfect synonym of x_j. The fuzzy thesaurus can be manually constructed, or can be generated from the lexicons. *Transitivity relationship* can be applied by computing the missing relationship degrees from the existing ones. The thesaurus, say T, is a *reflexive fuzzy relation*, defined over X^2. For each pair of index terms $(x_i, x_j) \in X^2$, the $T(x_i, x_j)$ expresses the degree of association of x_j with x_i, such that the degree to which the meaning of the index term x_j is compatible with the meaning of the index term x_i. The objective of this relation is to deal with the problem of *synonyms* among the index terms, for example a document's term is a synonym of query term or vice versa. The relation helps to identify the relevant documents which otherwise would not be selected in the absence of a perfect match between the keywords in the user inquiry and those in the text document.

Different approaches can be used for the construction of fuzzy thesaurus. For example, experts in the domain of text can be asked to identify, in a given set of index terms, the pairs of words whose meaning they consider are associated, and provide the degree of association for each pair. In Fuzzy Information Retrieval (FIR) an inquiry can be expressed in the form of a fuzzy set (say Q) based on the index term X. Then, by composing Q with the fuzzy thesaurus T, we obtain a new fuzzy set on X, say A—which represents the *augmented inquiry*, i.e.,

$$A = Q \circ T, \tag{18.5}$$

where "$\circ$" is called *max-min* composition operator, such that

$$A(x_j) = max\text{-}min[Q(x_i), T(x_i, x_j)]. \tag{18.6}$$

Here, $x_i \in X$, for all $x_j \in X$. The retrieved documents, expressed by a fuzzy set F defined over Y, are then obtained by composing the augmented inquiry, expressed by the fuzzy set A, with the relevance relation R, i.e.,

$$F = A * R \tag{18.7}$$

where "$*$" is a matching operator, which evaluates the degree of fuzzy matching by multiplying the fuzzy membership of the augmented inquiry with the fuzzy membership of the corresponding words in the text. Finally, the relevance measure of the text with the inquiry under consideration is obtained by summing all the values of fuzzy matching for the text.

Example 18.2 Fuzzy logic-based Information Retrieval.

Let the terms be $x_i, i = 1, 6$, representing keywords—"Einstein", "scientific", "Theory of Relativity", "Bohr", "subatomic", and "New idea", respectively. Let the given inquiry be $Q =$ *"Einstein's Scientific Theory of Relativity"*, and the vector representation of corresponding fuzzy inquiry be

$$\begin{matrix} & x_1 & x_2 & x_3 \end{matrix}$$
$$Q = [1 \;\; 0.6 \;\; 0.8] \tag{18.8}$$

where 1, 0.6, and 0.8 are called the *centralities* of x_1 (Einstein's), x_2 (Scientific), x_3 (Theory of Relativity), respectively. The centrality indicates the presence of certain qualities, whose computations are modeled as a computation of fuzzy membership degree. The relevant part of the fuzzy thesaurus T, restricted to the support of Q, is given by the matrix:

$$T = \begin{bmatrix} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ x_1 & 1 & .6 & .9 & 0 & .1 & 0 \\ x_2 & .6 & 1 & .8 & .6 & .5 & .8 \\ x_3 & .9 & .8 & 1 & 0 & 0 & .7 \end{bmatrix}. \tag{18.9}$$

Note that terms' pairs (x_1, x_1), (x_2, x_2), and (x_3, x_3) has each a fuzzy matching of 1. On using (18.5), and the data given by Eqs. (18.8) and (18.9), we get

$$A = \begin{bmatrix} 1 & .8 & .9 & .6 & .5 & .7 \end{bmatrix}. \tag{18.10}$$

The values in augmented query A are obtained as follows: we show the computation for third element (A_3), i.e., 0.9 as

$$\begin{aligned} A_3 =& max(min(1 \times 0.9), min(0.6 \times 0.8), min(0.8 \times 1)) \\ =& max(0.9, 0.6, 0.8) \\ =& 0.9. \end{aligned}$$

Assume that relevance relation R (i.,e., $R(x_i, y_j)$) is given by the matrix,

$$R = \begin{bmatrix} & y_1 & y_2 \\ x_1 & .7 & .3 \\ x_2 & .3 & .1 \\ x_3 & .6 & .1 \\ x_4 & .6 & 0 \\ x_5 & .6 & .3 \\ x_6 & .6 & .1 \end{bmatrix}, \tag{18.11}$$

where y_1, y_2 are the documents related to index terms x_i, $i = 1, \ldots, 6$. Using Eq. (18.7) and data given by (18.10) and (18.11), we get fuzzy relevance relation F as

$$F = A * R = \begin{bmatrix} 2.56 & 0.69 \end{bmatrix}. \tag{18.12}$$

Equation (18.12) shows that the relative relevance of y_1 with respect to y_2 is 2.56/0.69, i.e., 3.7. Thus, the result appears to be more realistic in comparison to the result obtained using crisp (binary) logic. This fact is also supported by the contents of texts y_1 and y_2. Once the FIR system lists the documents with their relevance values, the user can now decide whether to inspect all the retrieved documents supported by the fuzzy set F or to inspect only some of the documents depending on the degree of association of the document with the index terms. □

The use of fuzzy set theory for IR shows that the fuzzy relevance relation and fuzzy thesaurus are more expressive than their crisp set counterparts. Also, since the degree of association is returned along with the retrieved documents, it helps the user to decide the order in which the documents can be viewed, particularly when the documents are in large number.

The FIR promises a higher potential for cross-language text processing and IR. Every language and its semantics have a close association with the culture in which it has its roots, and therefore, exact matching terms for any language are not possible in other languages. In fact a degree of relevance or, only fuzzy relation exists between the matching words of the two or more languages. We have planned to work on the cross-language areas, which include English, and Sanskrit-based Indian languages [3].

18.8 Concept-Based IR

The keyword-based approach we discussed above is also called the BOW (Bag-of-Words) approach. The concept-based information retrieval makes use of *semantic concepts* for the representation of the documents and queries, instead of (or in addition to) keywords; and the approach performs the retrieval in the concept space of query and documents. Hence, this retrieval model is less dependent on the specific terms (keywords) used, and yields a match even when the same notion is described by a different set of terms in the query or in the target document, or in both. This helps in eliminating the problem of *synonymy*. Since more relevant documents are likely

to be retrieved from the store, the *recall* rate increases. In the similar way, if the concepts chosen for the words, particularly the ambiguous words in the query and document are accurate, the non-relevant documents that were retrieved with the BOW approach could be eliminated from the results. Since the non-relevant documents are lesser in the retrieved documents now, it increases the *precision* rate. Note that in the keyword-based approach of IR, non-relevant documents' share increases in the retrieved documents when the keywords have multiple meanings, e.g., "bank" (of river and of money). The problem of many senses of a word is called *polysemy*.

The concept-based methods can be characterized by the following three parameters:

1. *Concept representation.* The concept-based IR makes use of real-life concepts that closely resemble human perception. The concepts are based on the language of the text of documents and queries.
2. *Mapping method used.* It is the method used for mapping the natural language text to concepts. In fact, the ideal mechanism is the manual approach, which builds a hand-crafted ontology of concepts along with a list of words to be assigned to each concept. However, this approach is inefficient and time consuming. The automatic mapping between concepts and words can also be done through machine learning, but with loss of accuracy.
3. *Use in IR.* The concepts are used during the indexing and retrieval phases of the documents. A simpler but less-accurate method applies the concept analysis in one stage only. For example, it is better to apply the concept analysis in the query expansion rather than document retrieval.

A large and diverse knowledge repository, like Wikipedia, can create a powerful concept ontology, well-suited for concept-based IR. A wide range and exhaustive coverage of topics in Wikipedia's diversity, coupled with automatic ontology-building capability, can be used for the purpose of highly fine-grained ontology. Additionally, the inverted index language provides a mapping from massive natural language text *terms* of the entire Wikipedia collections, to the concepts as per the context and sense of the text terms. This entire process produces a powerful classifier that automatically maps any given text fragment to its concept ontology [9].

18.8.1 Concept-Based Indexing

A concept-based IR algorithm maps documents and queries individually to the Wikipedia concept space. The indexing and retrieval are performed in this space only. In the Wikipedia-based Semantic Analysis (SA), the semantics of a word are described by a vector that stores the word's association strengths to Wikipedia-based concepts. A concept is in the form of a *concept vector* $\overrightarrow{F_d}$, which is generated from a single Wikipedia article d, as a vector of words appearing in that article. The article is weighted by $tf.idf$ score. Once these concept vectors are generated, an inverted index is created for the purpose of mapping back each word to the concepts it is

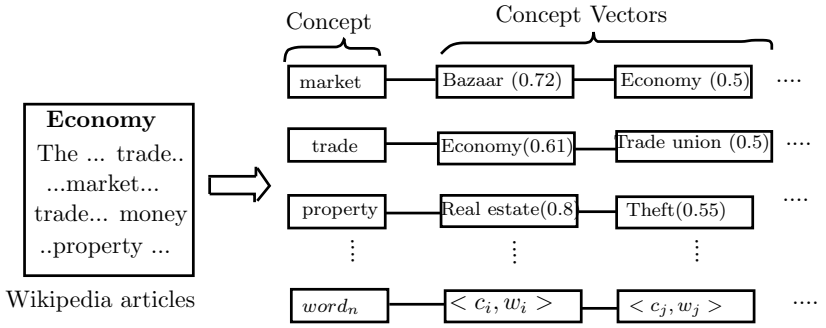

Fig. 18.8 Semantic analysis from Wikipedia articles

associated with. Thus, each word appearing in the Wikipedia corpus can be seen as triggering one or more concepts c_j it points to the inverted index. An attached weight w_j with the concept represents the degree of association between that word and the concept. This process is illustrated in Fig. 18.8, which shows how the semantic analysis is carried out of a Wikipedia article.[4] The articles and the words in the articles are processed to build a weighted inverted index, which represents each word of article as a vector in the space of all Wikipedia concepts, i.e., articles in the Wikipedia itself.

In the *concept-based indexing*, each document (i.e., article) in the corpus is mapped to a vector of weighted concepts, represented by $\langle c_1, w_1 \rangle$, $\langle c_2, w_2 \rangle$, ..., $\langle c_n, w_n \rangle$. Like Bag-of-Words (BOWs) vectors, the concept-based vectors are also sparse, hence concept weights are zero for a large majority of the Wikipedia concepts. Since every word in the document to be indexed may be related to a large number of concepts, and a document containing a collection of words is likely to be related to an even larger number of words, indexing an entire list of related concepts for every document is not feasible. Therefore, only the concepts having the highest weights (for example, "Bazaar", "Economy", etc., in Fig. 18.8) are used for indexing. In a sorted representation of the weighted vector, this subset of concepts is simply its prefix.

It more difficult to map the long documents in full, into the concept space. For example, a small part of a long document might be relevant to a given query, but the semantics of this small part are not likely to be fully represented in the concepts vector of the complete document. Note that, a somewhat similar problem also exists in the Bag-of-Words approach, where the term frequency (tf) value is normalized to account for the documents of varying lengths.

Due to the averaging effect of the representation of longer text fragments, and due to the practical limitation to use only a small subset of the representation of concepts, the challenge in the concept-based retrieval technique is even greater. Because the concepts generated for a subset of the larger document, where the subset is relevant

[4] A Wikipedia article is an article about some topic, for example, we find articles in the collections of Wikipedia, like websites, Internet, WWW, etc.

and the remaining document consists of non-relevant topics, the representation of the latter need to be pruned out of the index vector. This is necessary as otherwise the concepts weights in the overall document concepts vector might be too low to show any significance of the retrieval results.

Semantic Analysis-Based Indexing Algorithm

An algorithm suitable for indexing larger size documents, based on semantic analysis and using the inverted index, is given as Algorithm 18.3. This algorithm indexes a corpus D of documents using semantic analysis concepts, by trimming of semantic analysis vector to s as the first concepts. Each document $d \in D$ is represented by a concept vector $\overrightarrow{F_d}$. For each concept $\langle c_i, w_i \rangle \in \overrightarrow{F_d}$, the corresponding $\langle d, w_i \rangle$ is added into the inverted inverted index. Here, c_i is the concept and w_i is the concept weight.

To overcome the problem of mapping each large document $d \in D$ in full into concept space, d is divided into smaller fixed length (size l) overlapping passages set P_d. Then each passage $p \in P_d$ is represented separately by its own generated set of concepts $\overrightarrow{P_d}$.

Each passage p is indexed and can be retrieved as a stand-alone unit of information. For this, all concepts $\langle c_i, w_i \rangle \in \overrightarrow{F_p}$ corresponding to a passage p, the passage is ranked separately as an independent unit along with its relevance in its parent document (d), shown in the algorithm by "$add\langle p, w_i \rangle$ to $InvIndex[c_i]$". These concepts are indexed in a standard IR inverted index, with each concept having a unique identifier in the form of a token. The score w_i, associated with each concept c_i in the vector, is used as the token weight, which is equivalent to term frequency tf, in standard text indexing.

Algorithm 18.3 Semantic analysis-based indexing in an inverted index

```
1: Procedure SA-Indexing(D, s, l) {Indexes corpus D using SA concepts; trims SA vector to s as
   first concept; then segments document to passages, each of length l.}
2: for all d ∈ D do
3:    F_d ← SA(d, s)
4:    for all ⟨c_i, w_i⟩ ∈ F_d do
5:       add⟨d, w_i⟩ to InvIndex[c_i]
6:    end for
7:    P_d ← Divide-Into-Passages(d, l)
8:    for all p ∈ P_d do
9:       F_p ← SA(p, s)
10:      for all ⟨c_i, w_i⟩ ∈ F_p do
11:         add⟨p, w_i⟩ to InvIndex[c_i]
12:      end for
13:   end for
14: end for
```

18.8.2 Retrieval Algorithms

Once a query is received by an IR system, it is converted into a concept vector by the retrieval algorithm. The representation method used is identical to the one we discussed above, for documents and passages during the indexing. The indexes of full documents and passages, as evidences, are to be combined for ranking. This combination is performed by retrieving both set of results, and then by summing each document's full score with the score of the best performing passage in it. In the next phase, documents are sorted on a combined score, i.e., sum, and the top-scoring documents are output. All the above steps are described in the retrieval algorithm 18.4. The algorithm works as follows:

1. Retrieve the query results, which are based on SA concept, for query $\overrightarrow{q}$, as well as the cutoff concept vector at s;
2. Retrieve results for query $\overrightarrow{q}$ using the combined results;
3. Score the document's match to the query using the standard inverted index function *InvIndex-score()*.

Algorithm 18.4 SA-based Retrieval

Procedure SA-Retrieval($\overrightarrow{q}$, s) {Retrieve the SA concept-based results for query $\overrightarrow{q}$, and cutoff concept vector at s}
$\overrightarrow{F_q} \leftarrow SA(\overrightarrow{q}, s)$
return DocsPass-Retrieve($\overrightarrow{F_q}$)
Procedure $DocsPass\text{-}Retrieve(\overrightarrow{q})$ {Retrieve results for query $\overrightarrow{q}$ from the combined results; Score the document's match to the query using the standard inverted index function $InvIndex\text{-}score()$.}
for all $d \in D$ **do**
  $W_d \leftarrow InvIndex\text{-}Score(\overrightarrow{q}, d)$
  for all $p \in PASSAGES(d)$ **do**
    $W_p \leftarrow InvIndex\text{-}Score(\overrightarrow{q}, p)$
  end for
  $W'_d \leftarrow W_d + max\ W_p$
end for
return ranked list according to W'_d

The retrieval algorithm 18.4 has a single parameter s that controls the query concept vector's cutoff. The value of s can be chosen the same as that during the indexing, however it is not necessary. If the entire corpus is indexed with large cutoff values, the resultant costs for computation as well as storage will be high, hence it is not advisable to index the entire corpus with large cutoffs. Since the query is much smaller in size, there are no such costs as that for text corpus, hence a finer representation will be beneficial. That can be achieved using a higher value of s.

18.9 Automatic Query Expansion in IR

Most IR systems, and particularly the web search engines, have a standard user interface comprising of an input box to accept from a user, a query in the form of keywords. The submitted keywords are matched against the collection of index terms to find the documents that contain those keywords. The results are then sorted by various methods. When there are many topic-specific keywords in the user's query, which accurately describe user's information need, the system is likely to return good matches for the query. However, when this query is short—comprising 2–3 words, as the case usually is—there is likely ambiguity in the language of the query, then this simple retrieval model is sensitive to errors [1].

The most serious issue in retrieval effectiveness is the *term mismatch* problem, i.e., indexers and the users do often not use the same keywords. For example, a document uses "tv" while user submits the query with keyword "television". This is known as the *vocabulary problem*. This problem gets compounded due to *polysemy*, i.e., the same word with different/multiple meanings, such as Java (name of language, and also name of a place), and also due to *synonymy*, i.e., different words with the identical or similar meanings, such as "tv" and "television". Synonym words, along with word *inflections* (like in plural forms, "book" versus "books") may result in a failure to retrieve relevant documents. This may decrease the *recall*. The problem of polysemy may cause retrieval of erroneous or non-relevant documents, thus causing a decrease in *precision*.

Several approaches have been proposed, to deal with the vocabulary problem; some of them are the following:

Interactive query refinement,
Relevance feedback,
Word Sense Disambiguation,
Query expansion, and
Search results clustering.

The technique of query expansion is one of the most natural and successful techniques, using which the original query is expanded with other words that best capture the user's actual intent, or it simply produces a more useful query—a query that is more likely to retrieve relevant documents.

As the use of search engines increased over time, the size/length of the user's query has also increased. In the year 2009, an average query's length was 2.3 words. However, there has been an increase in the number of long queries per user or per session of interaction, of five or more words; the most common queries are still those of one, two, and three words. When a query is short, the vocabulary problem is more serious, because the shortage of query terms limits the scope of handling synonymy. At the same time, the reduced resizing of data makes the effects of polysemy more severe. This required the need and scope of Automatic Query Expansion (AQE). Over the years a number of AQE techniques have increased that employ sophisticated methods for finding new features related to query terms. Today,

there are firm theoretical foundations, and a better understanding of the utility and limitations of AQEs. For example, what the critical parameters are that affect the performance of the IR system, what types of AQEs are useful and what not, etc.

Along with the AQE, the basic techniques are being increasingly used in conjunction with mechanisms to increase their effectiveness, like *method combinations*, selection of information source dynamically for expansion, and discriminating policies for the application of methods. These advances have been supported by many experimental findings. The AQE methods have gained their popularity due to the evaluation results obtained at the *Text REtrieval Conference* series (TREC).

Document Ranking using AQE

The IR systems including search engines depend on computing the importance of terms occurring in the query and documents to determine the relevance of document(s) to the queries. The commonly used indicator of relevance is the similarity measure between the query and the document. Considering that a query is represented by q and a document by d, the similarity measure between them, $sim(q, d)$, is expressed by

$$sim(q, d) = \sum_{t \in (q \cap d)} w_{t,q} \times w_{t,d} \tag{18.13}$$

where $w_{t,q}$ and $w_{t,d}$ are the weights of term t in a query q and a document d, respectively, according to some weighting criteria adopted. The weight of a term t in a document is typically proportional to the term frequency (tf) in that document and to the inverse document frequency (idf). The purpose of idf is to diminish the effect of very frequent terms like "the" for which we are not interested to find the relevance. But we want to increase the effect of terms, which occur rarely, in a few documents only, for example, "blue" and "bird", which might be occurring only in a few documents. If N is the total number of documents and n is the number of documents in which the rare term occurs, the idf for this term is $\log(\frac{N}{n})$. Note that if a term like "the" occurs in every document, then idf is zero, hence it will not contribute to the similarity measure.

The similarity computation between a document and query representation, expressed in Eq. 18.13, can be easily modified by abstracting away from the original underlying weighting model, so as to work for the query expansion. In this revised model, the basic input to AQE module is 1. original query q and, 2. source of data from which to compute the weight and the expansion terms. The output of the AQE module is the query formed (say, q') due to the expansion of query terms, and their associated weights w'. These new weighted query terms $\langle q', w' \rangle$ are used for computing the similarity between the query q' and the original document d. The new similarity computation formula is expressed by

$$sim(q', d) = \sum_{t \in (q' \cap d)} w'_{t,q'} \times w_{t,d}. \tag{18.14}$$

The commonly used data source for generating new query terms is the collection itself into which we are searching, and sometimes it is the thesaurus to get the synonyms of the keywords. The simplest approach to weight the query expansion terms is to use the same weighting function that is being used by the ranking system. When more complex features in the terms, like phrases, are used for query expansion, the underlying system must be able to handle such features. An example is "Jodhpur departmental store", which though a phrase, can be treated as a single term, for the purpose of indexing.

In the following we address some new areas, in addition to document ranking, where AQE is used heavily; these are: question answering, multimedia IR, information filtering, and cross-language IR. This introduction is followed by pointers to more recent applications [1].

Question Answering

The goal of question answering (QA) is to provide concise responses (instead of full-length documents) to certain types of natural language questions, such as "Who built the Taj Mahal?" Like the Document Ranking (DR), the QA is faced with a fundamental challenge of mismatch between the vocabularies of the question and answer.

To improve on the early stage of QA, which is document retrieval, a common strategy used is to expand the original question terms with terms that are expected to appear in documents containing answers to this question. One important goal of QA translation is to learn the associations between question words and the answer words, which may be synonyms to the words in question. For example, for the question, "Where Taj Mahal is located?," the answer part embedded in a document may be "Taj Mahal is in Agra," or "Taj Mahal is situated in the city of Agra." Here, the words "situated" and "exists" are the synonyms of the word "located". Different resources for AQE for QA include using lexical ontologies like WordNet—a lexical database of synonyms (called *synsets*), and semantic parsing of questions based on roles, and other criterias [7].

Multimedia Information Retrieval

With the widespread use of digital media and digital libraries, the requirements of searching the multimedia documents like image, speech, and video have become important. Up till recently, the multimedia IR systems performed only text-based search over the media *metadata*, like annotations and captions, which are surrounding the *html/xml* descriptions. However, when these metadata are absent, this method is not suitable. Hence, the IR relies on some form of multimedia content analysis, which is implemented in combination with the AQE techniques. For example, a transcription is produced by an automatic speech recognition system for *spoken document retrieval* systems, and this is augmented with related terms, in advance to raising the query. Since automatic speech transcriptions often contain mistakes, this form of a document expansion is very useful for spoken document retrieval.

For image retrieval systems, a typical approach is making use of query examples having visual features, like colors, textures, and shapes. The query is iteratively refined through a relevance feedback.

For video retrieval systems, both the documents and queries are mostly *multimodal*, i.e., they have both textual as well as visual aspects. An expanded text query is usually compared against the textual description of the visual concepts, and the matched concepts are used for visual refinement.

Information Filtering

Information filtering (IF) is different from IR. It removes redundant or unwanted information from an information stream prior to presentation to a human user. Its main goal is the management of the information overload and increment of the semantic signal-to-noise ratio. The documents arrive continuously and the user's information needs evolve over time, as per the experience of the user. Some examples of information filtering are electronic news, blogs, e-commerce, and e-mails. There are two main approaches to IF: 1. *collaborative IF*, which is based on the preferences of like-minded users, and the other is 2. *content-based IF*. However, these techniques are said to bear a strong conceptual similarity to IR, because the user profile can be modeled as a query and the data stream can be treated as a set of collection of documents. The user profiles (i.e., queries) are learned using relevance feedback techniques, or other forms of query expansion, such as those based on similar users.

Cross-Language Information Retrieval

The Cross-Language Information Retrieval (CLIR) is concerned with retrieving the documents written in a language different than the language of the users' query. Earlier methods of such retrieval consisted of translating a query into the documents' language, and then using the standard techniques of IR. The query translation in these approaches was performed using machine-readable bilingual dictionaries, through machine translation, or using parallel corpora. However, no such translation is absolutely correct, and regardless of the translating resource used, there are usually limitations due to insufficient coverage, for example, there are terms which cannot be translated or do not require the translation, and due to the translation ambiguity from the source language to the target language.

To reduce the errors introduced due to translation, one standard technique is to use query expansion, so that even when the translation does not contain errors, use of semantically similar terms yields better results than those due to the literal translation of terms only. The query expansion can be applied either before the translation of query, or after, or even can be applied at both the places. It has been found that query expansion, before the translation of a query, yields better results than doing it after the translation. The expansion done at both the places provided even better results.

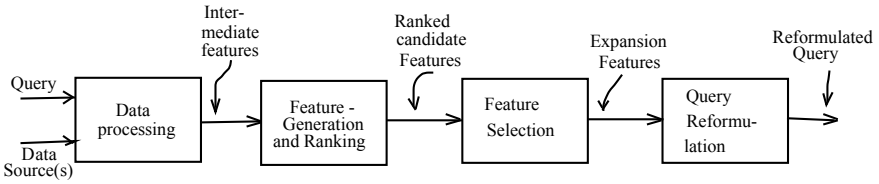

Fig. 18.9 Automatic Query Expansion process

18.9.1 Working of AQE

The Automatic Query Expansion (AQE) is performed in a number of steps, the major steps are

1. Preprocessing the source text,
2. Generating expansion terms (features) and ranking them,
3. Selection of expansion features, and
4. Reformulation of Query.

Figure 18.9 shows these steps of AQE.

The objective of preprocessing the data source data is to transform the raw data source used for expanding the user query into such a format so that it can be processed more efficiently by the subsequent steps. The preprocessing task performs extraction of intermediate features, followed by the construction of data structures for easy access and manipulation of such features. The preprocessing is usually independent of a particular user query to be expanded, but it is specific to the type of data source and expansion method used [1].

To compute the initial retrieval run, it is necessary to index the documents collection, and then run the query against this collection's index. The process of indexing consists of, in order, the following steps:

1. text extraction from documents, which are in a certain format, like HTML, PDF, MS Word, where there is format information as well as text inside them,
2. tokenization of the extracted text,
3. stop-word removal from the tokenized text (removal of common words such as articles and prepositions),
4. stemming (reduction of inflected or derivational words to their root form), e.g., reduce "tokenize, building, training" into "token, build, train", etc.
5. word weighting (score is assigned to each token such that the weight reflects the importance of the tokens in each document).

We take an example of a short HTML text fragment to illustrate the weights associated with tokens.

```
<p><b> An automatic query expansion </b>
increases the query's semantics.</p>
```

In the above HTML document, first, text is extracted, then stop words "the" and "an" are removed, then it is stemmed using Porter's stemmer, and weight is assigned to each word based on their frequency. Finally, we obtain the text representation as follows:

automat 0.16, *queri* 0.33, *expan* 0.16, *increase* 0.16, *semantic* 0.16.

This is an example of a very small document, however, it gives an understanding that each document can be represented as a set of weighted terms, such that the total weight of the document is 1. The index is created in the form of a complementary *inverted index* file, which maps terms to documents at the time of query. To reach the location of index terms in the document faster, the system may also store the terms' locations to provide proximity-based search.

The original query is preprocessed to remove the *stop words* and/or extract important terms to be expanded. In the second stage of AQE, the system generates and ranks the candidate expansion terms; most query expansion methods choose only a small proportion of the expansion features (i.e., terms) to be added into the query. Input to this stage of AQE is the original query and the transformed data source, and the output is a set of expansion features, with/without scores.

Once the ranking of the candidate features is carried out, the top elements are selected for query expansion. Selection of these top elements is performed on an individual basis, without regard to mutual dependencies between the expansion features. Usually, only a limited number of features are selected for expansion such that 1. resulting query is not bulky, thus helpful for processing faster, and 2. retrieval effectiveness of a small set of good terms is not necessarily less successful than the effectiveness we get by adding all candidates' expansion features. The addition of expansion features will also be helpful in reducing the noise.

Sometimes, the feature scores are interpreted as probabilities. In that case, only the terms having a probability greater than a certain threshold are selected for consideration.

The last stage of AQE is query reformulation, which describes the expanded query that will be submitted to the IR system. The description means the assignment of proper weight to each feature that is part of the expanded query—the process called query *re-weighting*.

The total time required for an AQE is the sum of two factors, 1. cost of generating expansion features, and 2. increase in the cost for the evaluation of the expanded query (due to its size), against the documents collection. In practice, the second factor is a more critical one. Consider the architecture (data structure) of most ranking systems, which are based on the inverted (linked) lists of N elements, one for each term in the collection. Here, each inverted list specifies the documents in which the particular term occurs, along with a pre-computed score for each term. At the time of query processing, the system retrieves the inverted list of every query term, and updates the score accumulators of the documents present in each list. The execution time of a ranked query is almost linearly proportional to the number of terms in the query;

this is because the query terms are processed one at a time. The AQE runs with sizes of practical interest, for example 10–20 word queries were found to be much slower (by a factor of ten), than the original queries of 3–4 words.

18.9.2 Related Techniques for Query Processing

The mismatch of words between the query and documents for relevant documents is an issue in IR for a long time. In the following section, we discuss AQE with respect to alternative strategies in reference to the vocabulary problem [1].

Interactive Query Refinement

In interactive query refinement, the system provides several suggestions for reformation of the query, and it is the user who selects the best choice out of that. With respect to the computations required to be performed, in Interactive Query Refinement (IQE) versus AQE, the first two stages are common with both, i.e., data acquisition and candidate feature extraction. The IQE does not follow the steps of feature selection and query reformulation of AQE. One of the best-known examples of Interactive Query Refinement is the suggestion of complete query, which offers real-time hints to user to complete a search query. This happens when a user progresses in typing the query in the inbox, like we note in many search engines, including Google. The IQE has better potential for superior results than AQE, but generally requires higher expertise on part of the user. Looking from the usability point of view, an IQE provides the user with better control over query processing than the AQE.

Relevance Feedback

The relevance feedback takes two inputs: 1. results initially returned due to the given query, and 2. feedback provided by the user about whether those results are relevant or not. Based on these two inputs, the system submits a new query to the search engine provided that previous results returned were not considered useful for fulfilling the need of the user.

The features in the assessed documents are used to adjust the weights of terms in the original query and/or for adding words to the query. The relevance feedback has the effect of reinforcing the system's original decision. This is done by the user by modifying the expanded query to look closer at the retrieved relevant documents. However, the AQE tries to form a better match with the user's existing intentions, and does not give the user a second chance to rethink, support/reject the results produced by the first query. The specific data sources using which the expansion features are generated in the relevance feedback may be more reliable than the sources generally used by AQE. In the relevance feedback, the user must assess the relevance of the documents, requiring a user to be better trained.

The relevance feedback has directly inspired one of the most popular AQE techniques, called *pseudo-relevance feedback*, which has also provided foundations for modeling query reformulation in a variety of AQE approaches.

Word Sense Disambiguation in IR

The Word Sense Disambiguation (WSD) is the ability of the system to correctly identify the senses of words in context to the remaining (surrounding) text in a document. This identification is carried out in a computational manner. WSD is a natural and well-known approach to the vocabulary problem in IR—usually, even if we do not know the meaning of a word, for example, in a newspaper, we are still able to understand the sentence, as well as able to discover the meaning of that unknown word due to its context words in a sentence [5].

Early work of WSD concentrated on representing words using their dictionary definitions, or using the WordNet Synsets. But, many experiments suggested that this straightforward technique is not effective in IR, at least as long as the selection of the correct sense from the resource is flawed. For example, if precision does result in a good value, say greater than 75%. However, more sophisticated methods in AQE still used the WordNet resource [11].

Instead of depending on short predefined lists of senses, using a corpus is found to be more convenient as an evidence for performing the Word Sense Disambiguation. Due to its nature of the process, it may be called as word sense induction.

In one approach based on the corpus, the context of every occurrence of a word in the corpus is identified and similar contexts are clustered together to help in determining the word senses. This method can provide a maximum disambiguation rate of 90%, hence can be used successfully with an IR system. With the reliance of this method on corpus-based analysis, this approach is similar in spirit to the global AQE techniques.

In another corpus-based WSD technique, a metaphor of small words is applied to word co-occurrence graphs, due to which it is capable of discovering low-frequency senses (senses which are not very common), that are as low as 1%.

Further, in the context of a query to web search engines, where the query is too small, it may be too difficult to disambiguate the word senses in it in the absence of sufficient context available in the query. Therefore, longer queries due to their higher contexts are likely to be helpful in performing the disambiguation, and consequently to produce better results. As a whole, we note that the application of WSD to IR presents the challenges of computational nature, there are limitations of effectiveness as well.

Search Results Clustering

The objective of Search Results Clustering (SRC) is that for the new queries, exactly similar to some previous queries by users, the IR system should store the previous results in some compact forms, so that it can provide the results directly, without performing any search process. For this, the SRC organizes the search results topic-wise, which allows direct access to the documents relevant to the user queries, making overall IR far faster. In contrast to the standard clustering techniques, the SRC algorithms try to optimize the clustering structures, as well as the quality of cluster labels. This is because, a cluster with a poor description (labels) is likely to be entirely omitted by the user, even though it may be pointing to a group of strongly related and relevant documents.

18.10 Using Bayesian Networks for IR

A Bayesian network is an annotated directed graph, which can be used to encode a probabilistic relationship among the distinctions of interest in an uncertain reasoning problem. The representation rigorously describes these relationships, and can provide a human-oriented qualitative structure which facilitates a communication between a user and the system based on a probabilistic model. As the computing power is available chiefly even in small systems, the modeling tools based on Bayesian networks are abundantly used in real-world applications, e.g., in forecasting, fault diagnosis, sensor fusion, anti-virus software, automated vision, and manufacturing control [10], [13].

18.10.1 Representation of Document and Query

For understanding the basics of Bayesian networks as well as Naive Bayes, the reader may refer to the previous chapters (section 12.4, page no. 344, and section no. 14.7, page no. 428). Using the conditional probability, the probability that a document d_k is relevant to the query q_j can be expressed as $P(R \mid q_j, d_k)$. An accurate definition of probability of relevance depends on the definition of relevance itself. The term *relevance* is to some extent a subjective entity, and depends on many variables, which are functions of documents, user's information need, and the user itself. A perfect retrieval is not achievable in true sense. However, it is possible to define an optimal retrieval for the probabilistic model for IR. This optimal retrieval can be proved theoretically with respect to representations (or descriptions) of documents and information needs [4].

Let us assume that collections of queries and documents are described by a set of index terms. Let $T = \{t_1, t_2, ..., t_n\}$ be the set of terms in the documents' collection, and a query q_j and document d_k are taken as subsets of terms in T. For the sake of retrieval, each document is described by the presence/absence of these index terms. Therefore, any document d_k can be represented using a *binary vector*:

$$\overrightarrow{x} = (x_1, x_2, \ldots, x_n), \tag{18.15}$$

where any term $x_i = 1$ if $t_i \in d_k$, and for $t_i \notin d_k$, the $x_i = 0$. A query q_j is also represented in similar way. The basic task of a relevance model-based IR system is to compute the probability that a given document is relevant. This can be achieved by estimating the probability $P(R \mid q_j, d_k)$, for every document d_k in the collection. Since relevancy in every document is computed for a single query, the term q_j being common, and can be dropped, and the relevancy can be expressed using the Bayes theorem as

$$P(R \mid \overrightarrow{x}) = \frac{P(\overrightarrow{x} \mid R)P(R)}{P(\overrightarrow{x})}. \tag{18.16}$$

In the above,
$P(R \mid \overrightarrow{x})$ is called *posterior probability*—the probability of relevance, given that the document is $\overrightarrow{x}$,
$P(\overrightarrow{x} \mid R)$ is called *likelyhood function* or *probability of evidence*, which is the probability of randomly selecting the document of description $\overrightarrow{x}$ from the set R of relevant documents,
$P(R)$ is *prior probability* of relevance, i.e., the probability that a randomly selected document from the entire collection is relevant, and
$P(\overrightarrow{x})$ is probability that a randomly selected document has a description $\overrightarrow{x}$. It is determined as a joint probability distribution of the n terms in the collection [6].

Equation (18.16) can be expressed in simple language as

$$Posterior\ probability \propto likelihood \times prior\ probability.$$

18.10.2 Bayes Probabilistic Inference Model

The Bayes probabilistic inference makes use of a network, which is an extension to the probability-based IR. The network is a Directed Acyclic Graph (DAG), in which nodes represent *propositional variables* or constants, and the edges represent the dependency relations between the propositions [10].

If p and q are two propositions, and there is a relation of implication from p to q, i.e., the first proposition "causes" the second, then p is cause and q is effect, and is represented by $p \rightarrow q$. In the DAG, p and q are nodes, and there is an edge from a node marked as p to node marked as q. A link matrix is stored at node q, which specifies the probability $P(p|q)$ for all possible values of variables p and q. The expression $P(p|q)$ is the expression for the probability of occurrence of event p given that q has already occurred. In the model we take q node as evidence, and it stands for a query. In a scenario, a node has more than one parent (say p_1, p_2, ...), the link matrix will indicate the dependence of query node q on all the parents. The query node q now characterizes the dependence relationship between itself and all the nodes p_1, p_2, ..., which are potential causes. This is illustrated in Fig. 18.10. Using the Bayes theorem, the conditional probability expression can be expanded as

$$P(p|q) = \frac{P(q|p).P(p)}{P(q)}. \tag{18.17}$$

When the set of prior probabilities $P(p)$ are given for the root of a DAG that represents the document, the network can be used to compute the probability of belief associated with all the remaining nodes. Figure 18.10 shows a document d_i at

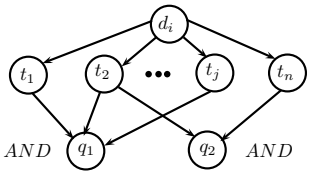

Fig. 18.10 Basic Inference Network Model for IR

root, corresponding keywords $t_1, \ldots, t_n$, and submitted queries q_1, q_2. Note that, t_1, ..., t_n are the representation of document d_i [17].

Through the inference network, the *random variables* are associated with the documents, index terms, and user queries. Multiple evidences of query terms in the document's representation for a given query are conjuncted to estimate the probability that the document satisfies the user's information need. For example, in Fig. 18.10, query q_1 is conjunct (ANDed) of terms t_1, t_2, t_j, and $q_2 = t_2 \wedge t_j \wedge t_n$. Thus, variables associated with document d_i represent the event that the document is observed. The index terms/document variables are represented as nodes of the DAG, and the edges, which are directed from document nodes to index terms nodes indicate that the observation of document results in an improved belief on its term nodes.

Further, a random variable associated with the user's query node models the event that information need expressed in the form of user's query has been met. The dependency in the form of direction arrows indicate that belief in the query node is function of beliefs in the nodes that correspond to the query terms. In Fig. 18.10, document d_i comprises $t_1, t_2, \ldots, t_j, t_n$ as its index terms. Similarly, the query q_1 comprises the query terms $t_1, t_2, \ldots, t_j$, hence, $q_1 = t_1 \wedge t_2 \wedge t_j$, and $q_2 = t_2 \wedge t_n$.

From this Bayes inference model for IR, we note that a set of edges pointing to a node represents the probabilistic dependence between that node and its parents. Through its structure, the Bayes network represents the conditional dependence relation among the variables in the network. These dependence relations provide a framework for retrieving the information [4, 6].

18.10.3 Bayes Inference Algorithm

For IR using Bayes inference, a user specifies one or more topics of interest while keeping in mind some document features, the latter are to be used as evidence for topics of interest mentioned above. The task of IR using Bayesian inference network is documented as Algorithm 18.5, which requires building an inference network for the representation of query terms and document features (i.e., terms), and computation of posterior probabilities ($P(p|q)$ in Eq. 18.17) based on the prior probability of the document [6, 10].

Algorithm 18.5 Bayesian Inference-based IR

1: *Construct the network representation of query*
2: {Steps to score all the documents}
3: **for all** documents **do**
4: *Extract the features* $\{t_1, t_2, ...\}$ *from document*
5: *Label features in network*
6: *Compute posterior probabilities*, $P(p|q)$
7: **end for**
8: *Rank the documents set in order of posterior probabilities*

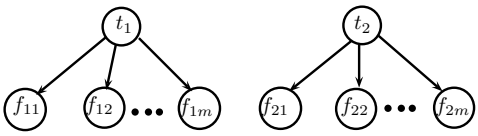

Fig. 18.11 Two-Level Bayesian network model for IR

In this algorithm, steps 4, 8 are IR routines, whereas steps 5, 6 are routines to draw inferences.

Figure 18.11 shows the term-weighting architecture of Bayesian network, which indicates that there are topics of interest, shown as t_1, t_2, and there are features to be examined, $f_{11}, ..., f_{1m}, f_{21}, ..., f_{2m}$. The features' set and topics set shown here are different, but they can be the same also. The occurrence of the topics t_1, t_2 on graph nodes represent the event that the document is related to topics t_1, t_2. Whereas, the nodes corresponding to the features represent the events, for example, the features $f_{11}, ..., f_{1m}$ represent the event that these features are present in the document d_1.

The network structure in Fig. 18.11 is based on the following assumptions concerning the Bayesian probability:

1. Given the topics $\{t_i, t_2, ...\}$ from the document (the document is relevant to these), the presence or absence of any feature does not imply about the presence or absence of some other feature. In other words, it is assumed that there are no dependency relations between the features.
2. Given that the document relevant to one topic does not affect one's belief about the relevance or non-relevance of that document to any other topic.

In the above, the first assumption specifies the conditional independence of features, when the topic is already given. It is called as *binary independence*. The absence of arcs between feature nodes in Fig. 18.11 is an explicit indication of independence between features. Given these conditions and the Bayes network in Fig. 18.11, we can draw some important conclusion. That is, the assumption "1" will not be valid, if the query includes features that are identical or closely related, like synonyms, because in that case the features are not independent.

Now, for the network in Fig. 18.11, we define two sets of probabilities, given below:

(a) $P(t_1)$, $P(t_2)$, .., called *prior probabilities*, that a document is relevant to topics $t_1, t_2, ...$, etc. If we are discussing one document only, then it is $P(t_1, t_2, ..)$.

(b) The *conditional probability* $P(f_{ij}|t_i)$ of each feature f_{ij} is defined as the probability that feature f_{ij} is present in the document, given that this document is relevant to topic t_i.

Given the above, the task of IR is to compute the *posterior probability* $P(t_i \mid f_{i1}, \ldots, f_{im})$, which is the probability that the document is relevant to t_i, given that we have observed the presence or absence of all of the features f_{ij}, called evidences.

The Bayes rule can be directly applied for this computation. The network topology shown in Fig. 18.11 is called *Bayes inference* and, can be expressed by

$$P(t_i \mid f_{i1}, \ldots, f_{im}) = \frac{P(t_i).P(f_{i1}, \ldots, f_{im} \mid t_i)}{P(f_{i1} \ldots, f_{im})}. \tag{18.18}$$

Generally, we are not interested in absolute numerical values of the posterior probabilities, but want to just rank the documents by the posteriors. Thus, we can eliminate the denominator term $P(f_{i1} \ldots, f_{im})$ in this equation as long as this denominator remains the same, with varying t_i,s. Further, we can simplify this Bayes rule to a *linear decision rule* given in Eq. 18.19, where $I(f_{ij})$ is an indicator variable that equals to 1.0 only if f_{ij} is present in the document and 0.0 otherwise, and w is a coefficient corresponding to a specific (feature, topic) pair:

$$g(t_i \mid f_{i1}, \ldots, f_{im}) = \sum_j I(f_{ij}) \times w(f_{ij}, t_i). \tag{18.19}$$

A careful choice of w results in a ranking of documents in descending order of $g()$, which turns out to be in the same order as that of ranking them in decreasing order of the posterior probabilities. However, since $g()$ does not include the priors probabilities ($P(t_i)$) of the topics t_i, which is indicator of the relative *rarity* of the topics, one cannot compare a document's strength of relevance to one topic with respect to its strength of relevance to a different topic.

The coefficients w can be interpreted as weights corresponding to each feature f_{ij} and term t_i. Similarly, the function $g()$ can be interpreted as the sum of weights of the features f_{ij} that are present in the document, which is relevant to topic t_i. Hence, this method is known as "term weighing".

In the Bayes network topology shown in Fig. 18.11, the query corresponding to each topic, e.g., t_1, is represented by its own subnetwork, which is shown disconnected from the subnetworks of other topics' queries. Hence, these isolated models fail to represent the possible relationships between the topics, for example between t_1, t_2. Therefore, it is difficult to acquire consistent, feature-conditional probabilities, as well as find out the probabilities by combining the topics.

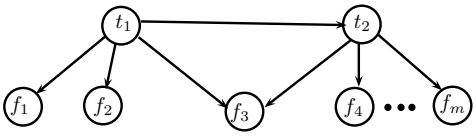

Fig. 18.12 Information retrieval model with two related topics

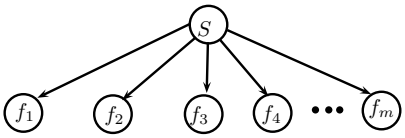

Fig. 18.13 Multi-topic query as a single compound-topic query

18.10.4 Representing Dependent Topics

In the previous section, we discussed that, a document relevant to one topic does not affect one's belief about the relevance or non-relevance of that document to any other topic.

An IR network topology that removes this assumption, and explicitly represents the relationships among different topics is shown in Fig. 18.12. We note that the Bayesian inference problem becomes more completed with multiple topics.

The addition of a relationship between topics in Fig. 18.12 requires two changes in specifying the network probabilities. In the first one, we must specify what are those topics (t_1, t_2, ...), and then compute the strength of the relevance between them. In the second step, we must compute the probability of each feature (f_{ij}), conditioned on each combination of its parent topic. Having done in this manner, it gives rise to a number of probabilities, which are combinatorial in size.

However, there is an approach to simplify the multiple-topic network, such that it looks like a single-topic network, and its range is only all the possible present/absence combinations of topics t_1, t_2, ..., etc. Thus, in a way we have translated the topology from one form to another. The modified topology of Fig. 18.12 is shown in Fig. 18.13, where node S represents the compound topic. An advantage of this representation is that the same set of formulas (18.18) and (18.19) for the computation of probabilities can be used now also. The disadvantage of this approach is that the compound query shall contain 2^n states for n parent states, which will complicate the computations.

18.11 Semantic IR on the Web

The semantic web is a vision of the future WWW. It is an extension of the current web, where information is given with well-defined meanings, that will better enable the computers so that computers and people can work in cooperation. However, a universal implementation of the semantic web—a full substitution for the existing web—is still far away from reality. Therefore, it could be useful to have a system that analyzes the documents from a contextual point of view for more accurate

retrieval. The semantic IR (SIR) on web is based on computing semantic relations to evaluate the relevance of documents with a query in a given context, and makes use of structures like lexical chains, semantic networks, and ontologies. The semantic-based approach is context-driven, where keywords (topics/terms) in documents are processed in the context of the information in which they are retrieved. This will help solve the semantic ambiguity so that the retrieval is accurate, and as per the true need of the user [15].

In the recent times, the information and knowledge representation using ontologies have acquired great importance, as it is found to be suitable for strategic requirements. These strategies are intrinsically independent on information codification, which helps to isolate the information, as well as to recover, organize, and integrate it with respect to its content.

Following are the definitions of ontologies.

Definition 18.1 *Ontology.* An explicit and formal specification of shared conceptualization is called an ontology. It is an abstract model of specified reality, such that the components are clearly identified. The terms in definition are further clarified as follows:

– *Explicit* means type of concepts used and the constraints on them are well defined,
– *Formal* refers to the ontology property of being machine-readable, and
– *Shared* means, a property of ontology of capturing consensual knowledge, accepted by a group of persons. □

Definition 18.2 *Ontology* (Definition-2). An ontology defines the basic *terms* and their *relations* consisting the vocabulary of a topic, as well as the *rules* for combining terms and relations to define extensions to the vocabulary. □

The above definition also indicates a path to be followed in order to construct an ontology:

1. first identify the basic terms and their mutual relations;
2. agreement on the rules that arrange them;
3. defining of terms, and relations among concepts.

From the above perspective it is clear that an ontology does not include just the terms that are explicitly defined in it, but keeps provision to derive new terms using defined rules and properties. Also, the ontology can be viewed as a "set of terms and relations between them, which denote the concepts used in a domain."

The concept of semantic relatedness refers to the relations between words and concepts that are in the practice, or those based on the perceptions. There can be several metrics to measure the semantic relatedness of words. Some of the approaches to these metrics are as follows.

- *Thesaurus-based metrics.* These metrics make use of thesaurus where words are related to concepts, and each word is referred to a category by an index structure.

- *Dictionary-based metrics.* Dictionaries are linguistic information sources of our knowledge about the world; they form a knowledge base in which headwords are defined using other headwords and/or their derivatives.
- *Semantic network-based metrics.* These metrics use semantic networks—graphs in which the nodes are the concepts, and the arcs between nodes represent relations between concepts. Number of edges' links between terms (nodes/concepts) in the semantic network, without loops, is a measure of conceptional distance between terms (nodes).

A sequence of related words in a text is called *lexical chain* (a linguistic structure), which may span short distances (adjacent words or sentences) or long distances, covering the entire text. Computing a lexical chains helps in the identification of the main topics of a document. The semantic relatedness measures use lexical chains to perform their computations, the lexical chains are used for IR and related areas, and to explore structure of texts as well. The lexical chains have been also used to index video-conference transcriptions by topic, construction of typical IR system and text segmentation systems, and for automatic generation of hypertext links.

The strength of a relation between words that connect different fragments of the text is measured by their *cohesion*, and the cohesion between lexical units of text is called *lexical cohesion*—the most common type of cohesion. The lexical cohesion can be expressed by repetitions of relations like *synonym*, *hyponym*, or by other linguistic relations between words, such as whole-part and object-property.

Semantic IR Systems

Use of ontologies in IR consists of an approach that identifies important concepts in documents using criteria of semantic relatedness and co-occurrence; this is followed by disambiguation of them using an external general purpose ontology (e.g., WordNet). On matching the ontology with a document provides a set of scored concept senses (nodes) with weighted links, called semantic representation of the document.

The steps for the process of Semantic IR (SIR) are as follows:

1. For improving the web searches, only important information is selected from the user query, which is helpful in extracting information from documents;
2. The user's query or a phrase expressed in natural language is sent to a lexical processing module;
3. The boundaries of words and phrases are detected through tokenization. The system labels the words using some tagger like Brill's tagger;
4. A phrase parser splits every phrase into several members, as nouns and verbs;
5. The stop words are removed, and the system uses some keywords to represent the main concept of the phrase;
6. The remaining process steps of SIR, like Word Sense Disambiguation (WSD), query expansion, and post-processing, have been already discussed in this chapter.

To solve the problem of polysemy,[5] the concept of "word sense" from WordNet is used, which helps the user interacting with the system to associate with every concept a list of terms semantically are related to.

18.12 Distributed IR

When the size of data is very large, or when it is required to support high query volumes, single machine is not enough to support the load of IR, even when the number of the enhancements discussed above have been used. For example, even when the Internet was not so common, in mid-2004, Google search engine processed more than 200 million queries a day against more than 20 GB of crawled data, and it used over 20,000 computers. The requirement in the present time is hundred times bigger. For handling large size of data loads, a combination of *distribution* and *replication* is necessary. The distribution means, document collections and their indexes are split across multiple machines, so that answers to every query is synthesized from many collections of components. The word replication means *mirroring*, which involves making enough identical copies of the system so that the required query load can be handled with acceptable minimum response time [18].

Document-Distributed Architectures

A very simple distributed document's architecture is to partition the collection and allocate one sub-collection to each of the separate processors (see Fig. 18.14). To make use of distributed document architecture, a local index is built and maintained for each sub-collection, and when a query arrives, it is passed to every sub-collection to search and evaluate in parallel against every local index. The sets of sub-collections' answers are then merged in some way to provide an overall answer. The main advantages of such document's partitioning system is that, collection growth is handled by designing one of the hosts in the form of dynamic collection, such that only this host needs to rebuild its index. The parts of the process that are computationally expensive are distributed equally across all the hosts in the computer cluster. These parts are searching the index, updating individual indexes, computation of weights, and idf for documents, etc.

Figure 18.14 shows a simple inverted index of a document-distributed retrieval system. It shows two index partitions: 1. a term-based index partition, and 2. document-based index partition. Elements in a inverted list have format ⟨document number, term-frequency⟩, for example ⟨2, 2⟩ in second row, second column indicates that the term "quick" has frequency 2 in the document number 2, while ⟨3, 1⟩ in the same row, third column indicates that the term "quick" has frequency 1 in the document number 3. Each of the two dashed regions in Fig. 18.14 show one component of a document-distributed retrieval system; with this, one processor indexes all terms

[5] Polysemy: A single term with several meanings.

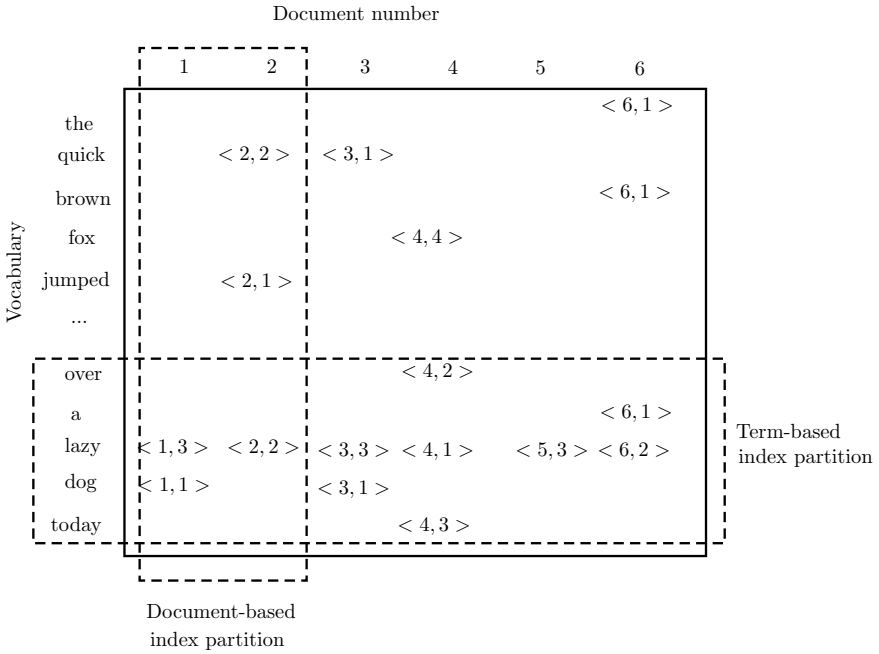

Fig. 18.14 Partition and distribute index across a cluster of machines

that appear in the first two documents of the collection, creating a document-based partition.

Term-Distributed Architectures

An alternative strategy for partitioning the index is term-based partitioning, where index is split into components by partitioning the vocabulary, with one possible partition shown by the dotted horizontal split in Fig. 18.14. Each processor has full information about a subset of the terms, i.e., all the necessary things to handle a query. Hence, only the relevant subset of the processors needs to respond to a query. The term partitioning has the advantage that it requires fewer disk seek and transfer operations during query evaluation than document partitioning because each term's inverted list is still stored contiguously on a single machine rather than being split in fragments across multiple machines.

On the other hand, each of these disk transfer operations involve more data. Mainly because, in a term-partitioned arrangement we are discussing, the majority of the processing load is on the coordinating machine; the experiments have shown that it can easily become a bottleneck and starve the other processors of work.

Term-Based Versus Document-Based Partitioning

Compared to the term-based partitioning, the document-based partitioning typically results in a better balance of workload and achieves higher throughput for queries.

Also, the document-based partitioning (document distribution) allows for more natural index construction and for document insertion. On the other side, for term-partitioned index, index construction involves first of all, distributing the documents and building the document-partitioned index. Then, once the vocabulary split has been agreed upon by the processors, the index fragments are exchanged between all pairs of processors.

The document-based partition also has the practical advantage of providing the search service even when one of the hosts is offline for some reason, because any answers not resident on that machine are still available to the system. For example, there are 10 machines, $m_1 ... m_{10}$ and 100 documents $d_1 ... d_{100}$, with document distribution as follows:

machine $m_1 : d_1 ... d_{10}$,
...
machine $m_{10} : d_{91} ... d_{100}$.

Now, if machine m_5 is offline, the queries for all the documents, except $d_{51} ... d_{60}$, is answered. Whereas, in the term-based distribution, since index terms of every document are uniformly distributed on all the processors, the query cannot be answered even if one processor is offline. However, in the term distribution case, if any machine is offline or idle, it is immediately noticeable because it will affect queries belonging to almost every document.

Google indexing uses document-based partitioning, with massive replication and redundancy at all the levels, for example at machine level, cluster level, and individual level. In addition, the document partitioning remains effective even if the collaborating systems are independent and unable to exchange their index data. The distributed system makes use of a *meta-searcher*, using which the final result answer list is synthesized from the possibly overlapping answer sets provided by a range of different services.

18.13 Summary

Information retrieval (IR) process is the identification of documents or other units of information in a collection that are relevant to particular information needs expressed through queries for which people are interested to find answers. The IR models consider that each document is described by a set of representative keywords, called *index* terms—a word from a document that represents the semantics of the document. In general the index terms are *nouns*. If t_i is an index term, d_j is a document then $w_{i,j} \geq 0$ is the weight associated with the pair (t_i, d_j), or single entity w_{ij}.

Two ratios, *precision* and *recall* are used to measure the effectiveness of an IR system. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved, i.e., what fraction of the retrieved documents are relevant, and recall is the ratio of the number of relevant documents retrieved to

the total number of relevant documents in the entire collection, i.e., what fraction of the relevant documents have been retrieved.

The common retrieval strategies are Boolean Model, Vector Space Model, Probabilistic Model, Inference networks, and Fuzzy set-based retrieval.

The *Boolean* model is a simple retrieval model based on Set Theory and Boolean Algebra, Vector Space Model computes the measure of similarity ($sim(q, d_j)$) between query $q \in Q$ and a document $d_j \in D$, where q and d_j are vectors for query and document, and the Q, D are sets for queries and documents, respectively.

The probabilistic retrieval model computes the similarity measure ($sim(q, d_j)$) between the query q and a document d_j as the probability that document d_j is relevant to q. This model estimates a query *term's weight* on how often the term appears or does not appear in *relevant* and *non-relevant* documents, respectively. One class of probabilistic approach for IR is Bayesian networks. These networks are annotated directed graphs encoding probabilistic relationship among distinctions of interest, in an uncertain reasoning problem.

The fuzzy retrieval technique is based on fuzzy set theory and fuzzy logic—an extension of the classical set theory. The word matching between the query set and the text word is not limited to the perfect matching, but, the matching is graded, depending on the degree or level of matching in the range from 0 to 1. In *fuzzy set* -based retrieval, a membership value $R(x_i, y_i)$ specifies for each $x_i \in X$, $y_i \in Y$, the grade of relevance of index term x_i with the document y_i.

In a *concept-based* information retrieval, queries and documents are represented using *semantic concepts*, instead of (or in addition to) keywords, and they perform retrieval in that concept space. This results in a retrieval model that is less dependent on the specific terms used, and yields matches even when the same sense is described by different terms in the query and target documents.

The retrieval models make use of an *inverted index* file structure. Instead of searching into a document, an inverted index is generated in advance for all the keywords in the document set. For each term, a pointer references to a linked list, which contains an entry for each document containing this term as well the term frequency in that document. The single key problem with this index is that volume of data involved cannot be held in the main memory. To solve this problem, the indexes are constructed in parallel and can be merged after a regular intervals.

Yet, there is another method for IR that uses an approach based on the measure of semantic relatedness, applied to evaluate the relevance of a document with respect to a query in a given context. The approach makes use of structures like *lexical chains*, *ontologies*, and *semantic networks*. The semantic approach implements a context-based system, such that keywords are processed in the context of the information from which they are retrieved. This approach helps in solving semantic ambiguity, and results in giving a more accurate retrieval, that is based on the real-world interests of the user.

In most information retrievals, user queries are short and the natural language is inherently ambiguous, consequently, the retrieval is prone to errors and omissions. The most critical language issues are polysemy and synonymy. To resolve this, the

query is expanded by appending the synonyms of query terms into the query, called Automatic Query Expansion (AQE).

When large volumes of data sets are involved or when the query volumes are high, one machine may be inadequate to support the users' query load, even when the various enhancements and optimizations are carried out. For handling heavy load of users' queries, a combination of *distribution* and *replication* is required. Distribution means, the document collection and their indexes are split across multiple machines (servers) and that answers to the query as a whole must be synthesized from the various collection components. Replication (or mirroring) involves making enough identical copies of the system so that the required query load can be handled at speed and accuracy.

Exercises

1. Show how the vector space model can be modeled using an inference network.
2. Consider a documents collection made of 100 documents. Given a query q, the set of documents relevant to the users is $D^* = \{d_4, d_{15}, d_{34}, d_{56}, d_{98}\}$. An IR system retrieves the following documents $D = \{d_4, d_{15}, d_{35}, d_{56}, d_{66}, d_{88}, d_{95}\}$

   a. Compute the number of True-Negatives, True-Positives, False-Negatives, False-Positives.
   b. Compute Precision, Recall, and F-measure.

3. Consider an IR scenario in the following: It has been found in some hospital, results of blood tests taken on a specific day are unreliable for diabetic patients due to equipment malfunction. The hospital uses an IR system to identify these patients. Suppose the collection of patients' records contains 10, 000 documents, 500 of that are relevant to the query. The system returns 350 documents, 225 of that are relevant to the query. Answer the following for this scenario:

   a. Calculate the precision and recall for this system.
   b. Based on your results from above, explain how well would you say about the working of hospital's IR system.
   c. Knowing about the precision-recall trade-off, what is likely to happen if an IR system is tuned to aim for 100% precision?
   d. Knowing about the precision-recall trade-off, what is likely to happen if an IR system is tuned to aim for 100% recall?
   e. For the trade-off given scenario, which measure do you think is more important, precision or recall? Why?

4. You are looking for information on "Economic growth in India" in a large document collection, during the period of last 3 years. You decide to search using the terms: *India, banks, growth, economy, business, agriculture*, using an IR system, which recommends three possible documents given below with term frequencies.

Term	Economy	India	Growth	Banks	Business	Agriculture
Document-1	15	10	3	4	2	9
Document-2	0	0	9	8	7	8
Document-3	4	2	4	4	6	10

There is no additional information about the documents. Make use of each of the following models to find out the relevancy of the documents to the query.

a. Boolean model
b. Vector space model
c. $tf.idf$ model

5. Take any three small documents of size, approximately 100 words.

a. Build a matrix of an inverted index for these documents, in the format shown in Fig. 18.5.
b. Weight terms by their presence/absence (binary), and also and by $tf \times idf$ (with estimated IDFs).
c. Compute the memory requirements for this inverted index. Make necessary assumptions for character size, pointer size, etc.
d. Construct a suitable query, and calculate document–query similarity, for the following scenarios:
   i Cosine (with normalization)
   ii Inner product (i.e., cosine without normalization)
   iii Does the normalization has any effect? Justify.

6. Consider that we submit the queries to search engines for searching the needed information on WWW.

a. Does the search process use a stop-word list?
b. Can you search "The", "The a", "An a", etc.? Justify.
c. Is it a practice to search the above terms?
d. Does the search process use stemming?
e. Are there different results for two queries "Human body", "Humanly body". Justify your answer.
f. Does it normalize words to lower case?

7. "Having the knowledge of the sense of a query term may help a document retrieval system, especially for short queries." Why it is not true for longer queries?
8. Comment on the validity of following statements for Boolean model:

a. "Stemming does not lower the precision of a Boolean retrieval system."
b. "Stemming does not lower recall of a Boolean retrieval system."

9. Answer the following in brief:

a. Why is the *idf* of a term always finite?
b. What is the *idf* of a term that occurs in every document?
c. What is the *idf* of a term that appears in one document only?
d. What is the *idf* of a term that appears in no document?

10. Answer the following in brief:

a. Name three criteria for evaluating a search engine.
b. What is an easy way to maximize the recall of a search engine?
c. What is an easy way to maximize the precision of a search engine?

11. What is the difference between *clustering* and *classification*? How can they be used in a complete IR system?
12. Discuss the merits and demerits of following, suggest as to which one will provide better response time?

a. Document-distributed architecture.
b. Term-distributed architecture.

References

1. Carpineto C, Romano G (2012). A Survey of Automatic Query Expansion in Information Retrieval. ACM Comput. Surv. 44(1): 50.https://doi.org/10.1145/2071389.2071390
2. Chowdhary K R, Bansal VS (2001) Current trends in information retrieval. In: The 4th International Conference of Asian Digital Libraries, Dec. 10–12, 2001 Bangalore, pp. 306–319
3. Chowdhary K R, Bansal VS (2003) Fuzzy Logic-based information retrieval. In: Conference proceedings on algorithms and artificial systems, Allied Publishers Pvt. Ltd. pp 297–307. ISBN 81-7764-403-3
4. Chowdhary KR (2004) Natural language processing for word sense disambiguation and information extraction. PhD Thesis, J.N.V. University, Jodhpur, May 2004
5. Chowdhary KR (2005) Word sense disambiguation. J Comput Sci 1(1):30–37
6. Chowdhary KR (2008) Information retrieval from digital libraries using probabilistic-possibilistic inferences. In: IR@INFLIBNET INFLIBNET's Convention Proceedings CALIBER 2008 Allahabad, http://ir.inflibnet.ac.in/handle/1944/1225
7. Chowdhary KR, Bansal VS (2006) Information extraction from natural language texts. J Institut Eng (India), 87:14–19
8. Chowdhary KR, Bansal VS (2011) Information retrieval using probability and belief theory. International conference emerging trends in networks and computer communications (ETNCC). https://doi.org/10.1109/ETNCC.2011.5958513
9. Egozi O et al (2011).Concept-based information retrieval using explicit semantic analysis. ACM Trans Informat Syst, 29(2):8.1–8.34. https://doi.org/10.1145/1961209.1961211
10. Fung R, DelFavero B (1995) Applying Bayesian networks to information retrieval. Commun ACM 38(3):42–49
11. Miller GA (1995) WordNet: a lexical database for English. Commun ACM 38(11):39–41. https://doi.org/10.1145/219717.219748
12. Grossman DA, Ophir F (2008) Information retrieval-algorithms and heuristics, 2nd edn. Springer
13. Heckerman D et al (1995) Real-world applications of Bayesian networks. Commun ACM 38(3):24–26
14. Recardo BY, Berthier RN (1999) Modern information retrieval. Addison Wesley-ACM Press

15. Rinaldi AM (2009) An ontology-driven approach for semantic information retrieval on the web. Trans Internet Technol 9(3):10:1–10:24. https://doi.org/10.1145/1552291.1552293
16. Smith LC (1976) Artificial intelligence in information retrieval systems. Informat Process Manage 12:189–222. Pergamon Press
17. Wright (1921) Correlation and causation. Agric Res 20:557–585
18. Zobel J, Moffat A (2006) Inverted files for text search engines. ACM Comput Surv 38(2):1–56