Euclidean distance, weighted Euclidean distance, Manhattan distance, and so forth. It is worth keeping in mind that while these standard distance functions can be extremely useful, they are primarily mathematical constructs and, as such, may not necessarily match our human intuition about similarity. This will be particularly relevant when we discuss similarity in the context of data types such as text and images, where the retrieval performance of humans based on semantic content can be difficult to match using algorithms based on general domain-independent distance functions.
In section 14.2 we discuss a subtle issue: how to objectively evaluate the performance of a specific retrieval algorithm. The evaluation is significantly complicated by the fact that the ultimate judgment of performance comes from the subjective opinion of the user issuing the query, who determines whether the retrieved data is relevant.
For structured data (such as sequences, images, and text), solving the retrieval by content problem has an additional aspect, namely, determining the *representation* used for calculation of the similarity measure. For example, it is common to use color, texture, and similar features in representing images, and to use word counts in representing text. Such abstractions typically involve significant loss of information such as local context. Yet they are often essential, due to the difficulty of defining meaningful similarity measures at the pixel or ascii character level (for images and text respectively). Section 14.3 discusses retrieval by content for text data, focusing in particular on the vector-space representation. Algorithms for matching queries to documents, latent semantic indexing, and document classification are all discussed in this context. Section 14.4 introduces the topics of relevance feedback and automated recommender systems for modeling human preferences for one object over another. In section 14.5 we discuss representation and retrieval issues in image retrieval algorithms. General image retrieval is a difficult problem, and we will look at both the strengths and limitations of current approaches, in particular the issue of invariance. Section 14.6 reviews basic concepts in time series and sequence matching. As the one-dimensional analog of image retrieval, similar representational and invariance issues arise for sequential data as for image data. Section 14.7 and section 14.8 contain a summary overview and discussion of further reading, respectively.

14.2 Evaluation of Retrieval Systems

14.2.1 The Difficulty of Evaluating Retrieval Performance

In classification and regression the performance of a model can always be judged in an objective manner by empirically estimating the accuracy of the model (or more generally its loss) on unseen test data. This makes comparisons of different models and algorithms straightforward.

For retrieval by content, however, the problem of evaluating the performance of a particular retrieval algorithm or technique is more complex and subtle. The primary difficulty is that the ultimate measure of a retrieval system's performance is determined by the usefulness of the retrieved information to the user. Thus, retrieval performance in a real-world situation is inherently subjective (again, in contrast to classification or regression). Retrieval is a human-centered, interactive process, which makes performance evaluation difficult. This is an important point to keep in mind.
Although it may be very difficult to measure how useful a particular retrieval system is to an average user directly, there are nonetheless some relatively objective methods we can use if we are willing to make some simplifications. First we assume that (for test purposes) objects can be labeled as being relevant or not, relative to a particular query. In other words, for any query Q, it will be assumed that there exists a set of binary classification labels of all objects in the data set indicating which are relevant and which are not. In practice, of course, this is a simplification, since relevance is not necessarily a binary concept; e.g., particular articles among a set of journal articles can be individually more or less relevant to a particular student's research questions. Furthermore, this methodology also implicitly assumes that relevance is absolute (not user-centric) in the sense that the relevance of any object is the same for any pair of users, relative to a given query Q. Finally, it is assumed that somehow the objects have been labeled,

presumably by a relatively objective and consistent human judge. In practice, with large data sets, getting such relevance judgments can be a formidable task.
Note in passing that one could treat the retrieval problem as a form of a classification task, where the class label is dependent on the query Q , i.e., "relevant or not to the query Q" and where the objects in the database are having their class labels estimated relative to Q. However, the retrieval problem has some distinguishing characteristics which make worthwhile to treat independently from classification. Firstly, the definition of the class variable is in the hands of the user (since the user defines the query Q) and can change every time the system is used. Secondly, the primary goal is not so much to classify all the objects in the database, but instead to return the set of most relevant objects to the user.

14.2.2 Precision versus Recall

Despite the caveats mentioned above, the general technique of labeling the objects in a large data set as being relevant or not (relative to a given set of predefined queries) is nonetheless quite useful in terms of providing a framework for objective empirical performance evaluation of various retrieval algorithms. We will discuss the issue of labeling in more detail in section 14.2.3. One practical approach is to use a committee of human experts to classify objects as relevant or nonrelevant.
Assume that we are evaluating the performance of a specific retrieval algorithm in response to a specific query Q on an independent test data set. The objects in the test data have already been preclassified as either relevant or irrelevant to the query Q. It is assumed that this test data set has not been used to tune the performance of this retrieval algorithm (otherwise the algorithm could simply memorize the mapping from the given query Q to the class labels). We can think of the retrieval algorithm as simply classifying the objects in the data set (in terms of relevance relative to Q), where the true class labels are hidden from the algorithm but are known for test purposes.
If the algorithm is using a distance measure (between each object in the data set and Q) to *rank* the set of objects, then the algorithm is typically parametrized by a threshold T. Thus, K_T objects will be returned by the algorithm, the ordered list of K_T objects that are closer than threshold T to the query object Q. Changing this threshold allows us to change the performance of the retrieval algorithm. If the threshold is very small, then we are being conservative in deciding which of the objects we classify as being relevant. However, we may miss some potentially relevant objects this way. A large threshold will have the opposite effect: more objects returned, but a potentially greater chance that (in truth) they are not relevant.
Suppose that in a test data set with N objects, a retrieval algorithm returns K_T objects of potential relevance. The performance of the algorithm can be summarized by table 14.1, where $N = TP + FP + FN + TN$ is the total number of labeled objects, $TP + FP = K_T$ is the number of objects returned by the algorithm, and $TP + FN$ is the total number of relevant objects. *Precision* is defined as the fraction of retrieved objects that are relevant, i.e., $TP/(TP + FP)$. *Recall* is defined as the proportion of relevant objects that are retrieved relative to the total number of relevant objects in the data set, i.e., $TP/(TP + FN)$. There is a natural trade-off here. As the number of returned objects K_T is increased (i.e., as we increase the threshold and allow the algorithm to declare more objects to be relevant) we can expect recall to increase (in the limit we can return all objects, in which case recall is 1), while precision can be expected to decrease (as K_T is increased, it will typically be more difficult to return only relevant objects). If we run the retrieval algorithm for different values of the threshold T, we will obtain a set of pairs of (recall, precision) points. In turn these can be plotted providing a recall-precision characterization of this particular retrieval algorithm (relative to the query Q, the particular data set, and the labeling of the data). In practice, rather than evaluating performance relative to a single query Q, we estimate the average recall-precision performance over a set of queries (see figure 14.1 for an example). Note that the recall-performance curve is essentially equivalent (except for a relabeling of the axes) to the well-known receiver-operating characteristic (ROC) used to characterize the performance of binary classifiers with variable thresholds.

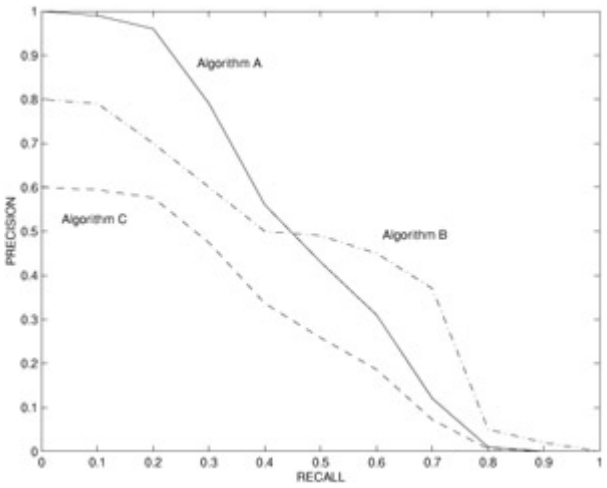

Figure 14.1: A Simple (Synthetic) Example of Precision-Recall Curves for Three Hypothetical Query Algorithms. Algorithm A Has the Highest Precision for Low Recall Values, While Algorithm B Has the Highest Precision for High Recall Values. Algorithm C is Universally Worse Than A or B, But We Cannot Make a Clear Distinction between A or B Unless (For Example) We were to Operate at a Specific Recall Value. The Actual Recall-Precision Numbers Shown are Fairly Typical of Current Text-Retrieval Algorithms; e.g., the Ballpark Figures of 50% Precision at 50% Recall are Not Uncommon Across Different Text-Retrieval Applications.

Table 14.1: A Schematic of the Four Possible Outcomes in a Retrieval Experiment Where Documents are Labeled as Being "Relevant" or "Not Relevant" (Relative to a Particular Query *Q*). The Columns Correspond to Truth and Rows Correspond to the Algorithm's Decisions on the Documents. *TP, FP, FN, TN* Refer to True Positive, False Positive, False Negative, And True Negative Respectively, Where Positive/Negative Refers to the Relevant/Nonrelevant Classification Provided by the Algorithm. A Perfect Retrieval Algorithm Would Produce a Diagonal Matrix with *FP* = *FN* = 0. This Form of Reporting Classification Results is Sometimes Referred to as a *Confusion Matrix*.

	Truth: Relevant	**Truth: Not-Relevant**
Algorithm: Relevant	*TP*	*FP*
Algorithm: Not Relevant	*FN*	*TN*

Now consider what happens if we plot the recall-precision of a set of different retrieval algorithms relative to the same data set and set of queries. Very often, no one curve will dominate the others; i.e., for different recall values, different algorithms may be best in terms of precision (see figure 14.1 for a simple example). Thus, precision-recall curves do not necessarily allow one to state that one algorithm is in some sense better than another. Nonetheless they can provide a useful characterization of the relative and absolute performance of retrieval algorithms over a range of operating conditions. There are a number of schemes we can use to summarize precision-recall performance with a single number, e.g., precision at some fixed number of documents retrieved, precision at the point where recall and precision are equal, or average precision over multiple recall levels.

14.2.3 Precision and Recall in Practice

Precision-recall evaluations have been particularly popular in text retrieval research, although in principle the methodology is applicable to retrieval of any data type. The Text Retrieval Conferences (TREC) are an example of a large-scale precision-recall evaluation experiment, held roughly annually by the U.S. National Institute of Standards and Technology (NIST). A number of gigabyte-sized text data sets are used, consisting of roughly 1 million separate documents (objects) indexed by about 500 terms on

average. A significant practical problem in this context is the evaluation of relevance, in particular determining the total number of relevant documents (for a given query *Q*) for the calculation of recall. With 50 different queries being used, this would require each human judge to supply on the order of 50 million class labels! Because of the large number of participants in the TREC conference (typically 30 or more), the TREC judges restrict their judgments to the set consisting of the union of the top 100 documents returned by each participant, the assumption being that this set typically contains almost all of the relevant documents in the collection. Thus, only a few thousand relevance judgments need to be made rather than tens of millions.

More generally, determining recall can be a significant practical problem. For example, in the retrieval of documents on the Internet it can be extremely difficult to accurately estimate the total number of potentially available relevant documents. Sampling techniques can in principle be used, but, combined with the fact that subjective human judgment is involved in determining relevance in the first place, precision-recall experiments on a large-scale can be extremely nontrivial to carry out.

14.3 Text Retrieval

Retrieval of text-based information has traditionally been termed information retrieval (IR) and has recently become a topic of great interest with the advent of text search engines on the Internet. Text is considered to be composed of two fundamental units, namely the *document* and the *term*. A *document* can be a traditional document such as a book or journal paper, but moregenerally is used as a name for any structured segment of text such as chapters, sections, paragraphs, or even e-mail messages, Web pages, computer source code, and so forth. A *term* can be a word, word-pair, or phrase within a document, e.g., the word `data` or word-pair `data mining`.
Traditionally in IR, *text queries* are specified as sets of terms. Although documents will usually be much longer than queries, it is convenient to think of a single representation language that we can use to represent both documents and queries. By representing both in a unified manner, we can begin to think of directly computing distances between queries and documents, thus providing a framework within which to directly implement simple text retrieval algorithms.

14.3.1 Representation of Text

As we will see again with image retrieval later in this chapter, much research in text retrieval focuses on finding general *representations* for documents that support both

- the capability to retain as much of the semantic content of the data as possible, and
- the computation of distance measures between queries and documents in an efficient manner.

A user who is using a text retrieval system (such as a search engine on the Web) wants to retrieve documents that are relevant to his or her needs in terms of semantic content. At a fundamental level, this requires solving a long-standing problem in artificial intelligence, namely, *natural language processing* (NLP), or the ability to program a computer to "understand" text data in the sense that it can map the ascii letters of the text into some well-defined semantic representation. In its general unconstrained form, this has been found to be an extremely challenging problem. *Polysemy* (the same word having several different meanings) and *synonmy* (several different ways to describe the same thing) are just two of the factors that make automated understanding of text rather difficult. Thus, perhaps not surprisingly, NLP techniques (which try to explicitly model and extract the semantic content of a document) are not the mainstay of most practical IR systems in use today; i.e., practical retrieval systems do not typically contain an explicit model of the meaning of a document.
Instead, current IR systems typically rely on simple term matching and counting techniques, where the content of a document is implicitly and approximately captured (at least in theory) by a vector of term occurrence counts. Assume that a set of terms t_j for retrieval, $1 = j = T$, has been defined a priori. The size of this set can be quite large (e.g., T = 50,000 terms or more).
Each individual document D_i, $1 = i = N$, is then represented as a *term vector*:

(14.1) $$D_i = (d_{i1}, d_{i2}, \ldots, d_{iT})$$

where d_{ij} represents some information about the occurrence of the *j*th term in the *i*th document. The individual d_{ij}s are referred to as *term weights* (although, to be more precise, they are just the component values of the term vector).
In the *Boolean representation*, the term weights simply indicate whether certain terms appear in the document, i.e., $d_{ij} = 1$ if document *i* contains term *j*, and $d_{ij} = 0$ otherwise. In the *vector space representation* each term weight can be some real-valued number, e.g., a function of how often the term appears in the document, or (perhaps) the relative frequency of that term in the overall set of documents. We will look in more detail at term weights in section 14.3.2.
Note that when a document is represented as a *T*-dimensional term vector, not only has the word order of the original document been lost, but so also has syntactic information such as sentence structure. Despite this loss of information, term vectors can nonetheless be quite useful and effective in a variety of retrieval applications.

As an example, consider a very simple toy example with 10 documents and 6 terms, where the terms are

- t1 = `database`
- t2 = `SQL`
- t3 = `index`
- t4 = `regression`
- t5 = `likelihood`
- t6 = `linear`,

and we have a 10 × 6 document-term frequency matrix *M* as shown in table 14.2. Entry *ij* (row *i*, column *j*) contains the number of times term *j* is contained in document *i*. We can clearly see that the first five documents *d*1 to *d*5 contain mainly database terms (combinations of `query, SQL, index`) while the last five documents *d*6 to *d*10 contain mainly regression terms (combinations of `regression, likelihood, linear`). We will return to this example later in this chapter.

Table 14.2: A Toy Document-Term Matrix for 10 Documents and 6 Terms. Each *ij*th Entry Contains the Number of Times that Term *j* Appears in Document *i*.

	t1	t2	t3	t4	t5	t6
d1	24	21	9	0	0	3
d2	32	10	5	0	3	0
d3	12	16	5	0	0	0
d4	6	7	2	0	0	0
d5	43	31	20	0	3	0
d6	2	0	0	18	7	16
d7	0	0	1	32	12	0
d8	3	0	0	22	4	2
d9	1	0	0	34	27	25
d10	6	0	0	17	4	23

Given a particular vector-space representation, it is straightforward to define distance between documents as some simple well-defined function of distance between their document vectors. Most of the various distance measures we described in chapter 2 can be (and have been) used for document comparison. One widely used distance measure in this context is the *cosine distance*, which is defined as

(14.2) $$d_c(D_i, D_j) = \frac{\sum_{k=1}^{T} d_{ik} d_{jk}}{\sqrt{\sum_{k=1}^{T} d_{ik}^2 \sum_{k=1}^{T} d_{jk}^2}}.$$

This is the cosine of the angle between the two vectors (equivalently, their inner product after each has been normalized to have unit length) and, thus, reflects similarity in terms of the *relative* distribution of their term components. There is nothing particularly special about this distance measure, however, it has been found historically to be quite effective in practical IR experiments.

Figure 14.2 shows in pixel form the distance matrices for the toy document-term frequency matrix of table 14.2. Both the Euclidean distance and cosine distance matrices are shown. For both distance matrices it is clear that there are two clusters of documents, one for database documents and one for regression, which show up as two relatively light subblocks in the figure. Conversely, the pairwise distances between elements of the two groups are relatively large (i.e., dark pixels). However, the cosine distance produces a better separation of the two groups. For example, documents 3 and 4 (in the database cluster) would be ranked as being closer to documents 6, 8, and 9 (which are documents about regression) than to document 5 (which is another document about databases). This happens simply because the term vectors for documents 3 and 4 (and 6, 8, and 9) are close to the origin compared to document 5. By using angle-based distances, the cosine distance emphasizes the relative contributions of individual terms, resulting in the better separation evident in figure 14.2 (bottom).

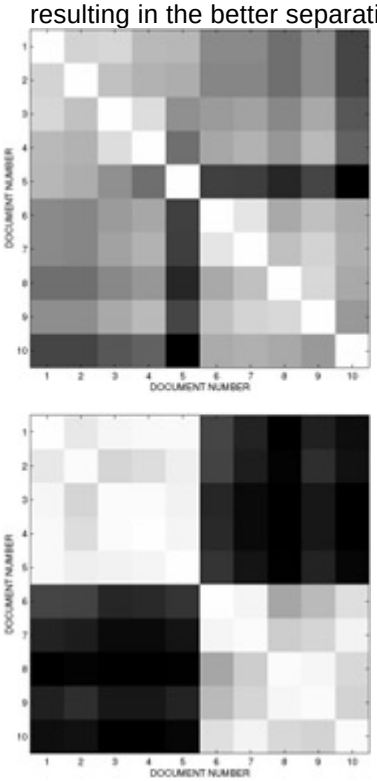

Figure 14.2: Pairwise Document Distances for the Toy Document-Term Matrix in the Text, For Euclidean Distance (Top) and Cosine Distance (Bottom). Brighter Squares are More Similar and Darker Squares are Less Similar, According to the Relevant Distance. For the Euclidean Distance, White Corresponds to Zero (e.g., On the Diagonals) and Black is the Maximum Distance between any Two Documents. For the Cosine Distance Brighter Pixels Correspond to Larger Cosine Values (Closer Angles) and Darker Pixels Correspond to Smaller Cosine Values (Larger Angles).

Each vector D_i can be thought of as a *surrogate* document for the original document. The entire set of vectors can be represented as an $N \times T$ matrix. Typically this matrix is very sparse, for example with only about 0.03% of cells being nonzero for the TREC collections mentioned earlier. A natural interpretation of this matrix is that each row D_i (a document) of this matrix is a vector in T-dimensional "term space." Thus, thinking in terms of the data matrix we have used to described data sets in earlier chapters, the documents play the role of individual objects, the terms play the role of variables, and the vector entries represent "measurements" on the documents.

In a practical implementation of a text-retrieval system, due to the sparsity of the term-document matrix, the original set of text documents are represented as an *inverted file* structure (rather than as a matrix directly), i.e., a file indexed by the T terms where each

term t_j points to a list of N numbers describing term occurrence (the d_{ij}, j fixed) for each document.

The process of generating a term-document matrix can by itself be quite nontrivial, including issues such as how to define terms, e.g., are plural and singular nouns counted as the same term? should very common words be used as terms? and so forth. We will not dwell on this issue except to point out that clearly there can be a substantial amount of "engineering" required at this level.

14.3.2 Matching Queries and Documents

Queries can also be expressed using the same term-based representation as that used for documents. In effect, the query is itself represented as a document, albeit one that typically contains very few terms (although we can, of course use a real document as a query itself, e.g., "Find documents that are similar to this one").

For the Boolean representation, a query is represented as a logical Boolean function on a subset of the available terms. Thus, for example, a typical query might be `data AND mining AND NOT(coal)`. The basic mechanism for retrieval in this context consists of scanning the inverted file to determine which documents *exactly* match the query specifications. Extensions of the basic Boolean query language are possible, such as adding weights to indicate the relative importance of certain terms over others. However, a general drawback of the Boolean representation is that there is no natural semantics for the notion of *distance* between queries and documents, and thus, no natural way to perform ranking of documents based on relevance. In addition, and perhaps somewhat surprisingly, humans often have great difficulty constructing Boolean queries that reflect precisely what they intend. Nonetheless, despite these drawbacks, the Boolean query method is popular in practical IR systems because of its efficiency and relative simplicity.

In the vector-space representation, a query can be expressed as a vector of weights. Terms not in the query are implicitly assigned zero weight. The simplest form of query assigns unit weight to each term in the query. More generally, individual weights can be assigned by the user to indicate the relative importance of each term (typically the weights are constrained to be between 0 and 1). In practice, it can be difficult for a user to know how to set the weights to effectively model his or her notion of relevance. Later we will see that a scheme called *relevance feedback* can be used to iteratively refine the weights over the course of multiple queries, but for now we will assume that the user provides the query as well as the weights for the terms in query.

Let the $Q = (q_1, \ldots, q_T)$ be a vector of query weights. In the simplest approach the query weights can be simply either 1 (the term is in the query) or 0 (the term is not in the query), or the same weighting scheme as used for document representation can be used (see below). As an example of the simple binary scheme consider three queries, each consisting of the single terms `database, SQL, regression`. For our toy example these three queries would be represented as three vectors: (1, 0, 0, 0, 0, 0), (0, 1, 0, 0, 0, 0), and (0, 0, 0, 1, 0, 0). Using the cosine distance to match these three queries to our toy example data set of table 14.2 results in documents *d*2, *d*3, and *d*9 being ranked as closest, respectively.

To discuss more general concepts in matching queries to documents, we must first revisit briefly the idea of weights in the vector-space model. Let d_{ik} be the weight (or component value) for the kth term, $1 = k = T$ in document D_i. There have been many different (largely ad hoc) suggestions in the IR literature on how these weights should be set to improve retrieval performance. Ideally these terms should be chosen so that more relevant documents are ranked higher than less relevant ones. The Boolean approach of setting the weights to 1 if the term occurs anywhere in the document has been found to favor large documents (over relevant ones) simply because a larger documents is more likely to include a given query term somewhere in the document.

One particular weighting scheme, known as the TF-IDF weighting, has proven very useful in practice. TF stands for *term frequency* and simply means that each term component in a term vector is multiplied by the frequency by which that term occurs in the document. This has the effect of increasing the weight on terms that occur frequently in a given document. The toy document-term matrix of table 14.2 is expressed in TF form.

However, if a term occurs frequently in many documents in the document set, then using TF weights for retrieval may have little discriminative power, i.e., it will increase recall but may have poor precision. The *inverse-document-frequency* (IDF) weight helps to improve discrimination. It is defined as $\log(N/n_j)$, i.e., the log of the inverse of the fraction of documents in the whole set that contain term j, where n_j is the number of documents containing term j, and the total number of documents is N. The IDF weight favors terms that occur in relatively few documents; i.e., it has discriminative power. The use of the logarithm of IDF rather than IDF directly is motivated by the desire to make the weight relatively insensitive to the exact number of documents N.

The TF-IDF weight is simply the product of TF and IDF for a particular term in a particular document. As with the cosine distance measure (with which it is often used), there is no particularly compelling motivation for defining weights in this manner. However, it has been found to be generally superior in terms of precision-recall performance when it is compared to alternative weighting schemes. There are various enhancements to the basic TF-IDF method, but TF-IDF weighting as described above is still usually considered the default baseline method in evaluation experiments.
The same TF-IDF weights derived from the set of documents can be used to weight query terms as well. Another alternative for query-weighting is to use just IDF weights to emphasize query terms that are relatively rare. For example, if we were to issue the query `Richard Nixon` we would probably be happier to retrieve documents containing `Nixon` and not `Richard` than vice versa.
The document-term matrix of table 14.2 produces the following set of IDF weights (using natural logs): (0.105, 0.693, 0.511, 0.693, 0.357, 0.693). Note, for example, that the first term *database* now receives a lower weight than the other terms, because it appears in more documents (i.e., is less discriminative). This results in the TF-IDF document-term matrix shown in table 14.3.

Table 14.3: TF-IDF Document-Term Matrix Resulting From Table 14.2

2.53	14.56	4.60	0	0	2.07
3.37	6.93	2.55	0	1.07	0
1.26	11.09	2.55	0	0	0
0.63	4.85	1.02	0	0	0
4.53	21.48	10.21	0	1.07	0
0.63	0	0	11.78	1.42	15.94
0.21	0	0	22.18	4.28	0
0.31	0	0	15.24	1.42	1.38
0.10	0	0	23.56	9.63	17.33

A classic approach to matching queries to documents is as follows:

- represent queries as term vectors with 1s for terms occurring in the query and 0s everywhere else,
- represent term-vectors for documents using TF-IDF weights for the vector components, and
- use the cosine distance measure to rank the documents in terms of distance to the query.

Table 14.4 shows a simple example of a query comparing the TF and TF-IDF methods. Note that, unlike the exact match retrieval results of the Boolean case (where all matching documents are returned), here the distance measure produces a ranking of all documents that include at least one relevant term.

Table 14.4: Distances Resulting From a Query Containing the Terms *Database* and *Index*, i.e., Q = (1, 0, 1, 0, 0, 0), for the Document-Term Matrix of Table 14.2, Using the Cosine Distance Measure. Using the TF Matrix, Document *d*5 is Closest; For the TF-IDF Matrix, *d*2 is Closest.

Document	TF distance	TF-IDF distance
d1	0.70	0.32
d2	0.77	0.51
d3	0.58	0.24
d4	0.60	0.23
d5	0.79	0.43
d6	0.14	0.02
d7	0.06	0.01
d8	0.02	0.02
d9	0.09	0.01
d10	0.01	0.00

14.3.3 Latent Semantic Indexing

In the schemes we have discussed so far for text retrieval, we have relied exclusively on the notion of representing a document as a T-dimensional vector of term weights. A criticism of the term-based approach is that users may pose queries using different terminology than the terms used to index a document. For example, from a term similarity viewpoint the term `data mining` has nothing directly in common with the term `knowledge discovery`. However, *semantically*, these two terms have much in common and (presumably) if we posed a query with one of these terms, we would consider documents containing the other to be relevant. One solution to this problem is to use a knowledge base (a thesaurus or ontology) that is created a priori with the purpose of linking semantically related terms together. However, such knowledge bases are inherently subjective in that they depend on a particular viewpoint of how terms are related to semantic content.

An interesting, and useful, alternative methodology goes by the name of *latent semantic indexing* (LSI). The name suggests that hidden semantic structure is extracted from text rather than just term occurrences. What LSI actually does is to approximate the original T-dimensional term space by the first k principal component directions in this space, using the $N \times T$ document-term matrix to estimate the directions. As discussed in chapter 3, the first k principal component directions provide the best set of k orthogonal basis vectors in terms of explaining the most variance in the data matrix. The principal components approach will exploit redundancy in the terms, if it exists. Frequently in practice there is such redundancy. For example, terms such as `database, SQL, indexing, query optimization` can be expected to exhibit redundancy in the sense that many database-related documents may contain all four of these terms together. The intuition behind principal components is that a single vector consisting of a weighted combination of the original terms may be able to approximate quite closely the effect of a much larger set of terms. Thus the original document-term matrix of size $N \times T$ can be replaced by a matrix of size $N \times k$, where k may be much smaller than T with little loss in information. From a text retrieval perspective, for fixed recall, LSI can increase precision compared to the vector-space methods discussed earlier.

An interesting aspect of the the principal component representation for the document-term matrix is that it captures relationships among terms by creating new terms that may more closely reflect the semantic content of the document. For example, if the terms `database, SQL, indexing, query optimization` are effectively combined into a single principal component term, we can think of this new term as defining whether the content of a document is about database concepts. Thus, for example, if the query is posed using the term `SQL`, but the database-related documents in the set of documents

contain only the term `indexing`, that set of database documents will nonetheless be returned by the LSI method (but would not be returned by a strictly term-based approach).

We can calculate a singular-value decomposition (SVD) for the document-term matrix **M** in table 14.2. That is, we find a decomposition **M** = **USV**^{T}. Here **U** is a 10×6 matrix where each row is a vector of weights for a particular document, **S** is a 6 × 6 diagonal matrix of eigenvalues for each principal component direction, and the columns of the 6 × 6 matrix **V**^{T} provide a new orthogonal basis for the data, often referred to as the principal component directions.

The **S** matrix for **M** has diagonal elements

$$?_1, ..., ?_6 = \{77.4, 69.5, 22.9, 13.5, 12.1, 4.8\}.$$

In agreement with our intuition, most of the variance in the data is captured by the first two principal components. In fact, if we were to retain only these two principal components (as two surrogate terms instead of the six original terms), the fraction of variance that our two-dimensional representation retains is $(\lambda_1^2+\lambda_2^2)/\sum_{i=1}^{6}\lambda_i^2 = 0.925$; i.e., only 7.5% of the information has been lost (in a mean-square sense). If we represent the documents in the new two-dimensional principal component space, the coefficients for each document correspond to the first two columns of the **U** matrix:

d1	30.8998	-11.4912
d2	30.3131	-10.7801
d3	18.0007	-7.7138
d4	8.3765	-3.5611
d5	52.7057	-20.6051
d6	14.2118	21.8263
d7	10.8052	21.9140
d8	11.5080	28.0101
d9	9.5259	17.7666
d10	19.9219	45.0751

and we can consider these two columns as representing new *pseudo-terms* that are in effect linear combinations of the original six terms.

It is informative to look at the first two principal component directions:

(14.3) $\mathbf{v}_1 = (0.74, 0.49 0.27, 0.28, 0.18, 0.19)$

(14.4) $\mathbf{v}_2 = (-0.28, -0.24, -0.12, 0.74, 0.37, 0.31).$

These are the two directions (a plane) in the original six-dimensional term space where the data is most "spread out" (has the most variance). The first direction emphasizes the first two terms (*query, SQL*) more than the others: this is in effect the direction that characterizes documents relating to databases. The second direction emphasizes the last three terms *regression, likelihood, linear*, and can be considered the direction that characterizes documents relating to regression. Figure 14.3 demonstrates this graphically. We can see that the two different groups of documents are distributed in two different directions when they are projected onto the plane spanned by the first two principal component directions. Note that document 2 is almost on top of document 1, somewhat obscuring it. (Symbols D1 and D2 are discussed below.) The distances of the points from the origin reflect the magnitude of the term-vector (i.e., number of terms) for each document. For example, documents 5 and 10 have the most terms and are furthest from the origin. We can see here that the angle difference between documents is clearly a very useful indicator of similarity, since regression and database documents are each clustered around two different angles in the plane.

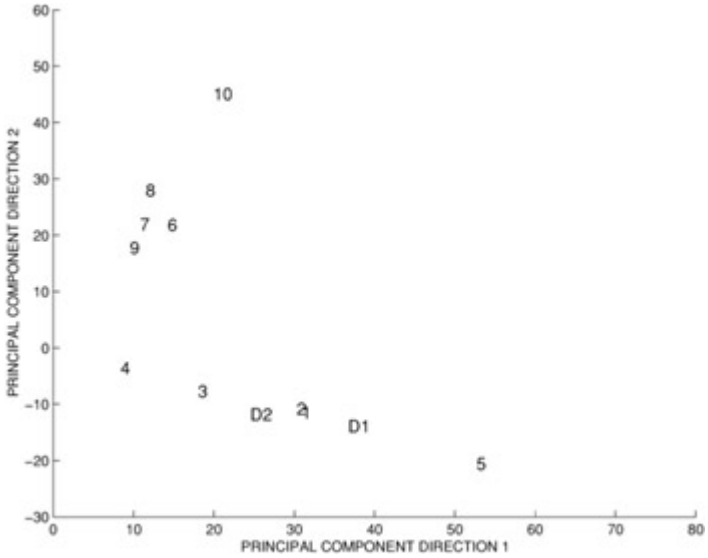

Figure 14.3: Projected Locations of the 10 Documents (From Table 14.2) in the Two Dimensional Plane Spanned by the First Two Principal Components of the Document-Term Matrix *M*.

An advantage of principal components in this context can be seen by considering a new document D1 which contains, for example, the term *database* 50 times and another document D2, which contains the term *SQL* 50 times, and both documents contain none of the other terms. In a direct keyword representation, these two documents would not be considered similar since they contain no common terms (among the specific terms used in our toy example). However, if we instead represent D1 and D2 using the two principal component terms, they are projected into this space, as shown in figure 14.3, i.e., both are projected along the "database" direction even though each is missing two of the three terms associated with databases. The principal component approach has implicitly modeled the interrelationship between terms. This feature can be taken advantage of for querying. Imagine now that we pose a query using only the term *SQL*. If we use the principal component representation (e.g., the first two pseudo-terms) for our documents, we can also convert the query into the pseudo-term representation, where again the query will be closer (in angle) to the database documents than the regression documents, allowing retrieval of documents that do not contain the term *SQL* at all but are related in content.

From a computational viewpoint, directly computing the principal component vectors (by seeking the eigenvectors of the correlation or covariance matrix for example) is usually neither computationally feasible or numerically stable. In practice, the PCA vectors can be estimated using special-purpose sparse SVD techniques that are well-suited for high-dimensional sparse matrices.

There are many other variations of this basic framework outline. The application of principal components for text retrieval is often referred to as latent semantic indexing (LSI). Use of this general technique has been shown in certain cases to improve retrieval performance in systematic tests, based on its ability to match documents and queries with no terms in common.
We can also model the document-term matrix in a probabilistic manner as being generated by a mixture of simpler component models, where each component represents the distribution of words we would expect to see conditioned on a particular topic. Each of the component models can be (for example) conditional independence (naive Bayes) models or multinomials, and the EM algorithm for fitting mixtures, as described in chapter 9, can be applied directly and rather straightforwardly.

14.3.4 Document and Text Classification

It is clear from our discussion that the term-vector representation of documents provides a natural framework for classifying documents, where we can apply either the supervised classification framework of chapter 10 for documents that have labels preassigned, or

the unsupervised learning (clustering) frameworks of chapter 9 for unlabeled documents. For example, a practical application of these ideas is that of automatically and accurately clustering Web documents into groups or taxonomies, for updating and maintaining the databases of large Web search engines.

Because of the fact that typical term-vectors are very high-dimensional (e.g., on the order of 10,000 or more terms is quite common), classifiers that are both accurate and efficient in high-dimensional spaces are usually the methods of choice in this context. For example, although classification trees can be useful for high-dimensional problems in general, for document classification the individual features (individual terms) may not be so informative. For documents in the class "sports" it is more likely to be the case that such documents contain some subset of words such as "score," "field," "stadium," "win," etc., rather than always containing any particular single word from this set. Thus, classification models such as the first-order Bayes classifier (naive Bayes) or linear combinations of weights (such as linear support vector machines) tend to work well for document representations, since they can combine evidence from many different features in a relatively simple (e.g., linear) manner. A feedforward neural network would not be a practical choice for most document modeling problems by virtue of the fact that it would be extremely complex, from the point of view of both the number of parameters in the model and the amount of time that it would take to train it.

Numerous interesting issues arise in the area of document classification that we will not dwell on here. For example, it makes sense to consider each document as belonging to multiple topics (classes) rather than just to a single class. Thus, instead of the usual framework where the class is considered to consist of mutually exclusive and exhaustive categories, for documents the classes need not be mutually exclusive. There are a number of different approaches to dealing with this "multiple membership" problem. One simple method is to train a binary classifier for each class separately, a method that is likely to be practical only if the total number of classes is relatively small.

14.4 Modeling Individual Preferences

14.4.1 Relevance Feedback

As mentioned earlier, retrieval algorithms tend to have a much more interactive flavor than most of the other data mining algorithms we have discussed in this book. In particular, a user with a particular query Q may be willing to iterate through a few sets of different retrieval "trials" by the algorithm, and provide user-based feedback to the algorithm by labeling the returned documents as relevant or nonrelevant from the user's viewpoint. We can use this idea with any retrieval system, not just text retrieval, but we will limit our discussion to text to keep things specific.

Rocchio's algorithm is particularly widely used in this context. The general idea is that relevance is essentially user-centric, i.e., if the user could (in theory) see all of the documents, he or she could in principle separate them into two sets, relevant R and irrelevant NR. Given these two sets, it can be shown that the optimal query (using a vector model) can be written as

(14.5) $$Q_{optimal} = \frac{1}{|R|} \sum_{D \in R} D - \frac{1}{|NR|} \sum_{D \in NR} D.$$

where D denotes a term-vector representation for document whose labelings (by the user) are known.

In practice, of course, a particular user has not personally labeled all of the documents in the database. Instead the user starts out with a specific query $Q_{current}$, which can be presumed to be suboptimal relative to $Q_{optimal}$. The algorithm uses this initial query to return a small set of documents that are then labeled by the user as being relevant R' or not NR'. Rocchio's algorithm then refines the query in the following manner:

(14.6) $$Q_{new} = \alpha Q_{current} + \frac{\beta}{|R'|} \sum_{D \in R'} D - \frac{\gamma}{|NR'|} \sum_{D \in NR'} D.$$

Thus, the query is modified by (in effect) moving the current query toward the mean vector of the documents judged to be relevant and away from the mean vector of of the documents judged to be irrelevant. The parameters a, ß, and ? are positive constants (chosen heuristically) that control how sensitive the new query is to the most recent labeling of the documents, relative to the existing query vector $Q_{current}$. The process repeats; i.e., the new query Q_{new} is matched to the document set, and the user again labels the documents. Note that even if the initial query Q_0 is stated incorrectly by the user, the algorithm can in principle adapt and learn the implicit preferences of the user in terms of relevance. In principle, if the labelings at each iteration are consistent, Q_{new} gradually approaches $Q_{optimal}$.

Experimental evidence indicates that such user feedback does indeed improve precision-recall performance. In other words, incorporating feedback has been shown to be systematically effective for information retrieval. Of course, there are many details that must be specified to implement this approach in practice, such as the number of documents that should be presented to the user, the relative number of relevant and irrelevant documents to be used, the method used to choose the irrelevant documents, and so forth, leading to a large number of variations on this basic theme.

14.4.2 Automated Recommender Systems

Instead of modeling just the preferences of a single user, we can generalize to the case in which the database has information about multiple users and their preferences in terms of a large number of objects. The technique of *collaborative filtering* is a simple and well-known approach to leveraging this group information. For example, imagine you are interested in a certain musical group and buy a CD for this group at an online Web site. Several hundred other people also may have bought this particular CD, and it is likely that at least some of their general preferences in musical taste match yours. Thus, collaborative filtering in this context simply means that an algorithm running at the Web site can provide for you a list of other CDs purchased by people who bought the same CD as you. Clearly we can generalize the basic idea in various directions. For example, if we have a purchase history for each user and/or users are willing to provide more detailed information about their specific interests (in the form of user profiles), we can have a vector representation of each user, and the discussion earlier in the chapter about defining appropriate similarity metrics once again becomes quite relevant.

Collaborative filtering is in a sense an attempt to capture the expert judgment and recommendations of a large group, where the group is automatically selected to match the interests of a specific user. The algorithms generally work by first finding the subset of profiles of those users that are judged to be most similar to the target profile and then calculating a recommendation as a function of the properties of the matched set of profiles. The quality of the recommendation will depend on the quality and quantity of information that is known about each user and the size of the user database. The technique tends to work best with large numbers of users. Acquiring a large number of user profiles can be difficult in practice, since users are naturally reluctant to take the time to provide detailed personal information. Capturing users preferences by their actions (e.g., by what they buy, or by what Web pages they visit) is a less intrusive approach that implicitly tries to estimate user preferences, and it is commonly employed in Internet-based recommendation systems (for example). A common practical requirement (for example, in e-commerce applications) is that the recommendation must be generated by the algorithm in real time, e.g., in less than a second. If we have a very large database of users (e.g., order of millions), this can pose significant computational and data engineering challenges.

14.5 Image Retrieval

Image and video data sets are increasingly common, from the hobbyist who stores digital images of family birthdays to organizations such as NASA and military agencies that gather and archive remotely sensed images of the earth on a continuous basis. Retrieval by content is particularly appealing in this context as the number of images becomes

large. Manual annotation of images is time-consuming, subjective, and may miss certain characteristics of the image depending on the specific viewpoint of the annotator. A picture may be worth a thousand words, but which thousand words to use is a nontrivial problem!

Thus, there is considerable motivation to develop efficient and accurate query-by-content algorithms for image databases, i.e., to develop interactive systems that allow a user to issue queries such as "find the K most similar images to this query image" or "find the K images which best match this set of image properties." Potential applications of such algorithms are numerous: searching for similar diagnostic images in radiology, finding relevant stock footage for advertising and journalism, and cataloging applications in geology, art, and fashion.

14.5.1 Image Understanding

Querying image data comes with an important caveat. Finding images that are similar to each other is in a certain sense equivalent to solving the general image understanding problem, namely the problem of extracting semantic content from image data. Humans excel at this. However, several decades of research in pattern recognition and computer vision have clearly shown that the performance of humans in visual understanding and recognition is extremely difficult to replicate with computer algorithms. (There is a direct analogy here to the NLP problem mentioned earlier in the context of text understanding.) Specific problems, such as face recognition or detection of airport runways, can be be successfully tackled, but general-purpose image understanding systems are still far off on the research horizon. For example, an infant can quickly learn to classify animals such as dogs of all sizes, colors, and shapes (including cartoon figures) in arbitrary scenes, whereas such completely unconstrained recognition is beyond the capability of any current vision algorithm. This ability to extract semantic content from raw image data is still relatively unique to the brain. Thus, not surprisingly, most current methods for image retrieval rely on relatively low-level visual cues.

14.5.2 Image Representation

For retrieval purposes, the original pixel data in an image can be abstracted to a feature representation. The features are typically expressed in terms of primitives such as color and texture features. As with text documents, the original images are converted into a more standard data matrix format where each row (object) represents a particular image and each column (variable) represents an image feature. Such feature representations are typically more robust to changes in scale and translation than the direct pixel measurements, but nonetheless may be invariant to only small changes in lighting, shading, and viewpoint.
Typically the features for the images in the image database are precomputed and stored for use in retrieval. Distance calculations and retrieval are thus carried out in multi-dimensional feature space. As with text, the original pixel data is reduced to a standard $N \times p$ data matrix, where each image is now represented as a p-dimensional vector in feature space.

Spatial information can be introduced in a coarse manner by computing features in localized subregions of an image. For example, we could compute color information in each 32 × 32 subregion of a 1,024 × 1,024 pixel image. This allows coarse spatial constraints to be specified in image queries, such as "Find images that are primarily red in the center and blue around the edges."
As well as regular $m \times n$ pixel images of *scenes*, an image database can also contain specific images of *objects*, namely objects on a constant background (such as an image of a black chair on a white background). Thus, we can also extract primitive properties that are object-specific such as features based on color, size, and shape (geometry) of the object. Video images represent a further generalization of image data, where images (frames) are linked sequentially over time.
A well-known commercial example of a retrieval by content system for images is the *Query by Image Content* (QBIC) system developed by IBM researchers in the early

1990s. The system is based on the general ideas described in section 14.5, allowing users to interactively query image and video databases based on example images, user-entered sketches, color and texture patterns, object properties, and so forth. Queries are allowed on scenes, objects (parts of scenes), and sequences of video frames, or any combination of these. The QBIC system uses a variety of features and a variety of associated distance measures for retrieval:

- A three-dimensional color feature vector, spatially averaged over the whole image: the distance measure is simple Euclidean distance.
- *K*-dimensional color histograms, where the bins of the histogram can be selected by a partition-based clustering algorithm such as *K*-means, and *K* is application dependent. QBIC uses a Mahalanobis-type distance measure between color histogram vectors to account for color correlations.
- A three-dimensional texture vector consisting of features that measure coarseness/scale, directionality, and contrast. Distance is computed as a weighted Euclidean distance measure, where the default weights are inverse variances of the individual features.
- A 20-dimensional shape feature for objects, based on area, circularity, eccentricity, axis orientation, and various moments. Similarity is computed using Euclidean distance.

14.5.3 Image Queries

As with text data, the nature of the abstracted representation for images (i.e., the computed features) critically determines what types of query and retrieval operations can be performed. The feature representation provides a language for query formulation. Queries can be expressed in two basic forms. In query by example we provide a sample image of what we are looking for, or sketch the shape of the object of interest. Features are then computed for the example image, and the computed feature vector of the query is then matched to the precomputed database of feature vectors. Alternatively, the query can be expressed directly in terms of the feature representation itself, e.g., "Find images that are 50% red in color and contain a texture with specific directional and coarseness properties." If the query is expressed in terms of only a subset of the features (e.g., only color features are specified in the query), only that subset of features is used in the distance calculations.

Clearly, depending on the application, we can generalize to relatively sophisticated queries (given a feature representation), allowing (for example) various Boolean combinations of query terms. For image data, the query language can also be specialized to allow queries that take advantage of spatial relations (such as "Find images with object 1 above object 2") or sequential relations (such as "Find video sequences with soccer players taking shots at goal followed by images of players celebrating").

Representing images and queries in a common feature-vector form is quite similar to the vector-space representation for text retrieval discussed earlier. One important difference is that typically a feature for an image is a real number indicating (for example) a particular color intensity in a particular region of the image, while a term component in a term-vector is typically some form of weighted count of how often a term occurs in the document. Nonetheless, the general similarity of the retrieval-by-content problem for both problems has led to numerous applications of text retrieval techniques to image retrieval applications including (for example) the use of principal component analysis to reduce the dimensionality of the feature space and relevance feedback via Rocchio's algorithm to improve the image retrieval process.

14.5.4 Image Invariants

For retrieval by content with images, it is important to keep in mind that (with current techniques at least) we can realistically work only with a restricted notion of semantic content, based on relatively simple "low-level" measurements such as color, texture, and

simple geometric properties of objects. There are many common distortions in visual data such as translations, rotations, nonlinear distortions, scale variability, and illumination changes (shadows, occlusion, lighting). The human visual system is able to handle many of these distortions with ease. For example, a human can view two scenes of the same object from completely different angles, with different lighting, and at different distances, and still easily attach the same semantic content to the scene (e.g., "It is a scene of my house taken after 1995").
However, the methods for content retrieval described here typically are *not* invariant under such distortions. Changes in scale, illumination, or viewing angle will typically change the feature measurements such that the distorted version of a scene is in a very different part of the feature space, compared to the original version of the scene. In other words, the retrieval results will not be invariant to such distortions, unless such distortion invariance is designed into the feature representation. Again, typically, distortion-invariant feature representations are currently known only for constrained visual environments, such as linear transformations of rigid objects, but not (for example) for general nonlinear transformations of nonrigid objects.

14.5.5 Generalizations of Image Retrieval

To conclude our discussion of image retrieval, we note that the term *image* can be interpreted much more broadly than the implicit interpretation as an image of a real-world scene (typically generated by a camera) as we have described it. More generally, image data can be embedded within text documents (such as books or Web pages). Other forms of images can include handwritten drawings (or text or equations), paintings, line drawings (such as in architecture or engineering), graphs, plots, maps, and so forth. Clearly, retrieval in each of these contexts needs to be handled in an application-specific manner, although many of the general principles discussed earlier will still apply. Video data also provides further challenges and opportunities in terms of automated indexing and interactive querying. For example, for a television news organization such as CNN, the ability to search through an archive of video footage to detect certain types of images would be quite useful.

14.6 Time Series and Sequence Retrieval

The problem of efficiently and accurately locating patterns of interest in time series and sequence data sets is an important and nontrivial problem in a wide variety of applications, including diagnosis and monitoring of complex systems, biomedical data analysis, and exploratory data analysis in scientific and business time series. Examples include

- finding customers whose spending patterns over time are similar to a given spending profile;
- searching for similar past examples of unusual current sensor signals for real-time monitoring and fault diagnosis of complex systems such as aircraft;
- noisy matching of substrings in protein sequences.

Sequential data can be considered to be the one-dimensional analog to two-dimensional image data. *Time series* data are perhaps the most well-known example, where a sequence of observations is measured over time, such that each observation is indexed by a time variable t. These measurements are often made at fixed time intervals, so that without loss of generality t can be treated as an integer taking values from 1 to T. The measurements at each time t can be multivariate (rather than just a single measurement), such as (for example) the daily closing stock prices of a *set* of individual stocks. Time series data are measured across a wide variety of applications, in areas as diverse as economics, biomedicine, ecology, atmospheric and ocean science, control engineering, and signal processing.
The notion of *sequential data* is more general than time series data in the sense that the sequence need not necessarily be a function of time. For example, in computational biology, proteins are indexed by sequential *position* in a protein sequence. (Text could, of course, be considered as just another form of sequential data; however, it is usually treated as a separate data type in its own right.)

As with image and text data, it is now commonplace to store large archives of sequential data sets. For example, several thousand sensors of each NASA Space Shuttle mission

are archived once per second during the duration of each mission. With each mission lasting several days, this is an enormous repository of data (on the order of 10 Gbytes per mission, with order of 100 missions flown to date).
Retrieval in this context can be stated as follows: find the subsequence that best matches a given query sequence *Q*. For example, for the Shuttle archive, an engineer might observe a potentially anomalous sensor behavior (expressed as a short query sequence *Q*) in real time and wish to determine whether similar behavior had been observed in past missions.

14.6.1 Global Models for Time Series Data

Traditional time series modeling techniques (such as statistical methods) are largely based on *global linear* models, as discussed in chapter 6. An example is the Box-Jenkins autoregressive family of models where the current value $y(t)$ is modeled as a weighted linear combination of past values $y(t - k)$ plus additive noise:

(14.7)
$$y(t) = \sum_{i=1}^{k} \alpha_i y(t-i) + e(t)$$

where the a_i are the weighting coefficients and $e(t)$ is the noise at time t (typically presumed to be Gaussian with zero mean). The term *autoregression* is derived from the notion of using a regression model on past values of the same variable. Techniques that are very similar in spirit to the linear regression techniques of chapter 11 can be used to estimate the parameters a_i from data. Determination of the model structure (or order, k) can be performed by the usual techniques of penalized likelihood or cross-validation.
This type of model is closely linked to the spectral representation of y, in the sense that specifying the as also specifies the frequency characteristics for a stationary time series process y. Thus it is clear that it only makes sense to consider the use of the autoregressive model for time series that can be well characterized by stationary spectral representations, i.e., a linear system whose frequency characteristics do not change with time.
A general contribution of the Box-Jenkins methodology has been to show that if a time series has an identifiable systematic nonstationary component (such as a trend), the nonstationary component can often be removed to reduce the time series to a stationary form. For example, economic indices such as annual gross domestic product or the Dow Jones Index contain an inherent upward trend (on average) which is typically removed before modeling. Even if the nonstationarity is not particularly simple, another useful approach is to assume that the signal is *locally stationary* in time. For example, this is how speech recognition systems work, by modeling the sequence of phoneme sounds produced by the human vocal tract and mouth as coming from a set of different linear systems. The model is then defined as a mixture of these systems, and the data are assumed to come from some form of switching process (typically Markov) between these different component linear systems.
Nonlinear global models generalize equation 14.7 to allow (for example) a nonlinear dependence of $y(t)$ on the past:

(14.8)
$$y(t) = g\left(\sum_{i=1}^{k} \alpha_i y(t-i)\right) + e(t)$$

where $g(.)$ is a nonlinearity.
There are a large number of extensions of both the linear and nonlinear models of this general form. One key aspect they have in common is that, given an initial condition $y(0)$ and the parameters of the model, the statistics of the process (the distribution of y as a function of time) are completely determined, i.e., these models provide global aggregate descriptions of the expected behavior of the time series.
In terms of data mining, if we assume that such a global model is an adequate description of the underlying time series, we can use the model parameters (e.g., the weights above) as the basis for representing the data, rather than the original time series itself. For example, given a set of different time series (such as daily returns over time of various stocks), we could fit a different global model to each time series (i.e., estimate p parameters of a model) and then perform similarity calculations among the time series in p-dimensional parameter space. A problem arises here if the different time series are not all adequately modeled by the same model structure. One solution to this is to assume a

nested model structure (i.e., where the model structures form a nested sequence of increasing complexity) and to simply fit the maximum-order model to all the time series.

By representing each time series as a vector of parameters, we have in essence performed the same trick as that used for representing documents and images in earlier sections of this chapter. Thus, in principle, we can define similarity measures in the vector space of parameters, define queries in this space for retrieval by content, and so forth.

One interesting application of data mining in this context is a technique known as *keyword spotting* in speech recognition. Consider, for example, a national security organization that monitors and records international telephone conversations (ignore the moral and legal ramifications!). From a security viewpoint the goal is to detect suspicious behavior. Naturally it is impossible for a human to monitor and listen to all of the hundreds of thousands of telephone conversations recorded daily. One automated approach to the problem is to build statistical models of specific keywords of interest. For example, we could construct (from training data) a different Markov linear-switching model (as mentioned earlier) for each specific word of interest. The incoming voice streams are run through each model in parallel, and if the likelihood of the observed audio data exceeds a certain threshold for any of the models, then that word is considered to be detected, and the voice stream can be tagged with the identified word and the location in time of that word. Naturally there are numerous practical engineering considerations for such a system, but the basic concept is to use a set of trained models to adaptively monitor a real-time stream of data to detect patterns of interest.

14.6.2 Structure and Shape in Time Series

Consider a real valued subsequence of time series values, $Q = [q(t), ..., q(t + m)]$, which we will call the query pattern, and a much larger archived time series $X = [x(1), ..., x(T)]$. The goal is find a subsequence within X that is most similar to Q. In reality X could be composed of many individual time series, but for simplicity imagine that they have all been concatenated into a single long series. In addition, for simplicity assume that both X and Q have been measured with the same time-sampling interval, i.e., the time units for one increment of t are the same for each. For example, Q could be a snapshot from an EEG monitor of a patient being monitored in real time, and X could be an archive of EEGs for patients who have already been diagnosed.

Clearly there is considerable latitude in how *similarity* is defined in this context. Note that the general approach of the last section provides only a global statistical characterization of a time series. In particular, there is no notion of local *shape* such as a peak in time. Global statistical models typically average out such local structural properties; i.e., they are not explicitly retained in the representation. However, many time series are more naturally described by their structural features. A good example is the S-T waveform in cardiac monitoring, which has a distinctive visual signature.

One approach is to perform a sequential scan of the query Q across the full length of the archival data X, moving the query Q along X one time point at a time, and calculating a distance metric (such as Euclidean distance) at each point. This is typically not only prohibitively expensive ($O(mT)$ for a brute-force approach) but also focuses on low-level sampled data points of the the signal rather than high-level structural properties such as peaks, plateaus, trends, and valleys. Direct Euclidean-distance matching is also very susceptible to slight deformations of the query Q or data X, e.g., a slight "stretching" along the time axis of the "ideal" query Q can result in a large increase in distance even though from the viewpoint of the human visual observer the query Q and the data X may appear to match well.

A popular approach in this context is to locally estimate shape-based features of both the query Q and the archived signal X, and to perform matching at the higher structural level. This can achieve a significant computational advantage in the matching process, since the abstraction is in essence a form of data compression, i.e., many irrelevant details of the signal can be ignored. More important, it can extract structural information in a form that is suitable for human interpretation. One example of this technique is to approximate a signal by piecewise linear (or polynomial) segments. The segmented series can now be represented as a list of locally parametrized curves, and structural features (such as peaks and valleys) can be directly calculated from the parametrized description. A

probabilistic model can then be used to parametrize the expected shape and variability in terms of these features, allowing for a flexible family of deformable template models. Matching a query *Q* to a data archive *X* can be formulated as a search problem in terms of finding local regions in *X* that maximize the likelihood of the local data within that region given the probabilistic model for *Q*. This type of representation is particularly useful for the types of signals that are not easily handled by the global statistical models, i.e., nonstationary signals containing transients, step functions, trends, and various other shapelike patterns.

For discrete-valued sequences we can also look for subpatterns within a longer sequence, e.g., the occurrence of motifs in biological sequence data. A wide variety of techniques are used for these sorts of problems, ranging from relatively nonparametric edit-distance methods for matching two strings, to probabilistic model-based approaches using generative Markov (or hidden Markov) models.

14.7 Summary

Retrieval by content is an important problem for the interactive exploration of large databases. In particular, for data types such as images, text, and sequences, algorithms for retrieval by content have great potential utility across a variety of applications. However, in their full generality, such algorithms require the solution of several fundamental and long-standing problems in artificial intelligence and pattern recognition, such as the general NLP problem (for text) or the general image understanding problem (for images). In short, we are a long way from developing general-purpose methods for automatically extracting semantic content from data such as text or images in a manner that is competitive with the human brain.

Nonetheless, given that in many applications it is impossible to analyze the data manually because of its sheer volume, researchers have developed a variety of techniques for retrieval by content, largely relying on what we might term "low-level semantic content." Thus, for example, we retrieve images based on low-level features such as color or texture, and retrieve text based on word co-occurrences.

Across different data types, we can see that a common strategy for retrieval is often used, roughly following these steps:

1. Determine a robust set of features by which to describe the objects of interest.
2. Convert the original objects (text, images, sequences) into a fixed-length vector representation using these features.
3. Perform matching, distance calculations, principal component analysis, and so forth, in this vector-space, taking advantage of the wealth of theory that exists for multivariate data analysis.

We might term such systems as *first-generation retrieval by content systems*. Certainly they can be very useful, as is evidenced by Web search engines and the QBIC system, for example. However, it is clear that retrieval by content is far from being a solved problem, and there is considerable room for further advances.

14.8 Further Reading

The collected volume by Sparck Jones and Willett (1997) contains many of the classic papers in information (text) retrieval, with some very insightful and broad-ranging editorial discussion of central themes in retrieval problems and research. Van Rijsbergen (1979), Salton and McGill (1983), and Frakes and Baeza-Yates (1992) provide a more introductory coverage of the field. Salton (1971) contains many of the seminal early ideas on the vector-space representation and Raghavan and Wong (1986) provide a later perspective. Salton and Buckley (1988) is a brief but informative account of different term-weighting methods, with particular emphasis on the TF-IDF method. The TREC conferences are documented in Harman (1993–1999), and Harman (1995) provides a useful overview of the TREC experiments. A more recent treatment of general issues in evaluation in the context of text retrieval is in the special issue of the *Journal of the American Society for Information Science* (1996). Witten, Moffat, and Bell (1999) provide an excellent account of many of the practical data engineering issues involved in storing and accessing large text document archives.