# Chapter 2
# Modeling

## 2.1 Introduction

Traditional information retrieval systems usually adopt index terms to index and retrieve documents. In a restricted sense, an index term is a keyword (or group of related words) which has some meaning of its own (i.e., which usually has the semantics of a noun). In its more general form, an index term is simply any word which appears in the text of a document in the collection. Retrieval based on index terms is simple but raises key questions regarding the information retrieval task. For instance, retrieval using index terms adopts as a fundamental foundation the idea that the semantics of the documents and of the user information need can be naturally expressed through sets of index terms. Clearly, this is a considerable oversimplification of the problem because a lot of the semantics in a document or user request is lost when we replace its text with a set of words. Furthermore, matching between each document and the user request is attempted in this very imprecise space of index terms. Thus, it is no surprise that the documents retrieved in response to a user request expressed as a set of keywords are frequently irrelevant. If one also considers that most users have no training in properly forming their queries, the problem is worsened with potentially disastrous results. The frequent dissatisfaction of Web users with the answers they normally obtain is just one good example of this fact.

Clearly, one central problem regarding information retrieval systems is the issue of predicting which documents are relevant and which are not. Such a decision is usually dependent on a ranking algorithm which attempts to establish a simple ordering of the documents retrieved. Documents appearing at the top of this ordering are considered to be more likely to be relevant. Thus, ranking algorithms are at the core of information retrieval systems.

A ranking algorithm operates according to basic premises regarding the notion of document relevance. Distinct sets of premises (regarding document relevance) yield distinct information retrieval models. The IR model adopted determines the predictions of what is relevant and what is not (i.e., the notion of relevance implemented by the system). The purpose of this chapter is to cover the most important information retrieval models proposed over the years. By

doing so, the chapter also provides a conceptual basis for most of the remaining chapters in this book.

We first propose a taxonomy for categorizing the 15 IR models we cover. Second, we distinguish between two types of user retrieval tasks: ad hoc and filtering. Third, we present a formal characterization of IR models which is useful for distinguishing the various components of a particular model. Last, we discuss each of the IR models included in our taxonomy.

## 2.2 A Taxonomy of Information Retrieval Models

The three classic models in information retrieval are called Boolean, vector, and probabilistic. In the Boolean model, documents and queries are represented as sets of index terms. Thus, as suggested in [327], we say that the model is *set theoretic*. In the vector model, documents and queries are represented as vectors in a t-dimensional space. Thus, we say that the model is *algebraic*. In the probabilistic model, the framework for modeling document and query representations is based on probability theory. Thus, as the name indicates, we say that the model is *probabilistic*.

Over the years, alternative modeling paradigms for each type of classic model (i.e., set theoretic, algebraic, and probabilistic) have been proposed. Regarding alternative set theoretic models, we distinguish the fuzzy and the extended Boolean models. Regarding alternative algebraic models, we distinguish the generalized vector, the latent semantic indexing, and the neural network models. Regarding alternative probabilistic models, we distinguish the inference network and the belief network models. Figure 2.1 illustrates a taxonomy of these information retrieval models.

Besides references to the text content, the model might also allow references to the structure normally present in written text. In this case, we say that we have a structured model. We distinguish two models for structured text retrieval namely, the non-overlapping lists model and the proximal nodes model.

As discussed in Chapter 1, the user task might be one of *browsing* (instead of retrieval). In Figure 2.1, we distinguish three models for browsing namely, flat, structure guided, and hypertext.

The organization of this chapter follows the taxonomy of information retrieval models depicted in the figure.We first discuss the three classic models. Second, we discuss the alternative models for each type of classic model. Third, we cover structured text retrieval models. At the end, we discuss models for browsing.

We emphasize that the IR model (Boolean, vector, probabilistic, etc.), the logical view of the documents (full text, set of index terms, etc.), and the user task (retrieval, browsing) are orthogonal aspects of a retrieval system as detailed in Chapter 1. Thus, despite the fact that some models are more appropriate for a certain user task than for another, the same IR model can be used with distinct document logical views to perform different user tasks. Figure 2.2 illustrates the retrieval models most frequently associated with each one of six distinct combinations of a document logical view and a user task.

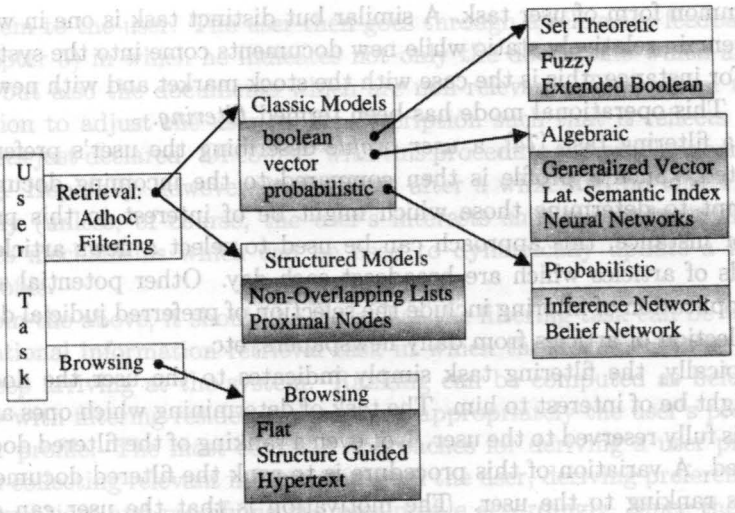


**Figure 2.1** A taxonomy of information retrieval models.

LOGICAL VIEW OF DOCUMENTS

| USER TASK | Index Terms | Full Text | Full Text + Structure |
|---|---|---|---|
| **Retrieval** | Classic<br>Set Theoretic<br>Algebraic<br>Probabilistic | Classic<br>Set Theoretic<br>Algebraic<br>Probabilistic | Structured |
| **Browsing** | Flat | Flat<br>Hypertext | Structure Guided<br>Hypertext |

**Figure 2.2** Retrieval models most frequently associated with distinct combinations of a document logical view and a user task.

## 2.3 Retrieval: Ad hoc and Filtering

In a conventional information retrieval system, the documents in the collection remain relatively static while new queries are submitted to the system. This operational mode has been termed *ad hoc* retrieval in recent years and is the

most common form of user task. A similar but distinct task is one in which the queries remain relatively static while new documents come into the system (and leave). For instance, this is the case with the stock market and with news wiring services. This operational mode has been termed *filtering*.

In a filtering task [74], a *user profile* describing the user's preferences is constructed. Such a profile is then compared to the incoming documents in an attempt to determine those which might be of interest to this particular user. For instance, this approach can be used to select a news article among thousands of articles which are broadcast each day. Other potential scenarios for the application of filtering include the selection of preferred judicial decisions, or the selection of articles from daily newspapers, etc.

Typically, the filtering task simply indicates to the user the documents which might be of interest to him. The task of determining which ones are really relevant is fully reserved to the user. Not even a ranking of the filtered documents is provided. A variation of this procedure is to rank the filtered documents and show this ranking to the user. The motivation is that the user can examine a smaller number of documents if he assumes that the ones at the top of this ranking are more likely to be relevant. This variation of filtering is called *routing* (see Chapter 3) but it is not popular.

Even if no ranking is presented to the user, the filtering task can compute an internal ranking to determine potentially relevant documents. For instance, documents with a ranking above a given threshold could be selected; the others would be discarded. Any IR model can be adopted to rank the documents, but the vector model is usually preferred due to its simplicity. At this point, we observe that filtering is really a type of user task (or operational mode) and not a model of information retrieval. Thus, the task of filtering and the IR model adopted are orthogonal aspects of an IR system.

In a filtering task, the crucial step is not the ranking itself but the construction of a user profile which truly reflects the user's preferences. Many approaches for constructing user profiles have been proposed and here we briefly discuss a couple of them.

A simplistic approach for constructing a user profile is to describe the profile through a set of keywords and to require the user to provide the necessary keywords. The approach is simplistic because it requires the user to do too much. In fact, if the user is not familiar with the service which generates the upcoming documents, he might find it fairly difficult to provide the keywords which appropriately describe his preferences in that context. Furthermore, an attempt by the user to familiarize himself with the vocabulary of the upcoming documents might turn into a tedious and time consuming exercise. Thus, despite its feasibility, requiring the user to precisely describe his profile might be impractical. A more elaborate alternative is to collect information from the user about his preferences and to use this information to build the user profile dynamically. This can be accomplished as follows.

In the very beginning, the user provides a set of keywords which describe an initial (and primitive) profile of his preferences. As new documents arrive, the system uses this profile to select documents which are potentially of interest and

shows them to the user. The user then goes through a relevance feedback cycle (see Chapter 5) in which he indicates not only the documents which are really relevant but also the documents which are non-relevant. The system uses this information to adjust the user profile description such that it reflects the new preferences just declared. Of course, with this procedure the profile is continually changing. Hopefully, however, it stabilizes after a while and no longer changes drastically (unless, of course, the user's interests shift suddenly). Chapter 5 illustrates mechanisms which can be used to dynamically update a keyword-based profile.

From the above, it should be clear that the filtering task can be viewed as a conventional information retrieval task in which the documents are the ones which keep arriving at the system. Ranking can be computed as before. The difficulty with filtering resides in describing appropriately the user's preferences in a user profile. The most common approaches for deriving a user profile are based on collecting relevant information from the user, deriving preferences from this information, and modifying the user profile accordingly. Since the number of potential applications of filtering keeps increasing, we should see in the future a renewed interest in the study and usage of the technique.

## 2.4 A Formal Characterization of IR Models

We have argued that the fundamental premises which form the basis for a ranking algorithm determine the IR model. Throughout this chapter, we will discuss different sets of such premises. However, before doing so, we should state clearly what exactly an IR model is. Our characterization is as follows.

**Definition** *An information retrieval* model *is a quadruple* $[\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j)]$ *where*

*(1)* $\mathbf{D}$ *is a set composed of logical views (or representations) for the documents in the collection.*

*(2)* $\mathbf{Q}$ *is a set composed of logical views (or representations) for the user information needs. Such representations are called* queries.

*(3)* $\mathcal{F}$ *is a framework for modeling document representations, queries, and their relationships.*

*(4)* $R(q_i, d_j)$ *is a ranking function which associates a real number with a query* $q_i \in \mathbf{Q}$ *and a document representation* $d_j \in \mathbf{D}$. *Such ranking defines an ordering among the documents with regard to the query* $q_i$.

To build a model, we think first of representations for the documents and for the user information need. Given these representations, we then conceive the framework in which they can be modeled. This framework should also provide the intuition for constructing a ranking function. For instance, for the classic Boolean model, the framework is composed of sets of documents and the standard operations on sets. For the classic vector model, the framework is composed of a

t-dimensional vectorial space and standard linear algebra operations on vectors. For the classic probabilistic model, the framework is composed of sets, standard probability operations, and the Bayes' theorem.

In the remainder of this chapter, we discuss the various IR models shown in Figure 2.1. Throughout the discussion, we do not explicitly instantiate the components $\mathbf{D}$, $\mathbf{Q}$, $\mathcal{F}$, and $R(q_i, d_j)$ of each model. Such components should be quite clear from the discussion and can be easily inferred.

## 2.5 Classic Information Retrieval

In this section we briefly present the three classic models in information retrieval namely, the Boolean, the vector, and the probabilistic models.

### 2.5.1 Basic Concepts

The classic models in information retrieval consider that each document is described by a set of representative keywords called index terms. An *index term* is simply a (document) word whose semantics helps in remembering the document's main themes. Thus, index terms are used to index and summarize the document contents. In general, index terms are mainly nouns because nouns have meaning by themselves and thus, their semantics is easier to identify and to grasp. Adjectives, adverbs, and connectives are less useful as index terms because they work mainly as complements. However, it might be interesting to consider all the distinct words in a document collection as index terms. For instance, this approach is adopted by some Web search engines as discussed in Chapter 13 (in which case, the document logical view is *full text*). We postpone a discussion on the problem of how to generate index terms until Chapter 7, where the issue is covered in detail.

Given a set of index terms for a document, we notice that not all terms are equally useful for describing the document contents. In fact, there are index terms which are simply vaguer than others. Deciding on the importance of a term for summarizing the contents of a document is not a trivial issue. Despite this difficulty, there are properties of an index term which are easily measured and which are useful for evaluating the potential of a term as such. For instance, consider a collection with a hundred thousand documents. A word which appears in each of the one hundred thousand documents is completely useless as an index term because it does not tell us anything about which documents the user might be interested in. On the other hand, a word which appears in just five documents is quite useful because it narrows down considerably the space of documents which might be of interest to the user. Thus, it should be clear that distinct index terms have varying relevance when used to describe document contents. This effect is captured through the assignment of numerical *weights* to each index term of a document.

Let $k_i$ be an index term, $d_j$ be a document, and $w_{i,j} \geq 0$ be a *weight* associated with the pair $(k_i, d_j)$. This weight quantifies the importance of the index term for describing the document semantic contents.

**Definition** *Let $t$ be the number of index terms in the system and $k_i$ be a generic index term. $K = \{k_1, \ldots, k_t\}$ is the set of all index terms. A weight $w_{i,j} > 0$ is associated with each index term $k_i$ of a document $d_j$. For an index term which does* not *appear in the document text, $w_{i,j} = 0$. With the document $d_j$ is associated an index term vector $\vec{d_j}$ represented by $\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$. Further, let $g_i$ be a function that returns the weight associated with the index term $k_i$ in any $t$-dimensional vector (i.e., $g_i(\vec{d_j}) = w_{i,j}$).*

As we later discuss, the index term weights are usually assumed to be mutually independent. This means that knowing the weight $w_{i,j}$ associated with the pair $(k_i, d_j)$ tells us nothing about the weight $w_{i+1,j}$ associated with the pair $(k_{i+1}, d_j)$. This is clearly a simplification because occurrences of index terms in a document are not uncorrelated. Consider, for instance, that the terms *computer* and *network* are used to index a given document which covers the area of computer networks. Frequently, in this document, the appearance of one of these two words attracts the appearance of the other. Thus, these two words are correlated and their weights could reflect this correlation. While mutual independence seems to be a strong simplification, it does simplify the task of computing index term weights and allows for fast ranking computation. Furthermore, taking advantage of index term correlations for improving the final document ranking is not a simple task. In fact, none of the many approaches proposed in the past has clearly demonstrated that index term correlations are advantageous (for ranking purposes) with general collections. Therefore, unless clearly stated otherwise, we assume mutual independence among index terms. In Chapter 5 we discuss modern retrieval techniques which are based on term correlations and which have been tested successfully with particular collections. These successes seem to be slowly shifting the current understanding towards a more favorable view of the usefulness of term correlations for information retrieval systems.

The above definitions provide support for discussing the three classic information retrieval models, namely, the Boolean, the vector, and the probabilistic models, as we now do.

### 2.5.2 Boolean Model

The Boolean model is a simple retrieval model based on set theory and Boolean algebra. Since the concept of a set is quite intuitive, the Boolean model provides a framework which is easy to grasp by a common user of an IR system. Furthermore, the queries are specified as Boolean expressions which have precise semantics. Given its inherent simplicity and neat formalism, the Boolean model received great attention in past years and was adopted by many of the early commercial bibliographic systems.

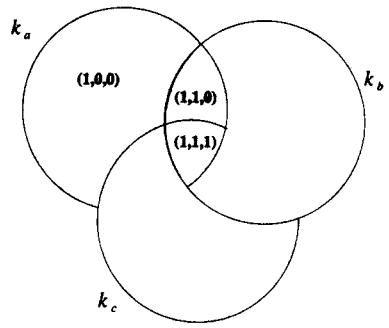


**Figure 2.3** The three conjunctive components for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$.

Unfortunately, the Boolean model suffers from major drawbacks. First, its retrieval strategy is based on a binary decision criterion (i.e., a document is predicted to be either relevant or non-relevant) without any notion of a grading scale, which prevents good retrieval performance. Thus, the Boolean model is in reality much more a data (instead of information) retrieval model. Second, while Boolean expressions have precise semantics, frequently it is not simple to translate an information need into a Boolean expression. In fact, most users find it difficult and awkward to express their query requests in terms of Boolean expressions. The Boolean expressions actually formulated by users often are quite simple (see Chapter 10 for a more thorough discussion on this issue). Despite these drawbacks, the Boolean model is still the dominant model with commercial document database systems and provides a good starting point for those new to the field.

The Boolean model considers that index terms are present or absent in a document. As a result, the index term weights are assumed to be all binary, i.e., $w_{i,j} \in \{0, 1\}$. A query $q$ is composed of index terms linked by three connectives: *not*, *and*, *or*. Thus, a query is essentially a conventional Boolean expression which can be represented as a disjunction of conjunctive vectors (i.e., in *disjunctive normal form* — DNF). For instance, the query $[q = k_a \wedge (k_b \vee \neg k_c)]$ can be written in disjunctive normal form as $[\vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)]$, where each of the components is a binary weighted vector associated with the tuple $(k_a, k_b, k_c)$. These binary weighted vectors are called the conjunctive components of $\vec{q}_{dnf}$. Figure 2.3 illustrates the three conjunctive components for the query $q$.

**Definition** *For the Boolean model, the index term weight variables are all binary i.e., $w_{i,j} \in \{0, 1\}$. A query $q$ is a conventional Boolean expression. Let $\vec{q}_{dnf}$ be the disjunctive normal form for the query $q$. Further, let $\vec{q}_{cc}$ be any of the conjunctive components of $\vec{q}_{dnf}$. The similarity of a document $d_j$ to the query $q$ is defined as*

$$sim(d_j, q) = \begin{cases} 1 & \text{if } \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i},\ g_i(\vec{d_j}) = g_i(\vec{q}_{cc})) \\ 0 & \text{otherwise} \end{cases}$$

*If $sim(d_j, q) = 1$ then the Boolean model predicts that the document $d_j$ is relevant to the query $q$ (it might not be). Otherwise, the prediction is that the document is* not *relevant.*

The Boolean model predicts that each document is either *relevant* or *non-relevant.* There is no notion of a *partial match* to the query conditions. For instance, let $d_j$ be a document for which $\vec{d_j} = (0, 1, 0)$. Document $d_j$ includes the index term $k_b$ but is considered non-relevant to the query $[q = k_a \wedge (k_b \vee \neg k_c)]$.

The main *advantages* of the Boolean model are the clean formalism behind the model and its simplicity. The main *disadvantages* are that exact matching may lead to retrieval of too few or too many documents (see Chapter 10). Today, it is well known that index term weighting can lead to a substantial improvement in retrieval performance. Index term weighting brings us to the vector model.

### 2.5.3 Vector Model

The vector model [697, 695] recognizes that the use of binary weights is too limiting and proposes a framework in which partial matching is possible. This is accomplished by assigning *non-binary* weights to index terms in queries and in documents. These term weights are ultimately used to compute the *degree of similarity* between each document stored in the system and the user query. By sorting the retrieved documents in decreasing order of this degree of similarity, the vector model takes into consideration documents which match the query terms only partially. The main resultant effect is that the ranked document answer set is a lot more precise (in the sense that it better matches the user information need) than the document answer set retrieved by the Boolean model.

**Definition** *For the vector model, the weight $w_{i,j}$ associated with a pair $(k_i, d_j)$ is positive and non-binary. Further, the index terms in the query are also weighted. Let $w_{i,q}$ be the weight associated with the pair $[k_i, q]$, where $w_{i,q} \geq 0$. Then, the query vector $\vec{q}$ is defined as $\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$ where $t$ is the total number of index terms in the system. As before, the vector for a document $d_j$ is represented by $\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$.*

Therefore, a document $d_j$ and a user query $q$ are represented as t-dimensional vectors as shown in Figure 2.4. The vector model proposes to evaluate the degree of similarity of the document $d_j$ with regard to the query $q$ as the correlation between the vectors $\vec{d_j}$ and $\vec{q}$. This correlation can be quantified, for instance, by the *cosine of the angle* between these two vectors. That is,

$$
\begin{aligned}
sim(d_j, q) &= \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} \\
&= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}
\end{aligned}
$$

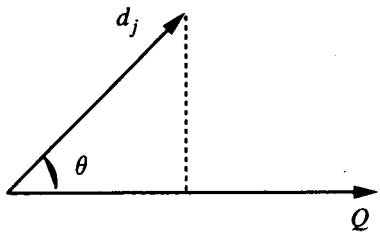


**Figure 2.4** The cosine of $\theta$ is adopted as $sim(d_j, q)$.

where $|\vec{d_j}|$ and $|\vec{q}|$ are the norms of the document and query vectors. The factor $|\vec{q}|$ does not affect the ranking (i.e., the ordering of the documents) because it is the same for all documents. The factor $|\vec{d_j}|$ provides a normalization in the space of the documents.

Since $w_{i,j} \geq 0$ and $w_{i,q} \geq 0$, $sim(q, d_j)$ varies from 0 to +1. Thus, instead of attempting to predict whether a document is relevant or not, the vector model ranks the documents according to their *degree of similarity* to the query. A document might be retrieved even if it matches the query only *partially*. For instance, one can establish a threshold on $sim(d_j, q)$ and retrieve the documents with a degree of similarity above that threshold. But to compute rankings we need first to specify how index term weights are obtained.

Index term weights can be calculated in many different ways. The work by Salton and McGill [698] reviews various term-weighting techniques. Here, we do not discuss them in detail. Instead, we concentrate on elucidating the main idea behind the most effective term-weighting techniques. This idea is related to the basic principles which support clustering techniques, as follows.

Given a collection $C$ of objects and a *vague* description of a set $A$, the goal of a simple clustering algorithm might be to separate the collection $C$ of objects into two sets: a first one which is composed of objects related to the set $A$ and a second one which is composed of objects not related to the set $A$. Vague description here means that we do not have complete information for deciding precisely which objects are and which are not in the set $A$. For instance, one might be looking for a set $A$ of cars which have a price *comparable* to that of a Lexus 400. Since it is not clear what the term *comparable* means exactly, there is not a precise (and unique) description of the set $A$. More sophisticated clustering algorithms might attempt to separate the objects of a collection into various clusters (or classes) according to their properties. For instance, patients of a doctor specializing in cancer could be classified into five classes: terminal, advanced, metastasis, diagnosed, and healthy. Again, the possible class descriptions might be imprecise (and not unique) and the problem is one of deciding to which of these classes a new patient should be assigned. In what follows, however, we only discuss the simpler version of the clustering problem (i.e., the one which considers only two classes) because all that is required is a decision on which documents are predicted to be relevant and which ones are predicted to be not relevant (with regard to a given user query).

To view the IR problem as one of clustering, we refer to the early work of Salton. We think of the documents as a collection $C$ of objects and think of the user query as a (vague) specification of a set $A$ of objects. In this scenario, the IR problem can be reduced to the problem of determining which documents are in the set $A$ and which ones are not (i.e., the IR problem can be viewed as a clustering problem). In a clustering problem, two main issues have to be resolved. First, one needs to determine what are the features which better describe the objects in the set $A$. Second, one needs to determine what are the features which better distinguish the objects in the set $A$ from the remaining objects in the collection $C$. The first set of features provides for quantification of *intra-cluster* similarity, while the second set of features provides for quantification of *inter-cluster* dissimilarity. The most successful clustering algorithms try to balance these two effects.

In the vector model, intra-clustering similarity is quantified by measuring the raw frequency of a term $k_i$ inside a document $d_j$. Such term frequency is usually referred to as the *tf factor* and provides one measure of how well that term describes the document contents (i.e., intra-document characterization). Furthermore, inter-cluster dissimilarity is quantified by measuring the inverse of the frequency of a term $k_i$ among the documents in the collection. This factor is usually referred to as the *inverse document frequency* or the *idf factor*. The motivation for usage of an idf factor is that terms which appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one. As with good clustering algorithms, the most effective term-weighting schemes for IR try to balance these two effects.

**Definition** *Let $N$ be the total number of documents in the system and $n_i$ be the number of documents in which the index term $k_i$ appears. Let $freq_{i,j}$ be the raw frequency of term $k_i$ in the document $d_j$ (i.e., the number of times the term $k_i$ is mentioned in the text of the document $d_j$). Then, the normalized frequency $f_{i,j}$ of term $k_i$ in document $d_j$ is given by*

$$f_{i,j} = \frac{freq_{i,j}}{max_l \; freq_{l,j}} \tag{2.1}$$

*where the maximum is computed over all terms which are mentioned in the text of the document $d_j$. If the term $k_i$ does not appear in the document $d_j$ then $f_{i,j} = 0$. Further, let $idf_i$, inverse document frequency for $k_i$, be given by*

$$idf_i = \log \frac{N}{n_i} \tag{2.2}$$

*The best known term-weighting schemes use weights which are given by*

$$w_{i,j} = f_{i,j} \times \log \frac{N}{n_i} \tag{2.3}$$

*or by a variation of this formula. Such term-weighting strategies are called tf-idf schemes.*

Several variations of the above expression for the weight $w_{i,j}$ are described in an interesting paper by Salton and Buckley which appeared in 1988 [696]. However, in general, the above expression should provide a good weighting scheme for many collections.

For the query term weights, Salton and Buckley suggest

$$w_{i,q} = \left(0.5 + \frac{0.5\ freq_{i,q}}{max_l\ freq_{l,q}}\right) \times \log \frac{N}{n_i} \tag{2.4}$$

where $freq_{i,q}$ is the raw frequency of the term $k_i$ in the text of the information request $q$.

The main *advantages* of the vector model are: (1) its term-weighting scheme improves retrieval performance; (2) its partial matching strategy allows retrieval of documents that *approximate* the query conditions; and (3) its cosine ranking formula sorts the documents according to their degree of similarity to the query. Theoretically, the vector model has the *disadvantage* that index terms are assumed to be mutually independent (equation 2.3 does *not* account for index term dependencies). However, in practice, consideration of term dependencies might be a disadvantage. Due to the locality of many term dependencies, their indiscriminate application to all the documents in the collection might in fact *hurt* the overall performance.

Despite its simplicity, the vector model is a resilient ranking strategy with general collections. It yields ranked answer sets which are difficult to improve upon without query expansion or relevance feedback (see Chapter 5) within the framework of the vector model. A large variety of alternative ranking methods have been compared to the vector model but the consensus seems to be that, in general, the vector model is either superior or almost as good as the known alternatives. Furthermore, it is simple and fast. For these reasons, the vector model is a popular retrieval model nowadays.

### 2.5.4 Probabilistic Model

In this section, we describe the classic probabilistic model introduced in 1976 by Roberston and Sparck Jones [677] which later became known as the *binary independence retrieval* (BIR) model. Our discussion is intentionally brief and focuses mainly on highlighting the key features of the model. With this purpose in mind, we do not detain ourselves in subtleties regarding the binary independence assumption for the model. The section on bibliographic discussion points to references which cover these details.

The probabilistic model attempts to capture the IR problem within a probabilistic framework. The fundamental idea is as follows. Given a user query, there is a set of documents which contains exactly the relevant documents and

no other. Let us refer to this set of documents as the *ideal* answer set. Given the description of this ideal answer set, we would have no problems in retrieving its documents. Thus, we can think of the querying process as a process of specifying the properties of an ideal answer set (which is analogous to interpreting the IR problem as a problem of clustering). The problem is that we do not know exactly what these properties are. All we know is that there are index terms whose semantics should be used to characterize these properties. Since these properties are not known at query time, an effort has to be made at initially guessing what they could be. This initial guess allows us to generate a preliminary probabilistic description of the ideal answer set which is used to retrieve a first set of documents. An interaction with the user is then initiated with the purpose of improving the probabilistic description of the ideal answer set. Such interaction could proceed as follows.

The user takes a look at the retrieved documents and decides which ones are relevant and which ones are not (in truth, only the first top documents need to be examined). The system then uses this information to refine the description of the ideal answer set. By repeating this process many times, it is expected that such a description will evolve and become closer to the real description of the ideal answer set. Thus, one should always have in mind the need to guess at the beginning the description of the ideal answer set. Furthermore, a conscious effort is made to model this description in probabilistic terms.

The probabilistic model is based on the following fundamental assumption.

*Assumption (Probabilistic Principle)* Given a user query $q$ and a document $d_j$ in the collection, the probabilistic model tries to estimate the probability that the user will find the document $d_j$ interesting (i.e., relevant). The model assumes that this probability of relevance depends on the query and the document representations only. Further, the model assumes that there is a subset of all documents which the user prefers as the answer set for the query $q$. Such an *ideal* answer set is labeled $R$ and should maximize the overall probability of relevance to the user. Documents in the set $R$ are predicted to be *relevant* to the query. Documents not in this set are predicted to be *non-relevant.*

This assumption is quite troublesome because it does not state explicitly how to compute the probabilities of relevance. In fact, not even the sample space which is to be used for defining such probabilities is give...

Given a query $q$, the probabilistic model assigns to each document $d_j$, as a measure of its similarity to the query, the ratio $P(d_j \text{ relevant-to } q)/P(d_j \text{ non-relevant-to } q)$ which computes the odds of the document $d_j$ being relevant to the query $q$. Taking the odds of relevance as the rank minimizes the probability of an erroneous judgement [282, 785].

**Definition** *For the probabilistic model, the index term weight variables are all binary i.e.,* $w_{i,j} \in \{0,1\}$, $w_{i,q} \in \{0,1\}$. *A query* $q$ *is a subset of index terms. Let* $R$ *be the set of documents known (or initially guessed) to be relevant. Let* $\overline{R}$ *be the complement of* $R$ *(i.e., the set of non-relevant documents). Let* $P(R|\vec{d}_j)$

*be the probability that the document $d_j$ is relevant to the query $q$ and $P(\overline{R}|\vec{d_j})$ be the probability that $d_j$ is non-relevant to $q$. The similarity $sim(d_j, q)$ of the document $d_j$ to the query $q$ is defined as the ratio*

$$sim(d_j, q) = \frac{P(R|\vec{d_j})}{P(\overline{R}|\vec{d_j})}$$

Using Bayes' rule,

$$sim(d_j, q) = \frac{P(\vec{d_j}|R) \times P(R)}{P(\vec{d_j}|\overline{R}) \times P(\overline{R})}$$

$P(\vec{d_j}|R)$ stands for the probability of randomly selecting the document $d_j$ from the set $R$ of relevant documents. Further, $P(R)$ stands for the probability that a document randomly selected from the entire collection is relevant. The meanings attached to $P(\vec{d_j}|\overline{R})$ and $P(\overline{R})$ are analogous and complementary.

Since $P(R)$ and $P(\overline{R})$ are the same for all the documents in the collection, we write,

$$sim(d_j, q) \sim \frac{P(\vec{d_j}|R)}{P(\vec{d_j}|\overline{R})}$$

Assuming independence of index terms,

$$sim(d_j, q) \sim \frac{(\prod_{g_i(\vec{d_j})=1} P(k_i|R)) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k}_i|R))}{(\prod_{g_i(\vec{d_j})=1} P(k_i|\overline{R})) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k}_i|\overline{R}))}$$

$P(k_i|R)$ stands for the probability that the index term $k_i$ is present in a document randomly selected from the set $R$. $P(\overline{k}_i|R)$ stands for the probability that the index term $k_i$ is not present in a document randomly selected from the set $R$. The probabilities associated with the set $\overline{R}$ have meanings which are analogous to the ones just described.

Taking logarithms, recalling that $P(k_i|R) + P(\overline{k}_i|R) = 1$, and ignoring factors which are constant for all documents in the context of the same query, we can finally write

$$sim(d_j, q) \sim \sum_{i=1}^{t} w_{i,q} \times w_{i,j} \times \left( \log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right)$$

which is a key expression for ranking computation in the probabilistic model.

Since we do not know the set $R$ at the beginning, it is necessary to devise a method for initially computing the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$. There are many alternatives for such computation. We discuss a couple of them below.

In the very beginning (i.e., immediately after the query specification), there are no retrieved documents. Thus, one has to make simplifying assumptions such as: (a) assume that $P(k_i|R)$ is constant for all index terms $k_i$ (typically, equal to 0.5) and (b) assume that the distribution of index terms among the non-relevant documents can be approximated by the distribution of index terms among all the documents in the collection. These two assumptions yield

$$\begin{aligned} P(k_i|R) &= 0.5 \\ P(k_i|\overline{R}) &= \frac{n_i}{N} \end{aligned}$$

where, as already defined, $n_i$ is the number of documents which contain the index term $k_i$ and $N$ is the total number of documents in the collection. Given this initial guess, we can then retrieve documents which contain query terms and provide an initial probabilistic ranking for them. After that, this initial ranking is improved as follows.

Let $V$ be a subset of the documents initially retrieved and ranked by the probabilistic model. Such a subset can be defined, for instance, as the top $r$ ranked documents where $r$ is a previously defined threshold. Further, let $V_i$ be the subset of $V$ composed of the documents in $V$ which contain the index term $k_i$. For simplicity, we also use $V$ and $V_i$ to refer to the number of elements in these sets (it should always be clear when the used variable refers to the set or to the number of elements in it). For improving the probabilistic ranking, we need to improve our guesses for $P(k_i|R)$ and $P(k_i|\overline{R})$. This can be accomplished with the following assumptions: (a) we can approximate $P(k_i|R)$ by the distribution of the index term $k_i$ among the documents retrieved so far, and (b) we can approximate $P(k_i|\overline{R})$ by considering that all the non-retrieved documents are not relevant. With these assumptions, we can write,

$$\begin{aligned} P(k_i|R) &= \frac{V_i}{V} \\ P(k_i|\overline{R}) &= \frac{n_i - V_i}{N - V} \end{aligned}$$

This process can then be repeated recursively. By doing so, we are able to improve on our guesses for the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ without any assistance from a human subject (contrary to the original idea). However, we can also use assistance from the user for definition of the subset $V$ as originally conceived.

The last formulas for $P(k_i|R)$ and $P(k_i|\overline{R})$ pose problems for small values of $V$ and $V_i$ which arise in practice (such as $V = 1$ and $V_i = 0$). To circumvent these problems, an adjustment factor is often added in which yields

$$\begin{aligned} P(k_i|R) &= \frac{V_i + 0.5}{V + 1} \\ P(k_i|\overline{R}) &= \frac{n_i - V_i + 0.5}{N - V + 1} \end{aligned}$$

An adjustment factor which is constant and equal to 0.5 is not always satisfactory. An alternative is to take the fraction $n_i/N$ as the adjustment factor which yields

$$P(k_i|R) = \frac{V_i + \frac{n_i}{N}}{V+1}$$
$$P(k_i|\overline{R}) = \frac{n_i - V_i + \frac{n_i}{N}}{N-V+1}$$

This completes our discussion of the probabilistic model.

The main *advantage* of the probabilistic model, in theory, is that documents are ranked in decreasing order of their *probability* of being relevant. The *disadvantages* include: (1) the need to guess the initial separation of documents into relevant and non-relevant sets; (2) the fact that the method does *not* take into account the frequency with which an index term occurs inside a document (i.e., all weights are binary); and (3) the adoption of the independence assumption for index terms. However, as discussed for the vector model, it is not clear that independence of index terms is a bad assumption in practical situations.

### 2.5.5 Brief Comparison of Classic Models

In general, the Boolean model is considered to be the weakest classic method. Its main problem is the inability to recognize partial matches which frequently leads to poor performance. There is some controversy as to whether the probabilistic model outperforms the vector model. Croft performed some experiments and suggested that the probabilistic model provides a better retrieval performance. However, experiments done afterwards by Salton and Buckley refute that claim. Through several different measures, Salton and Buckley showed that the vector model is *expected* to outperform the probabilistic model with general collections. This also seems to be the dominant thought among researchers, practitioners, and the Web community, where the popularity of the vector model runs high.

## 2.6 Alternative Set Theoretic Models

In this section, we discuss two alternative set theoretic models, namely the fuzzy set model and the extended Boolean model.

### 2.6.1 Fuzzy Set Model

Representing documents and queries through sets of keywords yields descriptions which are only partially related to the real semantic contents of the respective documents and queries. As a result, the matching of a document to the query terms is approximate (or vague). This can be modeled by considering that each

query term defines a *fuzzy* set and that each document has a *degree of membership* (usually smaller than 1) in this set. This interpretation of the retrieval process (in terms of concepts from fuzzy theory) is the basic foundation of the various fuzzy set models for information retrieval which have been proposed over the years. Instead of reviewing several of these models here, we focus on a particular one whose description fits well with the models already covered in this chapter. Thus, our discussion is based on the fuzzy set model for information retrieval proposed by Ogawa, Morita, and Kobayashi [616]. Before proceeding, we briefly introduce some fundamental concepts.

### Fuzzy Set Theory

*Fuzzy set theory* [846] deals with the representation of classes whose boundaries are not well defined. The key idea is to associate a membership function with the elements of the class. This function takes values in the interval [0,1] with 0 corresponding to no membership in the class and 1 corresponding to full membership. Membership values between 0 and 1 indicate *marginal* elements of the class. Thus, membership in a fuzzy set is a notion intrinsically *gradual* instead of abrupt (as in conventional Boolean logic).

**Definition** *A fuzzy subset A of a universe of discourse U is characterized by a membership function* $\mu_A : U \rightarrow [0,1]$ *which associates with each element u of U a number* $\mu_A(u)$ *in the interval [0,1].*

The three most commonly used operations on fuzzy sets are: the *complement* of a fuzzy set, the *union* of two or more fuzzy sets, and the *intersection* of two or more fuzzy sets. They are defined as follows.

**Definition** *Let U be the universe of discourse, A and B be two fuzzy subsets of U, and* $\overline{A}$ *be the complement of A relative to U. Also, let u be an element of U. Then,*

$$\begin{aligned} \mu_{\overline{A}}(u) &= 1 - \mu_A(u) \\ \mu_{A \cup B}(u) &= max(\mu_A(u), \mu_B(u)) \\ \mu_{A \cap B}(u) &= min(\mu_A(u), \mu_B(u)) \end{aligned}$$

Fuzzy sets are useful for representing vagueness and imprecision and have been applied to various domains. In what follows, we discuss their application to information retrieval.

### Fuzzy Information Retrieval

As discussed in Chapters 5 and 7, one additional approach to modeling the information retrieval process is to adopt a thesaurus (which defines term re-

lationships). The basic idea is to expand the set of index terms in the query with related terms (obtained from the thesaurus) such that additional relevant documents (i.e., besides the ones which would be normally retrieved) can be retrieved by the user query. A thesaurus can also be used to model the information retrieval problem in terms of fuzzy sets as follows.

A thesaurus can be constructed by defining a *term-term correlation matrix* $\vec{c}$ (called *keyword connection matrix* in [616]) whose rows and columns are associated to the index terms in the document collection. In this matrix $\vec{c}$, a normalized correlation factor $c_{i,l}$ between two terms $k_i$ and $k_l$ can be defined by

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

where $n_i$ is the number of documents which contain the term $k_i$, $n_l$ is the number of documents which contain the term $k_l$, and $n_{i,l}$ is the number of documents which contain both terms. Such a correlation metric is quite common and has been used extensively with clustering algorithms as detailed in Chapter 5.

We can use the term correlation matrix $\vec{c}$ to define a fuzzy set associated to each index term $k_i$. In this fuzzy set, a document $d_j$ has a degree of membership $\mu_{i,j}$ computed as

$$\mu_{i,j} = 1 - \prod_{k_l \in d_j} (1 - c_{i,l})$$

which computes an algebraic sum (here implemented as the complement of a negated algebraic product) over all terms in the document $d_j$. A document $d_j$ belongs to the fuzzy set associated to the term $k_i$ if its own terms are related to $k_i$. Whenever there is at least one index term $k_l$ of $d_j$ which is strongly related to the index $k_i$ (i.e., $c_{i,l} \sim 1$), then $\mu_{i,j} \sim 1$ and the index $k_i$ is a good fuzzy index for the document $d_j$. In the case when all index terms of $d_j$ are only loosely related to $k_i$, the index $k_i$ is not a good fuzzy index for $d_j$ (i.e., $\mu_{i,j} \sim 0$). The adoption of an algebraic sum over all terms in the document $d_j$ (instead of the classic *max* function) allows a smooth transition for the values of the $\mu_{i,j}$ factor.

The user states his information need by providing a Boolean-like query expression. As also happens with the classic Boolean model (see the beginning of this chapter), this query is converted to its disjunctive normal form. For instance, the query $[q = k_a \wedge (k_b \vee \neg k_c)]$ can be written in disjunctive normal form as $[\vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)]$, where each of the components is a binary weighted vector associated to the tuple $(k_a, k_b, k_c)$. These binary weighted vectors are the conjunctive components of $\vec{q}_{dnf}$. Let $cc_i$ be a reference to the $i$-th conjunctive component. Then,

$$\vec{q}_{dnf} = cc_1 \vee cc_2 \vee \ldots \vee cc_p$$

where $p$ is the number of conjunctive components of $\vec{q}_{dnf}$. The procedure to compute the documents relevant to a query is analogous to the procedure adopted

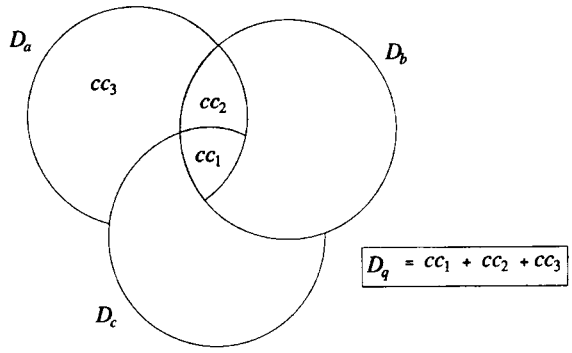


**Figure 2.5** Fuzzy document sets for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$. Each $cc_i$, $i \in \{1, 2, 3\}$, is a conjunctive component. $D_q$ is the query fuzzy set.

by the classic Boolean model. The difference is that here we deal with fuzzy (instead of *crispy* or Boolean) sets. We proceed with an example.

Consider again the query $[q = k_a \wedge (k_b \vee \neg k_c)]$. Let $D_a$ be the fuzzy set of documents associated to the index $k_a$. This set is composed, for instance, by the documents $d_j$ which have a degree of membership $\mu_{a,j}$ greater than a predefined threshold $K$. Further, let $\overline{D}_a$ be the complement of the set $D_a$. The fuzzy set $\overline{D}_a$ is associated to $\overline{k}_a$, the negation of the index term $k_a$. Analogously, we can define fuzzy sets $D_b$ and $D_c$ associated to the index terms $k_b$ and $k_c$, respectively. Figure 2.5 illustrates this example. Since the sets are all fuzzy, a document $d_j$ might belong to the set $D_a$, for instance, even if the text of the document $d_j$ does not mention the index $k_a$.

The query fuzzy set $D_q$ is a union of the fuzzy sets associated with the three conjunctive components of $\vec{q}_{dnf}$ (which are referred to as $cc_1$, $cc_2$, and $cc_3$). The membership $\mu_{q,j}$ of a document $d_j$ in the fuzzy answer set $D_q$ is computed as follows.

$$\begin{aligned}
\mu_{q,j} &= \mu_{cc_1+cc_2+cc_3,j} \\
&= 1 - \prod_{i=1}^{3}(1 - \mu_{cc_i,j}) \\
&= 1 - (1 - \mu_{a,j}\mu_{b,j}\mu_{c,j}) \times \\
&\quad (1 - \mu_{a,j}\mu_{b,j}(1 - \mu_{c,j})) \times (1 - \mu_{a,j}(1 - \mu_{b,j})(1 - \mu_{c,j}))
\end{aligned}$$

where $\mu_{i,j}$, $i \in \{a, b, c\}$, is the membership of $d_j$ in the fuzzy set associated with $k_i$.

As already observed, the degree of membership in a disjunctive fuzzy set is computed here using an *algebraic sum*, instead of the more common *max* function. Further, the degree of membership in a conjunctive fuzzy set is computed here using an *algebraic product*, instead of the more common *min* function. This adoption of algebraic sums and products yields degrees of membership which

vary more smoothly than those computed using the *min* and *max* functions and thus seem more appropriate to an information retrieval system.

This example illustrates how this fuzzy model ranks documents relative to the user query. The model uses a term-term correlation matrix to compute correlations between a document $d_j$ and its fuzzy index terms. Further, the model adopts algebraic sums and products (instead of *max* and *min*) to compute the overall degree of membership of a document $d_j$ in the fuzzy set defined by the user query. Ogawa, Morita, and Kobayashi [616] also discuss how to incorporate user relevance feedback into the model but such discussion is beyond the scope of this chapter.

Fuzzy set models for information retrieval have been discussed mainly in the literature dedicated to fuzzy theory and are not popular among the information retrieval community. Further, the vast majority of the experiments with fuzzy set models has considered only small collections which make comparisons difficult to make at this time.

### 2.6.2 Extended Boolean Model

Boolean retrieval is simple and elegant. However, since there is no provision for term weighting, no ranking of the answer set is generated. As a result, the size of the output might be too large or too small (see Chapter 10 for details on this issue). Because of these problems, modern information retrieval systems are no longer based on the Boolean model. In fact, most of the new systems adopt at their core some form of vector retrieval. The reasons are that the vector space model is simple, fast, and yields better retrieval performance. One alternative approach though is to extend the Boolean model with the functionality of partial matching and term weighting. This strategy allows one to combine Boolean query formulations with characteristics of the vector model. In what follows, we discuss one of the various models which are based on the idea of extending the Boolean model with features of the vector model.

The *extended Boolean model*, introduced in 1983 by Salton, Fox, and Wu [703], is based on a critique of a basic assumption in Boolean logic as follows. Consider a conjunctive Boolean query given by $q = k_x \wedge k_y$. According to the Boolean model, a document which contains either the term $k_x$ or the term $k_y$ is as irrelevant as another document which contains neither of them. However, this binary decision criteria frequently is not in accordance with common sense. An analogous reasoning applies when one considers purely disjunctive queries.

When only two terms are considered, we can plot queries and documents in a two-dimensional map as shown in Figure 2.6. A document $d_j$ is positioned in this space through the adoption of weights $w_{x,j}$ and $w_{y,j}$ associated with the pairs $[k_x, d_j]$ and $[k_y, d_j]$, respectively. We assume that these weights are normalized and thus lie between 0 and 1. For instance, these weights can be computed as normalized tf-idf factors as follows.

$$w_{x,j} = f_{x,j} \times \frac{idf_x}{max_i \ idf_i}$$

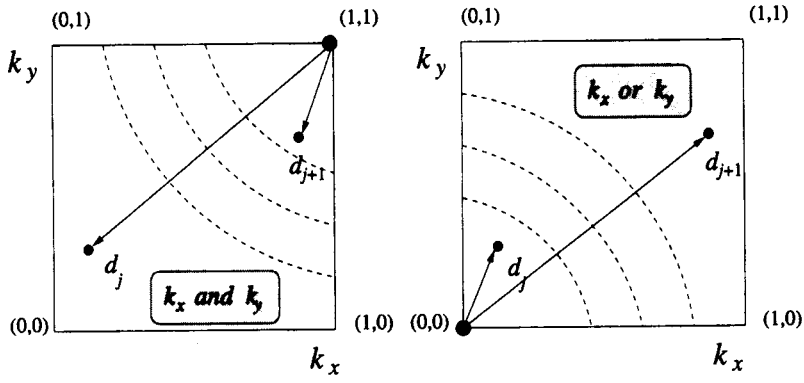


**Figure 2.6** Extended Boolean logic considering the space composed of two terms $k_x$ and $k_y$ only.

where, as defined by equation 2.3, $f_{x,j}$ is the normalized frequency of term $k_x$ in document $d_j$ and $idf_i$ is the inverse document frequency for a generic term $k_i$. For simplicity, in the remainder of this section, we refer to the weight $w_{x,j}$ as $x$, to the weight $w_{y,j}$ as $y$, and to the document vector $\vec{d_j} = (w_{x,j}, w_{y,j})$ as the point $d_j = (x, y)$. Observing Figure 2.6 we notice two particularities. First, for a disjunctive query $q_{or} = k_x \vee k_y$, the point $(0,0)$ is the spot to be avoided. This suggests taking the distance from $(0,0)$ as a measure of similarity with regard to the query $q_{or}$. Second, for a conjunctive query $q_{and} = k_x \wedge k_y$, the point $(1,1)$ is the most desirable spot. This suggests taking the complement of the distance from the point $(1,1)$ as a measure of similarity with regard to the query $q_{and}$. Furthermore, such distances can be normalized which yields,

$$sim(q_{or}, d) = \sqrt{\frac{x^2 + y^2}{2}}$$

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}$$

If the weights are all Boolean (i.e., $w_{x,j} \in \{0,1\}$), a document is always positioned in one of the four corners (i.e., (0,0), (0,1), (1,0), or (1,1)) and the values for $sim(q_{or}, d)$ are restricted to 0, $1/\sqrt{2}$, and 1. Analogously, the values for $sim(q_{and}, d)$ are restricted to 0, $1 - 1/\sqrt{2}$, and 1.

Given that the number of index terms in a document collection is $t$, the Boolean model discussed above can be naturally extended to consider Euclidean distances in a t-dimensional space. However, a more comprehensive generalization is to adopt the theory of vector norms as follows.

The *p-norm model* generalizes the notion of distance to include not only Euclidean distances but also p-distances, where $1 \leq p \leq \infty$ is a newly introduced parameter whose value must be specified at query time. A generalized disjunctive

query is now represented by

$$q_{or} = k_1 \vee^p k_2 \vee^p \ldots \vee^p k_m$$

Analogously, a generalized conjunctive query is now represented by

$$q_{and} = k_1 \wedge^p k_2 \wedge^p \ldots \wedge^p k_m$$

The respective query-document similarities are now given by

$$sim(q_{or}, d_j) = \left( \frac{x_1^p + x_2^p + \ldots + x_m^p}{m} \right)^{\frac{1}{p}}$$

$$sim(q_{and}, d_j) = 1 - \left( \frac{(1 - x_1)^p + (1 - x_2)^p + \ldots + (1 - x_m)^p}{m} \right)^{\frac{1}{p}}$$

where each $x_i$ stands for the weight $w_{i,d}$ associated to the pair $[k_i, d_j]$.

The p norm as defined above enjoys a couple of interesting properties as follows. First, when $p = 1$ it can be verified that

$$sim(q_{or}, d_j) = sim(q_{and}, d_j) = \frac{x_1 + \ldots + x_m}{m}$$

Second, when $p = \infty$ it can be verified that

$$sim(q_{or}, d_j) = max(x_i)$$
$$sim(q_{and}, d_j) = min(x_i)$$

Thus, for $p = 1$, conjunctive and disjunctive queries are evaluated by a sum of term-document weights as done by vector-based similarity formulas (which compute the inner product). Further, for $p = \infty$, queries are evaluated according to the formalism of fuzzy logic (which we view as a generalization of Boolean logic). By varying the parameter $p$ between 1 and infinity, we can vary the p-norm ranking behavior from that of a vector-like ranking to that of a Boolean-like ranking. This is quite powerful and is a good argument in favor of the extended Boolean model.

The processing of more general queries is done by grouping the operators in a predefined order. For instance, consider the query $q = (k_1 \wedge^p k_2) \vee^p k_3$. The similarity $sim(q, d_j)$ between a document $d_j$ and this query is then computed as

$$sim(q, d) = \left( \frac{\left(1 - \left(\frac{(1-x_1)^p + (1-x_2)^p}{2}\right)^{\frac{1}{p}}\right)^p + x_3^p}{2} \right)^{\frac{1}{p}}$$

This procedure can be applied recursively no matter the number of AND/OR operators.

One additional interesting aspect of this extended Boolean model is the possibility of using combinations of different values of the parameter $p$ in a same query request. For instance, the query

$$(k_1 \vee^2 k_2) \wedge^{\infty} k_3$$

could be used to indicate that $k_1$ and $k_2$ should be used as in a vector system but that the presence of $k_3$ is required (i.e., the conjunction is interpreted as a Boolean operation). Despite the fact that it is not clear whether this additional functionality has any practical impact, the model does allow for it and does so in a natural way (without the need for clumsy extensions to handle special cases).

We should also observe that the extended Boolean model relaxes Boolean algebra interpreting Boolean operations in terms of algebraic distances. In this sense, it is really a hybrid model which includes properties of both the set theoretic models and the algebraic models. For simplicity, we opted for classifying the model as a set theoretic one.

The extended Boolean model was introduced in 1983 but has not been used extensively. However, the model does provide a neat framework and might reveal itself useful in the future.

## 2.7 Alternative Algebraic Models

In this section, we discuss three alternative algebraic models namely, the generalized vector space model, the latent semantic indexing model, and the neural network model.

### 2.7.1 Generalized Vector Space Model

As already discussed, the three classic models assume independence of index terms. For the vector model, this assumption is normally interpreted as follows.

**Definition** *Let $\vec{k}_i$ be a vector associated with the index term $k_i$. Independence of index terms in the vector model implies that the set of vectors $\{\vec{k}_1, \vec{k}_2, \ldots, \vec{k}_t\}$ is linearly independent and forms a basis for the subspace of interest. The dimension of this space is the number $t$ of index terms in the collection.*

Frequently, independence among index terms is interpreted in a more restrictive sense to mean pairwise orthogonality among the index term vectors i.e., to mean that for each pair of index term vectors $\vec{k}_i$ and $\vec{k}_j$ we have $\vec{k}_i \bullet \vec{k}_j = 0$. In 1985, however, Wong, Ziarko, and Wong [832] proposed an interpretation in which the index term vectors are assumed linearly independent but are not pairwise orthogonal. Such interpretation leads to the *generalized vector space model* which we now discuss.

In the generalized vector space model, two index term vectors might be non-orthogonal. This means that index term vectors are not seen as the orthogonal vectors which compose the basis of the space. Instead, they are themselves composed of *smaller* components which are derived from the particular collection at hand as follows.

**Definition** *Given the set* $\{k_1, k_2, \ldots, k_t\}$ *of index terms in a collection, as before, let* $w_{i,j}$ *be the weight associated with the term-document pair* $[k_i, d_j]$. *If the* $w_{i,j}$ *weights are all binary then all possible patterns of term co-occurrence (inside documents) can be represented by a set of* $2^t$ minterms *given by* $m_1 = (0, 0, \ldots, 0)$, $m_2 = (1, 0, \ldots, 0)$, $\ldots$, $m_{2^t} = (1, 1, \ldots, 1)$. *Let* $g_i(m_j)$ *return the weight* $\{0,1\}$ *of the index term* $k_i$ *in the minterm* $m_j$.

Thus, the minterm $m_1$ (for which $g_i(m_1) = 0$, for all $i$) points to the documents containing none of the index terms. The minterm $m_2$ (for which $g_1(m_2) = 1$, for $i = 1$, and $g_i(m_2) = 0$, for $i > 1$) points to the documents containing solely the index term $k_1$. Further, the minterm $m_{2^t}$ points to the documents containing all the index terms. The central idea in the generalized vector space model is to introduce a set of pairwise orthogonal vectors $\vec{m}_i$ associated with the set of minterms and to adopt this set of vectors as the basis for the subspace of interest.

**Definition** *Let us define the following set of* $\vec{m}_i$ *vectors*

$$\begin{aligned} \vec{m}_1 &= (1, 0, \ldots, 0, 0) \\ \vec{m}_2 &= (0, 1, \ldots, 0, 0) \\ &\vdots \\ \vec{m}_{2^t} &= (0, 0, \ldots, 0, 1) \end{aligned}$$

*where each vector* $\vec{m}_i$ *is associated with the respective minterm* $m_i$.

Notice that $\vec{m}_i \bullet \vec{m}_j = 0$ for all $i \neq j$ and thus the set of $\vec{m}_i$ vectors is, by definition, *pairwise* orthogonal. This set of $\vec{m}_i$ vectors is then taken as the orthonormal basis for the generalized vector space model.

Pairwise orthogonality among the $\vec{m}_i$ vectors does not imply independence among the index terms. On the contrary, index terms are now correlated by the $\vec{m}_i$ vectors. For instance, the vector $\vec{m}_4$ is associated with the minterm $m_4 = (1, 1, \ldots, 0)$ which points to the documents in the collection containing the index terms $k_1$, $k_2$, and no others. If such documents do exist in the collection under consideration then we say that the minterm $m_4$ is *active* and that a dependence between the index terms $k_1$ and $k_2$ is induced. If we consider this point more carefully, we notice that the generalized vector model adopts as a basic foundation the idea that co-occurrence of index terms inside documents in the collection induces dependencies among these index terms. Since this is an idea which was introduced many years before the generalized vector space

model itself, novelty is not granted. Instead, the main contribution of the model is the establishment of a formal framework in which dependencies among index terms (induced by co-occurrence patterns inside documents) can be nicely represented.

The usage of index term dependencies to improve retrieval performance continues to be a controversial issue. In fact, despite the introduction in the 1980s of more effective algorithms for incorporating term dependencies (see Chapter 5), there is no consensus that incorporation of term dependencies in the model yields effective improvement with general collections. Thus, it is not clear that the framework of the generalized vector model provides a clear advantage in practical situations. Further, the generalized vector model is more complex and computationally more expensive than the classic vector model.

To determine the index term vector $\vec{k}_i$ associated with the index term $k_i$, we simply sum up the vectors for all minterms $m_r$ in which the term $k_i$ is in state 1 and normalize. Thus,

$$\vec{k}_i = \frac{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}\ \vec{m}_r}{\sqrt{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}^2}} \tag{2.5}$$

$$c_{i,r} = \sum_{d_j \mid g_l(\vec{d}_j)=g_l(m_r)\ for\ all\ l} w_{i,j}$$

These equations provide a general definition for the index term vector $\vec{k}_i$ in terms of the $\vec{m}_r$ vectors. The term vector $\vec{k}_i$ collects all the $\vec{m}_r$ vectors in which the index term $k_i$ is in state 1. For each $\vec{m}_r$ vector, a correlation factor $c_{i,r}$ is defined. Such a correlation factor sums up the weights $w_{i,j}$ associated with the index term $k_i$ and each document $d_j$ whose term occurrence pattern coincides exactly with that of the minterm $m_r$. Thus, a minterm is of interest (in which case it is said to be *active*) only if there is at least one document in the collection which matches its term occurrence pattern. This implies that no more than $N$ minterms can be active, where $N$ is the number of documents in the collection. Therefore, the ranking computation does not depend on an exponential number of minterms as equation 2.5 seems to suggest.

Notice that the internal product $\vec{k}_i \bullet \vec{k}_j$ can now be used to quantify a degree of correlation between the index terms $k_i$ and $k_j$. For instance,

$$\vec{k}_i \bullet \vec{k}_j = \sum_{\forall r \mid g_i(m_r)=1 \wedge g_j(m_r)=1} c_{i,r} \times c_{j,r}$$

which, as later discussed in Chapter 5, is a good technique for quantifying index term correlations.

In the classic vector model, a document $d_j$ and a user query $q$ are expressed by $\vec{d}_j = \sum_{\forall i} w_{i,j}\ \vec{k}_i$ and $\vec{q} = \sum_{\forall i} w_{i,q}\ \vec{k}_i$, respectively. In the generalized vector space model, these representations can be directly translated to the space of minterm vectors $\vec{m}_r$ by applying equation 2.5. The resultant $\vec{d}_j$ and $\vec{q}$ vectors

are then used for computing the ranking through a standard cosine similarity function.

The ranking that results from the generalized vector space model combines the standard $w_{i,j}$ term-document weights with the correlation factors $c_{i,r}$. However, since the usage of term-term correlations does not necessarily yield improved retrieval performance, it is not clear in which situations the generalized model outperforms the classic vector model. Furthermore, the cost of computing the ranking in the generalized model can be fairly high with large collections because, in this case, the number of active minterms (i.e., those which have to be considered for computing the $\vec{k}_i$ vectors) might be proportional to the number of documents in the collection. Despite these drawbacks, the generalized vector model does introduce new ideas which are of importance from a theoretical point of view.

### 2.7.2 Latent Semantic Indexing Model

As discussed earlier, summarizing the contents of documents and queries through a set of index terms can lead to poor retrieval performance due to two effects. First, many unrelated documents might be included in the answer set. Second, relevant documents which are not indexed by any of the query keywords are not retrieved. The main reason for these two effects is the inherent vagueness associated with a retrieval process which is based on keyword sets.

The ideas in a text are more related to the concepts described in it than to the index terms used in its description. Thus, the process of matching documents to a given query could be based on concept matching instead of index term matching. This would allow the retrieval of documents even when they are not indexed by query index terms. For instance, a document could be retrieved because it shares concepts with another document which is relevant to the given query. Latent semantic indexing is an approach introduced in 1988 which addresses these issues (for clustering-based approaches which also address these issues, see Chapter 5).

The main idea in the *latent semantic indexing model* [287] is to map each document and query vector into a lower dimensional space which is associated with concepts. This is accomplished by mapping the index term vectors into this lower dimensional space. The claim is that retrieval in the reduced space may be superior to retrieval in the space of index terms. Before proceeding, let us define basic terminology.

**Definition** *As before, let t be the number of index terms in the collection and N be the total number of documents. Define* $\vec{M}=(M_{ij})$ *as a term-document association matrix with t rows and N columns. To each element* $M_{ij}$ *of this matrix is assigned a weight* $w_{i,j}$ *associated with the term-document pair* $[k_i, d_j]$. *This* $w_{i,j}$ *weight could be generated using the tf-idf weighting technique common in the classic vector space model.*

Latent semantic indexing proposes to decompose the $\vec{M}$ association matrix in three components using singular value decomposition as follows.

$$\vec{M} = \vec{K}\vec{S}\vec{D}^t$$

The matrix $\vec{K}$ is the matrix of eigenvectors derived from the term-to-term correlation matrix given by $\vec{M}\vec{M}^t$ (see Chapter 5). The matrix $\vec{D}^t$ is the matrix of eigenvectors derived from the transpose of the document-to-document matrix given by $\vec{M}^t\vec{M}$. The matrix $\vec{S}$ is an $r \times r$ diagonal matrix of singular values where $r = min(t, N)$ is the *rank* of $\vec{M}$.

Consider now that only the $s$ largest singular values of $\vec{S}$ are kept along with their corresponding columns in $\vec{K}$ and $\vec{D}^t$ (i.e., the remaining singular values of $\vec{S}$ are deleted). The resultant $\vec{M}_s$ matrix is the matrix of rank $s$ which is closest to the original matrix $\vec{M}$ in the least square sense. This matrix is given by

$$\vec{M}_s = \vec{K}_s\vec{S}_s\vec{D}_s^t$$

where $s$, $s < r$, is the dimensionality of a reduced concept space. The selection of a value for $s$ attempts to balance two opposing effects. First, $s$ should be large enough to allow fitting all the structure in the real data. Second, $s$ should be small enough to allow filtering out all the non-relevant representational details (which are present in the conventional index-term based representation).

The relationship between any two documents in the reduced space of dimensionality $s$ can be obtained from the $\vec{M}_s^t\vec{M}_s$ matrix given by

$$\begin{aligned}
\vec{M}_s^t\vec{M}_s &= (\vec{K}_s\vec{S}_s\vec{D}_s^t)^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{K}_s^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{S}_s\vec{D}_s^t \\
&= (\vec{D}_s\vec{S}_s)(\vec{D}_s\vec{S}_s)^t
\end{aligned}$$

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

To rank documents with regard to a given user query, we simply model the query as a *pseudo-document* in the original $\vec{M}$ term-document matrix. Assume the query is modeled as the document with number 0. Then, the first row in the matrix $\vec{M}_s^t\vec{M}_s$ provides the ranks of all documents with respect to this query.

Since the matrices used in the latent semantic indexing model are of rank $s$, $s << t$, and $s << N$, they form an efficient indexing scheme for the documents in the collection. Further, they provide for elimination of noise (present in index term-based representations) and removal of redundancy.

The latent semantic indexing model introduces an interesting conceptualization of the information retrieval problem based on the theory of singular value decomposition. Thus, it has its value as a new theoretical framework. Whether it is superior in practical situations with general collections remains to be verified.

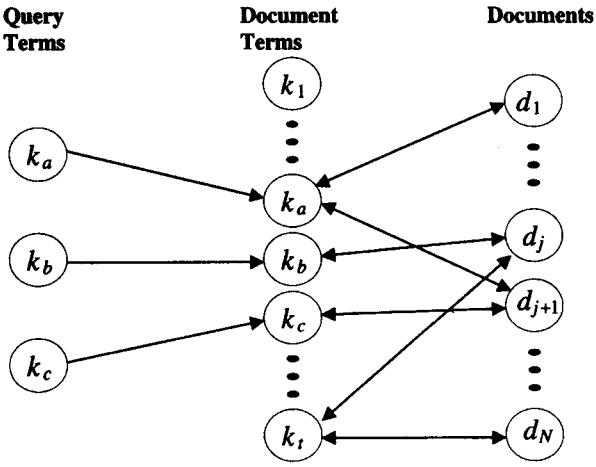


**Figure 2.7** A neural network model for information retrieval.

### 2.7.3 Neural Network Model

In an information retrieval system, document vectors are compared with query vectors for the computation of a ranking. Thus, index terms in documents and queries have to be matched and weighted for computing this ranking. Since neural networks are known to be good pattern matchers, it is natural to consider their usage as an alternative model for information retrieval.

It is now well established that our brain is composed of billions of neurons. Each *neuron* can be viewed as a basic processing unit which, when stimulated by input signals, might emit output signals as a reactive action. The signals emitted by a neuron are fed into other neurons (through synaptic connections) which can themselves emit new output signals. This process might repeat itself through several layers of neurons and is usually referred to as a *spread activation* process. As a result, input information is processed (i.e., analyzed and interpreted) which might lead the brain to command physical reactions (e.g., motor actions) in response.

A *neural network* is an oversimplified graph representation of the mesh of interconnected neurons in a human brain. The nodes in this graph are the processing units while the edges play the role of the synaptic connections. To simulate the fact that the strength of a synaptic connection in the human brain changes over time, a *weight* is assigned to each edge of our neural network. At each instant, the state of a node is defined by its *activation level* (which is a function of its initial state and of the signals it receives as input). Depending on its activation level, a node $A$ might send a signal to a neighbor node $B$. The strength of this signal at the node $B$ depends on the weight associated with the edge between the nodes $A$ and $B$.

A neural network for information retrieval can be defined as illustrated in Figure 2.7. The model depicted here is based on the work in [815]. We first

observe that the neural network in Figure 2.7 is composed of three layers: one for the query terms, one for the document terms, and a third one for the documents themselves. Observe the similarity between the topology of this neural network and the topology of the inference and belief networks depicted in Figures 2.9 and 2.10. Here, however, the query term nodes are the ones which initiate the inference process by sending signals to the document term nodes. Following that, the document term nodes might themselves generate signals to the document nodes. This completes a first phase in which a signal travels from the query term nodes to the document nodes (i.e., from the left to the right in Figure 2.7).

The neural network, however, does not stop after the first phase of signal propagation. In fact, the document nodes in their turn might generate new signals which are directed back to the document term nodes (this is the reason for the bidirectional edges between document term nodes and document nodes). Upon receiving this stimulus, the document term nodes might again fire new signals directed to the document nodes, repeating the process. The signals become weaker at each iteration and the spread activation process eventually halts. This process might activate a document $d_l$ even when such a document does not contain any query terms. Thus, the whole process can be interpreted as the activation of a built-in thesaurus.

To the query term nodes is assigned an initial (and fixed) activation level equal to 1 (the maximum). The query term nodes then send signals to the document term nodes which are attenuated by normalized query term weights $\overline{w}_{i,q}$. For a vector-based ranking, these normalized weights can be derived from the weights $w_{i,q}$ defined for the vector model by equation 2.4. For instance,

$$\overline{w}_{i,q} = \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

where the normalization is done using the norm of the query vector.

Once the signals reach the document term nodes, these might send new signals out directed towards the document nodes. These signals are attenuated by normalized document term weights $\overline{w}_{i,j}$ derived from the weights $w_{i,j}$ defined for the vector model by equation 2.3. For instance,

$$\overline{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

where the normalization is done using the norm of the document vector.

The signals which reach a document node are summed up. Thus, after the first round of signal propagation, the activation level of the document node associated to the document $d_j$ is given by

$$\sum_{i=1}^{t} \overline{w}_{i,q}\, \overline{w}_{i,j} = \frac{\sum_{i=1}^{t} w_{i,q}\, w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

which is exactly the ranking provided by the classic vector model.

To improve the retrieval performance, the network continues with the spreading activation process after the first round of propagation. This modifies the initial vector ranking in a process analogous to a user relevance feedback cycle (see Chapter 5). To make the process more effective, a minimum activation threshold might be defined such that document nodes below this threshold send no signals out. Details can be found in [815].

There is no conclusive evidence that a neural network provides superior retrieval performance with general collections. In fact, the model has not been tested extensively with large document collections. However, a neural network does present an alternative modeling paradigm. Further, it naturally allows retrieving documents which are not initially related to the query terms — an appealing functionality.

## 2.8 Alternative Probabilistic Models

One alternative which has always been considered naturally appealing for quantifying document relevance is the usage of probability theory and its main streams. One such stream which is gaining increased attention concerns the *Bayesian belief networks* which we now discuss.

Bayesian (belief) networks are useful because they provide a clean formalism for combining distinct sources of evidence (past queries, past feedback cycles, and distinct query formulations) in support of the rank for a given document. This combination of distinct evidential sources can be used to improve retrieval performance (i.e., to improve the 'quality' of the ranked list of retrieved documents) as has been demonstrated in the work of Turtle and Croft [771].

In this chapter we discuss two models for information retrieval based on Bayesian networks. The first model is called *inference network* and provides the theoretical basis for the retrieval engine in the Inquery system [122]. Its success has attracted attention to the use of Bayesian networks with information retrieval systems. The second model is called *belief network* and generalizes the first model. At the end, we briefly compare the two models.

Our discussion below uses a style which is quite distinct from that employed by Turtle and Croft in their original writings. Particularly, we pay more attention to probabilistic argumentation during the development of the model. We make a conscious effort of consistently going back to the Bayesian formalism for motivating the major design decisions. It is our view that such an explanation strategy allows for a more precise argumentation which facilitates the task of grasping the subtleties involved.

Before proceeding, we briefly introduce Bayesian networks.

### 2.8.1 Bayesian Networks

Bayesian networks [630] are directed acyclic graphs (DAGs) in which the nodes represent random variables, the arcs portray causal relationships between these

variables, and the strengths of these causal influences are expressed by conditional probabilities. The *parents* of a node (which is then considered as a *child* node) are those judged to be direct *causes* for it. This causal relationship is represented in the DAG by a link directed from each parent node to the child node. The *roots* of the network are the nodes without parents.

Let $x_i$ be a node in a Bayesian network $G$ and $\Gamma_{x_i}$ be the set of parent nodes of $x_i$. The influence of $\Gamma_{x_i}$ on $x_i$ can be specified by any set of functions $F_i(x_i, \Gamma_{x_i})$ that satisfy

$$\begin{aligned} \sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) &= 1 \\ 0 \leq F_i(x_i, \Gamma_{x_i}) &\leq 1 \end{aligned}$$

where $x_i$ also refers to the states of the random variable associated to the node $x_i$. This specification is complete and consistent because the product $\prod_{\forall i} F_i(x_i, \Gamma_{x_i})$ constitutes a joint probability distribution for the nodes in $G$.

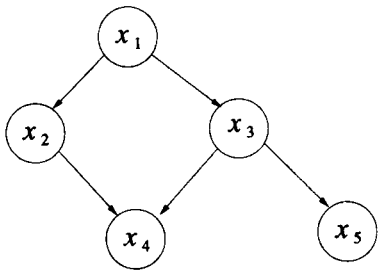


**Figure 2.8** An example of a Bayesian network.

Figure 2.8 illustrates a Bayesian network for a joint probability distribution $P(x_1, x_2, x_3, x_4, x_5)$. In this case, the dependencies declared in the network allow the natural expression of the joint probability distribution in terms of local conditional probabilities (a key advantage of Bayesian networks) as follows.

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2|x_1)P(x_3|x_1)P(x_4|x_2, x_3)P(x_5|x_3)$$

The probability $P(x_1)$ is called the *prior* probability for the network and can be used to model previous knowledge about the semantics of the application.

### 2.8.2 Inference Network Model

The two most traditional schools of thought in probability are based on the *frequentist* view and the *epistemological* view. The frequentist view takes probability as a statistical notion related to the laws of chance. The epistemological

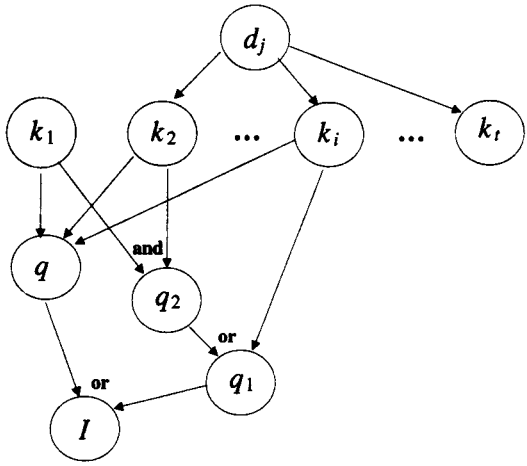


**Figure 2.9** Basic inference network model.

view interprets probability as a degree of belief whose specification might be devoid of statistical experimentation. This second viewpoint is important because we frequently refer to probabilities in our daily lives without a clear definition of the statistical experiment which yielded those probabilities.

The inference network model [772, 771] takes an epistemological view of the information retrieval problem. It associates random variables with the index terms, the documents, and the user queries. A random variable associated with a document $d_j$ represents the event of observing that document (i.e., the model assumes that documents are being observed in the search for relevant documents). The observation of the document $d_j$ asserts a belief upon the random variables associated with its index terms. Thus, observation of a document is the *cause* for an increased belief in the variables associated with its index terms.

Index term and document variables are represented as nodes in the network. Edges are directed from a document node to its term nodes to indicate that observation of the document yields improved belief on its term nodes.

The random variable associated with the user query models the event that the information request specified by the query has been met. This random variable is also represented by a node in the network. The belief in this (query) node is a function of the beliefs in the nodes associated with the query terms. Thus, edges are directed from the index term nodes to the query node. Figure 2.9 illustrates an inference network for information retrieval. The document $d_j$ has $k_2$, $k_i$, and $k_t$ as its index terms. This is modeled by directing the edges from the node $d_j$ to the nodes $k_2$, $k_i$, and $k_t$. The query $q$ is composed of the index terms $k_1$, $k_2$, and $k_i$. This is modeled by directing the edges from the nodes $k_1$, $k_2$, and $k_i$ to the node $q$. Notice that Figure 2.9 also includes three extra nodes: $q_2$, $q_1$, and $I$. The nodes $q_2$ and $q_1$ are used to model an (alternative) Boolean formulation $q_1$ for the query $q$ (in this case, $q_1 = (k_1 \wedge k_2) \vee k_i$). When such

(additional) information is available, the user information need $I$ is supported by both $q$ and $q_1$.

In what follows, we concentrate our attention on the support provided to the query node $q$ by the observation of a document $d_j$. Later on, we discuss the impact of considering multiple query representations for an information need $I$. This is important because, as Turtle and Croft have demonstrated, a keyword-based query formulation (such as $q$) can be combined with a Boolean-like query formulation (such as $q_1$) to yield improved retrieval performance for the same information need.

The complete inference network model also includes text nodes and query concept nodes but the model discussed above summarizes the essence of the approach.

A simplifying assumption is made which states that all random variables in the network are binary. This seems arbitrary but it does simplify the modeling task and is general enough to capture all the important relationships in the information retrieval problem.

**Definition** *Let $\vec{k}$ be a t-dimensional vector defined by $\vec{k} = (k_1, k_2, \ldots, k_t)$ where $k_1$, $k_2$, ..., $k_t$ are binary random variables i.e., $k_i \in \{0, 1\}$. These variables define the $2^t$ possible states for $\vec{k}$. Further, let $d_j$ be a binary random variable associated with a document $d_j$ and let $q$ be a binary random variable associated with the user query.*

Notice that $q$ is used to refer to the query, to the random variable associated with it, and to the respective node in the network. This is also the case for $d_j$ and for each index term $k_i$. We allow this overloading in syntax because it should always be clear whether we are referring to either the query or to its associated random variable.

The ranking of a document $d_j$ with respect to a query $q$ is a measure of how much evidential support the observation of $d_j$ provides to the query $q$. In an inference network, the ranking of a document $d_j$ is computed as $P(q \wedge d_j)$ where $q$ and $d_j$ are short representations for $q = 1$ and $d_j = 1$, respectively. In general, such a ranking is given by

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q \wedge d_j | \vec{k}) \times P(\vec{k}) \\ &= \sum_{\forall \vec{k}} P(q \wedge d_j \wedge \vec{k}) \\ &= \sum_{\forall \vec{k}} P(q | d_j \times \vec{k}) \times P(d_j \times \vec{k}) \\ &= \sum_{\forall \vec{k}} P(q | \vec{k}) \times P(\vec{k} | d_j) \times P(d_j) \qquad \textbf{(2.6)} \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

which is obtained by basic conditioning and the application of Bayes' rule. Notice that $P(q|d_j \times \vec{k}) = P(q|\vec{k})$ because the $k_i$ nodes separate the query node $q$ from the document node $d_j$. Also, the notation $\overline{q \wedge d_j}$ is a short representation for $\neg(q \wedge d_j)$.

The instantiation of a document node $d_j$ (i.e., the observation of the document) separates its children index term nodes making them mutually independent (see Bayesian theory for details). Thus, the degree of belief asserted to each index term node $k_i$ by instantiating the document node $d_j$ can be computed separately. This implies that $P(\vec{k}|d_j)$ can be computed in product form which yields (from equation 2.6),

$$P(q \wedge d_j) = \sum_{\forall \vec{k}} P(q|\vec{k}) \times \left( \prod_{\forall i | g_i(\vec{k})=1} P(k_i|d_j) \times \prod_{\forall i | g_i(\vec{k})=0} P(\overline{k}_i|d_j) \right) \times P(d_j) \quad (2.7)$$

$$P(\overline{q \wedge d_j}) = 1 - P(q \wedge d_j)$$

where $P(\overline{k}_i|d_j) = 1 - P(k_i|d_j)$. Through proper specification of the probabilities $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$, we can make the inference network cover a wide range of useful information retrieval ranking strategies. Later on, we discuss how to use an inference network to subsume the Boolean model and tf-idf ranking schemes. Let us first cover the specification of the $P(d_j)$ probabilities.

### Prior Probabilities for Inference Networks

Since the document nodes are the root nodes in an inference network, they receive a *prior probability distribution* which is of our choosing. This prior probability reflects the probability associated to the event of observing a given document $d_j$ (to simplify matters, a single document node is observed at a time). Since we have no prior preferences for any document in particular, we usually adopt a prior probability distribution which is uniform. For instance, in the original work on inference networks [772, 771], the probability of observing a document $d_j$ is set to $1/N$ where $N$ is the total number of documents in the system. Thus,

$$P(d_j) = \frac{1}{N}$$

$$P(\overline{d}_j) = 1 - \frac{1}{N}$$

The choice of the value $1/N$ for the prior probability $P(d_j)$ is a simple and natural specification given that our collection is composed of $N$ documents. However, other specifications for $P(d_j)$ might also be interesting. For instance, in the cosine formula of the vector model, the contribution of an index term to

the rank of the document $d_j$ is inversely proportional to the norm of the vector $\vec{d}_j$. The larger the norm of the document vector, the smaller is the relative contribution of its index terms to the document final rank. This effect can be taken into account through proper specification of the prior probabilities $P(d_j)$ as follows.

$$\begin{aligned} P(d_j) &= \frac{1}{|\vec{d}_j|} \\ P(\bar{d}_j) &= 1 - P(d_j) \end{aligned}$$

where $|\vec{d}_j|$ stands for the norm of the vector $\vec{d}_j$. Therefore, in this case, the larger the norm of a document vector, the smaller its associated prior probability. Such specification reflects a prior knowledge that we have about the behavior of vector-based ranking strategies (which normalize the ranking in the document space). As commanded by Bayesian postulates, previous knowledge of the application domain should be asserted in the specification of the priors in the network, as we have just done.

### Inference Network for the Boolean Model

Here we demonstrate how an inference network can be tuned to subsume the Boolean model. First, for the Boolean model, the prior probabilities $P(d_j)$ are all set to $1/N$ because the Boolean model makes no prior distinction on documents. Thus,

$$\begin{aligned} P(d_j) &= \frac{1}{N} \\ P(\bar{d}_j) &= 1 - P(d_j) \end{aligned}$$

Regarding the conditional probabilities $P(k_i|d_j)$ and $P(q|\vec{k})$, the specification is as follows.

$$\begin{aligned} P(k_i|d_j) &= \begin{cases} 1 & \text{if } g_i(d_j) = 1 \\ 0 & \text{otherwise} \end{cases} \\ P(\bar{k}_i|d_j) &= 1 - P(k_i|d_j) \end{aligned}$$

which basically states that, when the document $d_j$ is being observed, only the nodes associated with the index terms of the document $d_j$ are *active* (i.e., have an induced probability greater than 0). For instance, observation of a document node $d_j$ whose term vector is composed of exactly the index terms $k_2$, $k_i$, and $k_t$ (see Figure 2.9) activates the index term nodes $\{k_2,k_i,k_t\}$ and no others.

Once the beliefs in the index term nodes have been computed, we can use them to compute the evidential support they provide to the user query $q$ as

follows.

$$
\begin{aligned}
P(q|\vec{k}) &= \begin{cases} 1 & \text{if } \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i},\ g_i(\vec{k}) = g_i(\vec{q}_{cc})) \\ 0 & \text{otherwise} \end{cases} \\
P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k})
\end{aligned}
$$

where $\vec{q}_{cc}$ and $\vec{q}_{dnf}$ are as defined for the classic Boolean model. The above equation basically states that one of the conjunctive components of the user query (expressed in disjunctive normal form) must be matched by the set of active terms in $\vec{k}$ (in this case, those activated by the document observed) exactly.

Substituting the above definitions for $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$ into equation 2.7, it can be easily shown that the set of documents retrieved is exactly the set of documents returned by the Boolean model as defined in section 2.5.2. Thus, an inference network can be used to subsume the Boolean model without difficulties.

### Inference Network for tf-idf Ranking Strategies

For tf-idf ranking strategies (i.e., those related to the vector model), we adopt prior probabilities which reflect our prior knowledge of the importance of document normalization. Thus, we set the prior $P(d_j)$ to $1/|\vec{d}_j|$ as follows.

$$
\begin{aligned}
P(d_j) &= \frac{1}{|\vec{d}_j|} \\
P(\overline{d}_j) &= 1 - P(d_j)
\end{aligned}
\tag{2.8}
$$

Further, we have to decide where to introduce the tf (term-frequency) and the idf (inverse-document-frequency) factors in the network. For that purpose, we consider that the tf and idf factors are normalized (as in equation 2.1) and that these normalized factors are strictly smaller than 1.

We first focus on capturing the impact of the tf factors in the network. Normalized tf factors are taken into account through the beliefs asserted upon the index term nodes as follows.

$$
\begin{aligned}
P(k_i|d_j) &= f_{i,j} \\
P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j)
\end{aligned}
\tag{2.9}
$$

These equations simply state that, according to the observed document $d_j$, the relevance of a term $k_i$ is determined by its normalized term-frequency factor.

We are now in a position to consider the influence of idf factors. They are taken into account through the specification of the impact of index term nodes

on the query node. Define a vector $\vec{k}_i$ given by,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\, g_j(\vec{k}) = 0)$$

The vector $\vec{k}_i$ is a reference to the state of the vector $\vec{k}$ in which the node $k_i$ is active and all others are inactive. The motivation is that tf-idf ranking strategies sum up the *individual* contributions of index terms and $\vec{k}_i$ allows us to consider the influence of the term $k_i$ in isolation. We are now ready to define the influence of the index term nodes in the query node $q$ as

$$\begin{aligned} P(q|\vec{k}) &= \begin{cases} idf_i & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{q}) = 1 \\ 0 & \text{if } \vec{k} \neq \vec{k}_i \vee g_i(\vec{q}) = 0 \end{cases} \\ P(\bar{q}|\vec{k}) &= 1 - P(q|\vec{k}) \end{aligned} \tag{2.10}$$

where $idf_i$ here is a normalized version of the idf factor defined in equation 2.2.

By applying equations 2.8, 2.9, and 2.10 to equation 2.7, we can then write

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}_i} P(q|\vec{k}_i) \times P(k_i|d_j) \times \left( \prod_{\forall\, l \neq i} P(\bar{k}_l|d_j) \right) \times P(d_j) \\ &= \left( \prod_{\forall\, i} P(\bar{k}_i|d_j) \right) \times P(d_j) \times \sum_{\forall \vec{k}_i} P(k_i|d_j) \times P(q|\vec{k}_i) \times \frac{1}{P(\bar{k}_i|d_j)} \\ &= C_j \times \frac{1}{|\vec{d}_j|} \times \sum_{\forall i | g_i(\vec{d}_j) = 1 \wedge g_i(\vec{q}) = 1} f_{i,j} \times idf_i \times \frac{1}{1 - f_{i,j}} \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

which provides a tf-idf-like ranking. Unfortunately, $C_j$ depends on a product of the various probabilities $P(\bar{k}_i|d_j)$ which vary from document to document and thus the ranking is distinct from the one provided by the vector model. Despite this peculiarity in the tf-idf ranking generated, it has been shown that an inference network is able to provide good retrieval performance with general collections. The reason is that the network allows us to consistently combine evidence from distinct evidential sources to improve the final ranking, as we now discuss.

### Combining Evidential Sources

In Figure 2.9, the first query node $q$ is the standard keyword-based query formulation for the user information need $I$. The second query $q_1$ is a Boolean-like query formulation for the same information need (i.e., an additional evidential source collected from a specialist). The joint support these two query formulations provide to the information need node $I$ can be modeled through, for

instance, an OR operator (i.e., $I = q \vee q_1$). In this case, the ranking provided by the inference network is computed as,

$$\begin{aligned} P(I \wedge d_j) &= \sum_{\vec{k}} P(I|\vec{k}) \times P(\vec{k}|d_j) \times P(d_j) \\ &= \sum_{\vec{k}} (1 - P(\overline{q}|\vec{k})\ P(\overline{q}_1|\vec{k})) \times P(\vec{k}|d_j) \times P(d_j) \end{aligned}$$

which might yield a retrieval performance which surpasses the retrieval performance obtained with each of the query nodes in isolation as demonstrated in [771].

### 2.8.3 Belief Network Model

The belief network model, introduced in 1996 by Ribeiro-Neto and Muntz [674], is also based on an epistemological interpretation of probabilities. However, it departs from the inference network model by adopting a clearly defined sample space. As a result, it yields a slightly different network topology which provides a separation between the document and the query portions of the network. This is the main difference between the two models and one which has theoretical implications.

#### The Probability Space

The probability space adopted was first introduced by Wong and Yao [830] and works as follows. All documents in the collection are indexed by *index terms* and the universe of discourse $U$ is the set $K$ of all index terms.

**Definition** *The set $K = \{k_1, \ldots, k_t\}$ is the universe of discourse and defines the sample space for the belief network model. Let $u \subset K$ be a subset of $K$. To each subset $u$ is associated a vector $\vec{k}$ such that $g_i(\vec{k}) = 1 \Longleftrightarrow k_i \in u$.*

The introduction of the vector $\vec{k}$ is useful to keep the notation compatible with the one which has been used throughout this chapter.

Each index term is viewed as an *elementary concept* and $K$ as a concept space. A concept $u$ is a subset of $K$ and might represent a document in the collection or a user query. In a belief network, set relationships are specified using random variables as follows.

**Definition** *To each index term $k_i$ is associated a binary random variable which is also referred to as $k_i$. The random variable $k_i$ is set to 1 to indicate that the index $k_i$ is a member of a concept/set represented by $\vec{k}$.*

This association of concepts with subsets is useful because it allows us to express the logical notions of conjunction, disjunction, negation, and implication as the more familiar set-theoretic notions of intersection, union, complementation, and inclusion. Documents and user queries can be defined as concepts in the sample space $K$ as follows.

**Definition** *A document $d_j$ in the collection is represented as a concept (i.e., a set) composed of the terms which are used to index $d_j$. Analogously, a user query $q$ is represented as a concept composed of the terms which are used to index $q$.*

A probability distribution $P$ is defined over $K$ as follows. Let $c$ be a generic concept in the space $K$ representing a document or user query. Then,

$$P(c) = \sum_u P(c|u) \times P(u) \tag{2.11}$$

$$P(u) = \left(\frac{1}{2}\right)^t \tag{2.12}$$

Equation 2.11 defines $P(c)$ as the *degree of coverage* of the space $K$ by $c$. Such a coverage is computed by contrasting each of the concepts in $K$ with $c$ (through $P(c|u)$) and by summing up the individual contributions. This sum is weighted by the probability $P(u)$ with which $u$ occurs in $K$. Since at the beginning the system has no knowledge of the probability with which a concept $u$ occurs in the space $K$, we can assume that each $u$ is equally likely which results in equation 2.12.

### Belief Network Model

In the belief network model, the user query $q$ is modeled as a network node to which is associated a binary random variable (as in the inference network model) which is also referred to as $q$. This variable is set to 1 whenever $q$ completely covers the concept space $K$. Thus, when we assess $P(q)$ we compute the degree of coverage of the space $K$ by $q$. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that $q$ completely covers all possible concepts in $K$?

A document $d_j$ is modeled as a network node with which is associated a binary random variable which is also referred to as $d_j$. This variable is 1 to indicate that $d_j$ completely covers the concept space $K$. When we assess $P(d_j)$, we compute the degree of coverage of the space $K$ by $d_j$. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that $d_j$ completely covers all possible concepts in $K$?

According to the above formalism, the user query and the documents in the collection are modeled as subsets of index terms. Each of these subsets is interpreted as a *concept* embedded in the concept space $K$ which works as a common *sample space*. Furthermore, user queries and documents are modeled

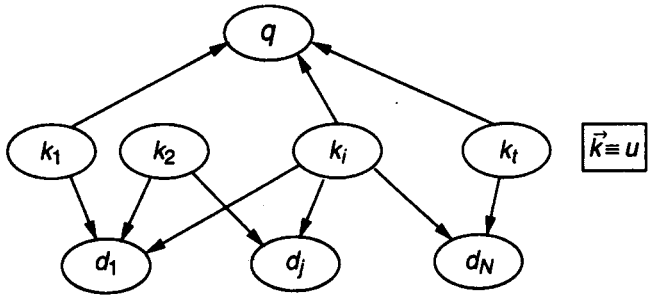


**Figure 2.10** Basic belief network model.

identically. This is an important observation because it defines the topology of the belief network.

Figure 2.10 illustrates our belief network model. As in the inference network model, a query $q$ is modeled as a binary random variable which is pointed to by the index term nodes which compose the query concept. Documents are treated analogously to user queries (i.e., both are concepts in the space $K$). Thus, contrary to the inference network model, a document node is pointed to by the index term nodes which compose the document. This is the topological difference between the two models and one which has more implications than it seems at first glance.

The ranking of a document $d_j$ relative to a given query $q$ is interpreted as a concept matching relationship and reflects the degree of coverage provided to the concept $d_j$ by the concept $q$.

*Assumption* In the belief network model, $P(d_j|q)$ is adopted as the rank of the document $d_j$ with respect to the query $q$.

By the application of Bayes' theorem, we can write $P(d_j|q) = P(d_j \wedge q)/P(q)$. Since $P(q)$ is a constant for all documents in the collection, we can write $P(d_j|q) \sim P(d_j \wedge q)$ i.e., the rank assigned to a document $d_j$ is directly proportional to $P(d_j \wedge q)$. This last probability is computed through the application of equation 2.11 which yields

$$P(d_j|q) \sim \sum_{\forall u} P(d_j \wedge q|u) \times P(u)$$

In the belief network of Figure 2.10, instantiation of the index term variables logically separates the nodes $q$ and $d$ making them mutually independent (i.e., the document and query portions of the network are logically separated by in-

stantiation of the index term nodes). Therefore,

$$P(d_j|q) \sim \sum_{\forall u} P(d_j|u) \times P(q|u) \times P(u)$$

which can be rewritten as

$$P(d_j|q) \sim \sum_{\forall \vec{k}} P(d_j|\vec{k}) \times P(q|\vec{k}) \times P(\vec{k}) \quad \textbf{(2.13)}$$

To complete the belief network we need to specify the conditional probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$. Distinct specifications of these probabilities allow the modeling of different ranking strategies (corresponding to different IR models). We now discuss how to specify these probabilities to subsume the vector model.

For the vector model, the probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$ are specified as follows. Let,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\ g_j(\vec{k}) = 0)$$

as before. Also,

$$P(q|\vec{k}) = \begin{cases} \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(q) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{q}|\vec{k}) = 1 - P(q|\vec{k})$$

Further, define

$$P(d_j|\vec{k}) = \begin{cases} \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{d}_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{d_j}|\vec{k}) = 1 - P(d_j|\vec{k})$$

Then, the ordering of the retrieved documents (i.e., the ranking) defined by $P(d_j|q)$ coincides with the ordering generated by the vector model as specified in section 2.5.3. Thus, the belief network model can be tuned to subsume the vector model which cannot be accomplished with the inference network model.

### 2.8.4 Comparison of Bayesian Network Models

There is a close resemblance between the belief network model and the inference network model. However, this resemblance hides important differences between the two models. First, the belief network model is based on a set-theoretic view

of the IR ranking problem and adopts a clearly defined sample space. The inference network model takes a purely epistemological view of the IR problem which is more difficult to grasp (because, for instance, the sample space is not clearly defined). Second, the belief network model provides a separation between the document and the query portions of the network which facilitates the modeling of additional evidential sources such as past queries and past relevance information. Third, as a result of this document-query space separation, the belief network model is able to reproduce any ranking strategy generated by the inference network model while the converse is not true.

To see that the belief network ranking subsumes any ranking generated by an inference network, compare equations 2.6 and 2.13. The key distinction is between the terms $P(d_j|\vec{k})$ and $P(\vec{k}|d_j)$. For the latter, instantiation of the document node $d_j$ separates the index term nodes making them mutually independent. Thus, the joint probability $P(\vec{k}|d_j)$ can always be computed as the product of the individual probabilities $P(k_i|d_j)$. However, the computation of $P(d_j|\vec{k})$ might be non-decomposable in a product of term-based probabilities. As a result, $P(d_j|\vec{k})$ can express any probability function defined with $P(\vec{k}|d_j)$ while the converse is not true.

One should not infer from the above comparison that the inference network model is not a good model. On the contrary, it has been shown in the literature that the inference network model allows top retrieval performance to be accomplished with general collections. Further, it is the retrieval model used by the Inquery system. The point of the comparison is that, from a theoretical point of view, the belief network model is more general. Also, it provides a separation between the document space and the query space which simplifies the modeling task.

### 2.8.5 Computational Costs of Bayesian Networks

In the inference network model, according to equation 2.6, only the states which have a single document active node are considered. Thus, the cost of computing the ranking is linear on the number of documents in the collection. As with conventional collections, index structures such as inverted files (see Chapter 8) are used to restrict the ranking computation to those documents which have terms in common with the query. Thus, the cost of computing an inference network ranking has the same complexity as the cost of computing a vectorial ranking.

In the belief network model, according to equation 2.13, the only states (of the roots nodes) considered (for computing the rank of a document $d_j$) are the ones in which the active nodes are exactly those associated with the query terms. Thus, again, the cost of computing the ranking is linear on the number of documents in the collection. If index structures are used, the cost of computing a belief network ranking has the same complexity as the cost of computing a vectorial ranking.

Therefore, the Bayesian network models discussed here do not impose significant additional costs for ranking computation. This is so because the networks presented do not include cycles, which implies that belief propagation can be done in a time proportional to the number of nodes in the network.

### 2.8.6 The Impact of Bayesian Network Models

The classic Boolean model is based on a neat formalism but is not very effective for information retrieval. The classic vector model provides improved answer sets but lacks a more formal framework. Many attempts have been made in the past to combine the best features of each model. The extended Boolean model and the generalized vector space model are two well known examples. These past attempts are grounded in the belief that the combination of selected properties from distinct models is a promising approach towards improved retrieval.

Bayesian network models constitute modern variants of probabilistic reasoning whose major strength (for information retrieval) is a framework which allows the neat combination of distinct evidential sources to support a relevance judgement (i.e., a numerical rank) on a given document. In this regard, belief networks seem more appropriate than previous approaches and more promising. Further, besides allowing the combination of Boolean and vector features, a belief network can be naturally extended to incorporate evidential information derived from past user sessions [674] and feedback cycles [332].

The inference network model has been successfully implemented in the Inquery retrieval system [122] and compares favorably with other retrieval systems. However, despite these promises, whether Bayesian networks will become popular and widely used for information retrieval remains to be seen.

## 2.9 Structured Text Retrieval Models

Consider a user with a superior visual memory. Such a user might then recall that the specific document he is interested in contains a page in which the string '*atomic holocaust*' appears in italic in the text surrounding a Figure whose label contains the word 'earth.' With a classic information retrieval model, this query could be expressed as ['atomic holocaust' and 'earth'] which retrieves all the documents containing both strings. Clearly, however, this answer contains many more documents than desired by this user. In this particular case, the user would like to express his query through a richer expression such as

same-page (near ('*atomic holocaust*,' Figure (label ('earth'))))

which conveys the details in his visual recollection. Further, the user might be interested in an advanced interface which simplifies the task of specifying this (now complex) query. This example illustrates the appeal of a query language which allows us to combine the specification of strings (or patterns) with the

specification of structural components of the document. Retrieval models which combine information on text content with information on the document structure are called *structured text retrieval* models.

For a query such as the one illustrated above, a structured text retrieval system searches for all the documents which *satisfy* the query. Thus, there is no notion of relevance attached to the retrieval task. In this sense, the current structured text retrieval models are data (instead of information) retrieval models. However, the retrieval system could search for documents which match the query conditions only partially. In this situation, the matching would be approximate and some ranking would have to be used for ordering the approximate answers. Thus, a structured text retrieval algorithm can be seen as an information retrieval algorithm for which the issue of appropriate ranking is not well established. In fact, this is an actual, interesting, and open research problem.

At the end of the 1980s and throughout the 1990s, various structured text retrieval models have appeared in the literature. Usually, the more expressive the model, the less efficient is its query evaluation strategy. Thus, selection of a structured model for a given application must be exercised with care. A good policy is to select the most efficient model which supports the functionality required by the application in view.

Here, we do not survey all the structured text retrieval models. Instead, we briefly discuss the main features of two of them, namely, a model based on *non-overlapping lists* and a model based on *proximal nodes*. These two models should provide a good overview of the main issues and tradeoffs in structured text retrieval.

We use the term *match point* to refer to the position in the text of a sequence of words which matches (or satisfies) the user query. Thus, if the user specifies the simple query ['atomic holocaust in Hiroshima'] and this string appears in three positions in the text of a document $d_j$, we say that the document $d_j$ contains three match points. Further, we use the term *region* to refer to a contiguous portion of the text and the term *node* to refer to a structural component of the document such as a chapter, a section, a subsection, etc. Thus, a node is a region with predefined topological properties which are known both to the author of the document and to the user who searches the document system.

### 2.9.1 Model Based on Non-Overlapping Lists

Burkowski [132, 133] proposes to divide the whole text of each document in non-overlapping text regions which are collected in a *list*. Since there are multiple ways to divide a text in non-overlapping regions, multiple lists are generated. For instance, we might have a list of all chapters in the document, a second list of all sections in the document, and a third list of all subsections in the document. These lists are kept as separate and distinct data structures. While the text regions in the same (flat) list have no overlapping, text regions from distinct lists might overlap. Figure 2.11 illustrates four separate lists for the same document.

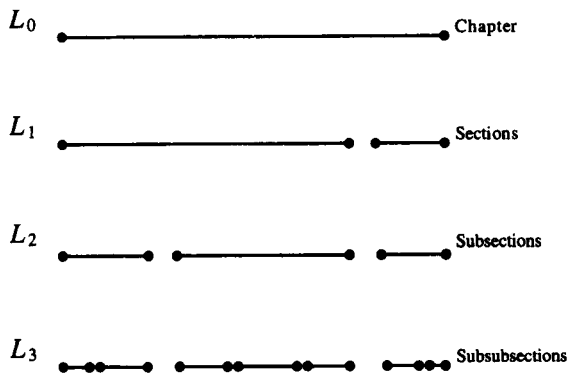


**Figure 2.11** Representation of the structure in the text of a document through four separate (flat) indexing lists.

To allow searching for index terms and for text regions, a single inverted file (see Chapter 8 for a definition of inverted files) is built in which each structural component stands as an entry in the index. Associated with each entry, there is a list of text regions as a list of occurrences. Moreover, such a list could be easily merged with the traditional inverted file for the words in the text. Since the text regions are non-overlapping, the types of queries which can be asked are simple: (a) select a region which contains a given word (and does not contain other regions); (b) select a region $A$ which does not contain any other region $B$ (where $B$ belongs to a list distinct from the list for $A$); (c) select a region not contained within any other region, etc.

### 2.9.2 Model Based on Proximal Nodes

Navarro and Baeza-Yates [41, 589, 590] propose a model which allows the definition of independent hierarchical (non-flat) indexing structures over the same document text. Each of these indexing structures is a strict hierarchy composed of chapters, sections, paragraphs, pages, and lines which are called *nodes* (see Figure 2.12). To each of these nodes is associated a text region. Further, two distinct hierarchies might refer to overlapping text regions.

Given a user query which refers to distinct hierarchies, the compiled answer is formed by nodes which all come from only one of them. Thus, an answer cannot be composed of nodes which come from two distinct hierarchies (which allows for faster query processing at the expense of less expressiveness). Notice, however, that due to the hierarchical structure, nested text regions (coming from the same hierarchy) are allowed in the answer set.

Figure 2.12 illustrates a hierarchical indexing structure composed of four

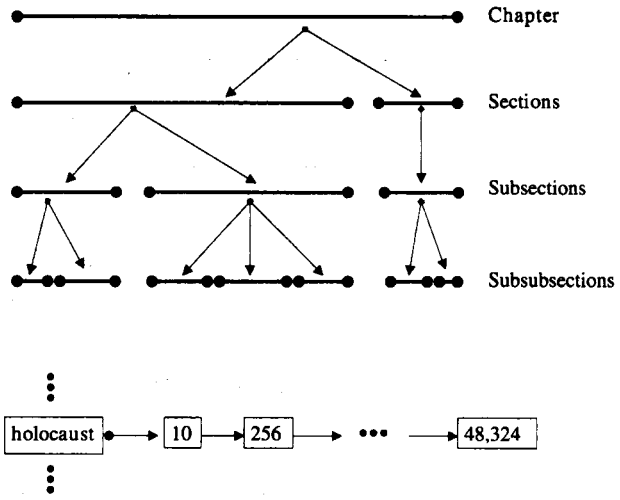


**Figure 2.12** Hierarchical index for structural components and flat index for words.

levels (corresponding to a chapter, sections, subsections, and subsubsections of the same document) and an inverted list for the word 'holocaust.' The entries in this inverted list indicate all the positions in the text of the document in which the word 'holocaust' occurs. In the hierarchy, each node indicates the position in the text of its associated structural component (chapter, section, subsection, or subsubsection).

The query language allows the specification of regular expressions (to search for strings), the reference to structural components by name (to search for chapters, for instance), and a combination of these. In this sense, the model can be viewed as a compromise between expressiveness and efficiency. The somewhat limited expressiveness of the query language allows efficient query processing by first searching for the components which match the strings specified in the query and, subsequently, evaluating which of these components satisfy the structural part of the query.

Consider, for instance, the query [(*section) with ('holocaust')] which searches for sections, subsections, or subsubsections which contain the word 'holocaust.' A simple query processing strategy is to traverse the inverted list for the term 'holocaust' and, for each entry in the list (which indicates an occurrence of the term 'holocaust' in the text), search the hierarchical index looking for sections, subsections, and subsubsections containing that occurrence of the term. A more sophisticated query processing strategy is as follows. For the first entry in the list for 'holocaust,' search the hierarchical index as before. This implies traversing down the hierarchy until no more successful matches occur (or the bottom of the hierarchy is reached). Let the last matching structural component be referred to as the innermost matching component. Once this first search is concluded, do not start all over again for the following entry in the

inverted list. Instead, verify whether the innermost matching component also matches the second entry in the list. If it does, we immediately conclude that the larger structural components above it (in the hierarchy) also do. Proceed then to the third entry in the list, and so on. Notice that the query processing is accelerated because only the nearby (or proximal) nodes in the list need to be searched at each time. This is the reason for the label *proximal nodes*.

The model based on proximal nodes allows us to formulate queries which are more complex than those which can be formulated in the model based on non-overlapping lists. To speed up query processing, however, only nearby (proximal) nodes are looked at which imposes restrictions on the answer set retrieved (all nodes must come from the same hierarchy). More complex models for structured retrieval have been proposed in the literature as discussed in [41, 590].

## 2.10 Models for Browsing

As already observed, the user might not be interested in posing a specific query to the system. Instead, he might be willing to invest some time in exploring the document space looking for interesting references. In this situation, we say that the user is *browsing* the space instead of searching. Both with browsing and searching, the user has goals which he is pursuing. However, in general, the goal of a searching task is clearer in the mind of the user than the goal of a browsing task. As is obvious, this is not a distinction which is valid in all scenarios. But, since it is simple and provides a clear separation between the tasks of searching and browsing, it is adopted here. We distinguish three types of browsing namely, flat, structure guided, and hypertext.

### 2.10.1 Flat Browsing

The idea here is that the user explores a document space which has a flat organization. For instance, the documents might be represented as dots in a (two-dimensional) plan or as elements in a (single dimension) list. The user then glances here and there looking for information within the documents visited. For instance, he might look for correlations among neighbor documents or for keywords which are of interest to him. Such keywords could then be added to the original query in an attempt to provide better contextualization. This is a process called *relevance feedback* which is discussed in detail in Chapter 5. Also, the user could explore a single document in a flat manner. For example, he could use a browser to look into a Web page, using the arrows and the scroll bar. One disadvantage is that in a given page or screen there may not be any indication about the context where the user is. For example, if he opens a novel at a random page, he might not know in which chapter that page is.

Web search engines such as 'Yahoo!' provide, besides the standard search interface, a hierarchical directory which can be used for browsing (and frequently, for searching). However, the organization is not flat as discussed below.

### 2.10.2 Structure Guided Browsing

To facilitate the task of browsing, the documents might be organized in a structure such as a directory. Directories are hierarchies of classes which group documents covering related topics. Such hierarchies of classes have been used to classify document collections for many centuries now. Thus, it seems natural to adapt them for use with modern browsing interfaces. In this case, we say that the user performs a structure guided type of browsing. The same idea can be applied to a single document. For example, if we are browsing an electronic book, a first level of content could be the chapters, the second level, all sections, and so on. The last level would be the text itself (flat). A good user interface could go down or up those levels in a focused manner, assisting the user with the task of keeping track of the context.

Besides the structure which directs the browsing task, the interface can also include facilities such as a history map which identifies classes recently visited. This might be quite useful for dealing with very large structures – an issue discussed in Chapters 10 and 13. When searching, the occurrences can also be displayed showing just the structure (for example, using the table of contents). This allows us to see the occurrences in a global context instead of in a page of text that may have no indication of where we are.

### 2.10.3 The Hypertext Model

One fundamental concept related to the task of writing down text is the notion of *sequencing*. Written text is usually conceived to be read sequentially. The reader should not expect to fully understand the message conveyed by the writer by randomly reading pieces of text here and there. One might rely on the text structure to skip portions of the text but this might result in miscommunication between reader and writer. Thus, a sequenced organizational structure lies underneath most written text. When the reader fails to perceive such a structure and abide by it, he frequently is unable to capture the essence of the writer's message.

Sometimes, however, we are looking for information which is subsumed by the whole text but which cannot be easily captured through sequential reading. For instance, while glancing at a book about the history of the wars fought by man, we might be momentarily interested solely in the regional wars in Europe. We know that this information is in the book, but we might have a hard time finding it because the writer did not organize his writings with this purpose (he might have organized the wars chronologically). In such a situation, a different organization of the text is desired. However, there is no point in rewriting the whole text. Thus, the solution is to define a new organizational structure besides the one already in existence. One way to accomplish such a goal is through the design of a hypertext.

## Hypertext Definition and the Navigational Task

A *hypertext* is a high level interactive navigational structure which allows us to browse text non-sequentially on a computer screen. It consists basically of nodes which are correlated by directed links in a graph structure.

To each node is associated a text region which might be a chapter in a book, a section in an article, or a Web page. Two nodes $A$ and $B$ might be connected by a *directed link* $l_{AB}$ which correlates the texts associated with these two nodes. In this case, the reader might move to the node $B$ while reading the text associated with the node $A$.

In its most conventional form, a hypertext link $l_{AB}$ is attached to a specific string inside the text for node $A$. Such a string is marked specially (for instance, its characters might appear in a different color or underlined) to indicate the presence of the underlying link. While reading the text, the user might come across a marked string. If the user clicks on that string, the underlying directed link is followed, and a new text region (associated with the node at the destination) is displayed on the screen.

The process of navigating the hypertext can be understood as a traversal of a directed graph. The linked nodes of the graph represent text nodes which are semantically related. While traversing this graph the reader visualizes a flow of information which was conceived by the designer of the hypertext. Consider our previous example regarding a book on the wars fought by man. One might design a hypertext composed of two distinct *webs* (here, a web is simply a connected component formed by a subset of all links in the hypertext). While the first web might be designed to provide access to the local wars fought in Europe in chronological order, the second web might be designed to provide access to the local wars fought by each European country. In this way, the user of this hypertext can access the information according to his particular need.

When the hypertext is large, the user might lose track of the organizational structure of the hypertext. The effect is that the user might start to take bad navigational decisions which might sidetrack him from his main goal (which usually consists of finding a piece of information in the hypertext). When this happens, the user is said to be *lost in hyperspace* [604]. To avoid this problem, it is desirable that the hypertext include a hypertext map which shows where the user is at all times. In its simplest form, this map is a directed graph which displays the current node being visited. Additionally, such a map could include information on the paths the user has traveled so far. This can be used to remind the user of the uselessness of following paths which have been explored already.

While navigating a hypertext, the user is restricted to the intended flow of information previously conceived by the hypertext designer. Thus, the task of designing a hypertext should take into account the needs of its potential users. This implies the execution of a requirement analysis phase before starting the actual implementation of the hypertext. Such a requirement analysis is critically important but is frequently overlooked.

Furthermore, during the hypertext navigation, the user might find it difficult to orient himself. This difficulty arises even in the presence of a guiding

tool such as the hypertext map discussed above. One possible reason is an excessively complex hypertext organization with too many links which allow the user to travel back and forth. To avoid this problem, the hypertext can have a simpler structure which can be quickly remembered by the user at all times. For instance, the hypertext can be organized hierarchically to facilitate the navigational task.

Definition of the structure of the hypertext should be accomplished in a domain modeling phase (done after a requirement analysis phase). Further, after the modeling of the domain, a user interface design should be concluded prior to implementation. Only then, can we say that we have a proper hypertext structure for the application at hand. In the Web, however, pages are usually implemented with no attention paid to requirement analysis, domain modeling, and user interface design. As a result, Web pages are frequently poorly conceived and often fail to provide the user with a proper hypertext structure for assistance with the information seeking task.

With large hypertexts, it might be difficult for the user to position himself in the part of the whole graph which is of most interest to him. To facilitate this initial positioning step, a search based on index terms might be used. In [540], Manber discusses the advantages of this approach.

Hypertexts provided the basis for the conception and design of the hypertext markup language (HTML) and the hypertext transfer protocol (HTTP) which originated the *World Wide Web* (which we simply refer to as the Web). In Chapter 13, we discuss the Web in detail. We briefly discuss some of its features below.

### About the Web

When one talks about the Web, the first concept which comes to mind is that of a hypertext. In fact, we frequently think of the Web as a huge distributed hypertext domain. However, the Web is not exactly a proper hypertext because it lacks an underlying data model, it lacks a navigational plan, and it lacks a consistently designed user interface. Each one of the millions of Web page designers devises his own interface with its own peculiar characteristics. Many times we visit a Web site simply looking for a phone number and cannot find it because it is buried in the least expected place of the local hypertext structure. Thus, the Web user has no underlying metaphor to assist him in the search for information of interest.

Instead of saying that the Web is a hypertext, we prefer to say that it is a pool of (partially) interconnected webs. Some of these webs might be characterized as a local hypertext (in the sense that they have an underlying structure which enjoys some consistency) but others might be simply a collection of pages designed separately (for instance, the web of a university department whose professors design their own pages). Despite not being exactly a hypertext, the Web has provided us with a new dimension in communication functionality because it is easily accessible world wide at very low cost. And maybe most important, the Web has no control body setting up regulations and censorship rules. As a

result, for the first time in the history of mankind, any one person can publish his writings through a large medium without being subjected to the filtering of an editorial board.

For a more thorough discussion of these and many other issues related to the Web, the user is referred to Chapter 13.

## 2.11 Trends and Research Issues

There are three main types of products and systems which can benefit directly from research in models for information retrieval: library systems, specialized retrieval systems, and the Web.

Regarding library systems, there is currently much interest in cognitive and behavioral issues oriented particularly at a better understanding of which criteria the users adopt to judge relevance. From the point of view of the computer scientist, a main question is how this knowledge about the user affects the ranking strategies and the user interface implemented by the system. A related issue is the investigation of how models other than the Boolean model (which is still largely adopted by most large commercial library systems) affect the user of a library.

A specialized retrieval system is one which is developed with a particular application in mind. For instance, the LEXIS-NEXIS retrieval system (see Chapter 14), which provides access to a very large collection of legal and business documents, is a good example of a specialized retrieval system. In such a system, a key problem is how to retrieve (almost) all documents which might be relevant to the user information need without also retrieving a large number of unrelated documents. In this context, sophisticated ranking algorithms are highly desirable. Since ranking based on single evidential sources is unlikely to provide the appropriate answers, research on approaches for combining several evidential sources seems highly relevant (as demonstrated at the various TREC conferences, see Chapter 3 for details).

In the Web, the scenario is quite distinct and unique. In fact, the user of the Web frequently does not know what he wants or has great difficulty in properly formulating his request. Thus, research in advanced user interfaces is highly desirable. From the point of view of the ranking engine, an interesting problem is to study how the paradigm adopted for the user interface affects the ranking. Furthermore, it is now well established that the indexes maintained by the various Web search engines are almost disjoint (e.g., the ten most popular search engines have indexes whose intersection corresponds to less than 2% of the total number of pages indexed). In this scenario, research on meta-search engines (i.e., engines which work by fusing the rankings generated by other search engines) seems highly promising.

## 2.12 Bibliographic Discussion

Early in 1960, Maron and Kuhns [547] had already discussed the issues of relevance and probabilistic indexing in information retrieval. Twenty-three years

later, Salton and McGill wrote a book [698] which became a classic in the field. The book provides a thorough coverage of the three classic models in information retrieval namely, the Boolean, the vector, and the probabilistic models. Another landmark reference is the book by van Rijsbergen [785] which, besides also covering the three classic models, presents a thorough and enjoyable discussion on the probabilistic model. The book edited by Frakes and Baeza-Yates [275] presents several data structures and algorithms for IR and is more recent. Further, it includes a discussion of ranking algorithms by Harman [340] which provides interesting insights into the history of information retrieval from 1960 to 1990.

Boolean operations and their implementation are covered in [803]. The inadequacy of Boolean queries for information retrieval was characterized early on by Verhoeff, Goffman, and Belzer [786]. The issue of adapting the Boolean formalism to operate with other frameworks received great attention. Bookstein discusses the problems related with merging Boolean and weighted retrieval systems [101] and the implications of Boolean structure for probabilistic retrieval [103]. Losee and Bookstein [522] cover the usage of Boolean queries with probabilistic retrieval. Anick *et al.* [21] propose an interface based on natural language for Boolean retrieval. A thesaurus-based Boolean retrieval system is proposed in [493].

The vector model is maybe the most popular model among the research community in information retrieval. Much of this popularity is due to the long-term research of Salton and his associates [697, 704]. Most of this research revolved around the SMART retrieval system developed at Cornell University [695, 842, 696]. Term weighting for the vector model has also been investigated thoroughly. Simple term weighting was used early on by Salton and Lesk [697]. Sparck Jones introduced the idf factor [409, 410] and Salton and Yang verified its effectiveness for improving retrieval [704]. Yu and Salton [842] further studied the effects of term weighting in the final ranking. Salton and Buckley [696] summarize 20 years of experiments in term weighting with the SMART system. Raghavan and Wong [665] provide a critical analysis of the vector model.

The probabilistic model was introduced by Robertson and Sparck Jones [677] and is thoroughly discussed in [785]. Experimental studies with the model were conducted by Sparck Jones [411, 412] which used feedback from the user to estimate the initial probabilities. Croft and Harper [199] proposed a method to estimate these probabilities without feedback from the user. Croft [198] later on added within-document frequency weights into the model. Fuhr discusses probabilistic indexing through polynomial retrieval functions [281, 284]. Cooper, Gey, and Dabney [186] and later on Gey [295] propose the use of logistic regression with probabilistic retrieval. Lee and Kantor [494] study the effect of inconsistent expert judgements on probabilistic retrieval. Fuhr [282] reviews various variants of the classic probabilistic model. Cooper [187], in a seminal paper, raises troubling questions on the utilization of the probabilistic ranking principle in information retrieval.

The inference network model was introduced by Turtle and Croft [772, 771] in 1990. Haines and Croft [332] discuss the utilization of inference networks for user relevance feedback (see Chapter 5). Callan, Lu, and Croft [139] use an

inference network to search distributed document collections. Callan [138], in his turn, discusses the application of inference networks to information filtering. The belief network model, due to Ribeiro-Neto and Muntz [674], generalizes the inference network model.

The extended Boolean model was introduced by Salton, Fox, and Wu [703]. Lee, Kim, Kim, and Lee [496] discuss the evaluation of Boolean operators with the extended Boolean model, while properties of the model are discussed in [495]. The generalized vector space model was introduced in 1985 by Wong, Ziarko, and Wong [832, 831]. Latent semantic indexing was introduced in 1988 by Furnas, Deerwester, Dumais, Landauer, Harshman, Streeter, and Lochbaum [287]. In a subsequent paper, Bartell, Cottrell, and Belew [62] show that latent semantic indexing can be interpreted as a special case of multidimensional scaling.

Regarding neural network models for information retrieval, our discussion in this book is based mainly on the work by Wilkinson and Hingston [815]. But we also benefited from the writings of Kwok on the subject and related topics [466, 467, 469, 468].

The fuzzy set model (for information retrieval) covered in this book is due to Ogawa, Morita, and Kobayashi [616]. The utilization of fuzzy theory in information retrieval goes back to the 1970s with the work of Radecki [658, 659, 660, 661], of Sachs [691], and of Tahani [755]. Bookstein [102] proposes the utilization of fuzzy operators to deal with weighted Boolean searches. Kraft and Buel [461] utilize fuzzy sets to generalize a Boolean system. Miyamoto, Miyake, and Nakayama [567] discuss the generation of a pseudothesaurus using co-occurrences and fuzzy operators. Subsequently, Miyamoto and Nakayama [568] discuss the utilization of this thesaurus with information retrieval systems.

Our discussion on structured text is based on the survey by [41]. Another survey of interest (an older one though) is the work by MacLeod [533]. Burkowski [132, 133] proposed a model based on non-overlapping regions. Clarke, Cormack, and Burkowski [173] extended this model with overlapping capabilities. The model based on proximal nodes was proposed by Navarro and Baeza-Yates [589, 590]. In [534], MacLeod introduced a model based on a single hierarchy which also associates attributes with nodes in the hierarchy (for database-like querying) and hypertext links with pairs of nodes. Kilpelainen and Mannila [439] discuss the retrieval from hierarchical texts through the specification of partial patterns. In [183], Consens and Milo discuss algebras for querying text regions.

A classic reference on hypertexts is the book by Nielsen [604]. Another popular reference is the book by Shneiderman and Kearsley [727]. Conklin [181] presents an introductory survey of the area. The *Communications of the ACM* dedicated an special edition [177] to hypermedia which discusses in detail the Dexter model — a reference standard on the terminology and semantics of basic hypermedia concepts. A subsequent edition [178] was dedicated to the presentation of various models for supporting the design of hypermedia applications.