Principles of Data Mining

David Hand
Imperial College

Heikki Mannila
Microsoft Research

Padhraic Smyth
University of California at Irvine

MIT Press, to appear
Draft — November 24, 1999
Do not distribute

Chapter 14

Retrieval by Content

Contents

14.1 Introduction

In a database context, the traditional notion of a query is well-defined, namely an operation which returns a set of records (or entities) which exactly match a set of required specifications. An example of such a query in a personnel database would be `[level = MANAGER] AND [age < 30]` which (presumably) would return a list of young employees with significant responsibility.

However, there are many instances, particularly in data analysis, where we are interested in more general queries. Consider a medical context. We have a patient for whom we have demographic information (such as age, sex, and so forth), results from blood tests and other routine physical tests, as well as biomedical time series and X-ray images. To assist in the diagnosis of this patient, a physician would like to know if the hospital's database contains any similar patients, and if so, what the diagnoses, treatment, and outcome was for each. The difficult part of this problem is determining *similarity* among patients based on different data-types (here, multivariate, time-series and image data). Note that the notion of an exact match is not directly relevant for this problem, since it is highly unlikely that any other patient will match this particular patient exactly in terms of measurements.

In this chapter we will discuss problems of this nature, specifically the technical problems which must be addressed to allow queries to our data set of the following general form:

> Find the k objects in the database which are most similar to either a specific query or a specific object.

Example of such queries might be

- Searching historical records of the Dow Jones index for past occurrences of a particular time-series pattern.
- Searching a database of satellite images of the Earth for any images which contain evidence of recent volcano eruptions in Central America.
- Searching the internet for online documents which provide reviews of restaurants in Helsinki.

This form of retrieval can be viewed as *interactive* data mining in the sense that the user is directly involved in exploring a data set by specifying queries and interpreting the results of the matching process. This is in contrast to many of the predictive and descriptive forms of data mining discussed in earlier chapters, where the role of human judgement is often not as prominent.

If the data sets are annotated by content (for example, if the image database above had been manually reviewed and indexed based on visual content) then the retrieval problem reduces to a standard (but potentially challenging) problem in database indexing as discussed in Chapter 4. We are interested here, however, in the more common case in practice where the database is not pre-indexed. Instead, we have an example of what we are trying to find, namely, *the query pattern* Q. From the query pattern, we must infer which other objects in the data set are most similar to it. This approach to retrieval is known as *retrieval by content*. The best known applications of such an approach are in text retrieval. Here the query pattern Q is usually quite short (a list of query words) and is matched with large sets of documents.

We will discuss general issues in retrieval by content not only for text, but also for image, time-series, and multivariate data. The general problem can be thought of as having three fundamental components, namely,

- how to define a similarity measure between objects,

- how to implement a computationally efficient search algorithm (given a similarity measure), and
- how to incorporate user feedback and interaction in the retrieval process.

We will focus primarily on the first and third problems. The second problem typically reduces to an indexing problem (i.e., find the closest record in a database to a specific query) which was discussed in Chapter ???.

Retrieval by content relies heavily on the notion of *similarity*. Throughout the discussion we will use both the terms *similarity* and *distance*. From a retrieval point of view it is not so important whether we use one or other, since we can either maximize similarity or minimize distance. Thus, we will implicitly assume that, loosely speaking, these two terms are inverses of each other, and either can be used in practice.

We will see that across different applications (text, images, and so forth) it is common to reduce the measurements to a standard fixed-length vector format, and to then define distance measures using standard geometric notions of distance between vectors. Recall that in Chapter ???? we discussed several standard distance measures such as 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 the subtle issue of how one can objectively evaluate the performance of a specific retrieval algorithm. The evaluation is significantly complicated by the fact that the ultimate judgement of performance comes from the subjective opinion of the user issuing the query, who determines whether or not the retrieved data is relevant or not.

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. Concepts such as latent-semantic indexing relevance feedback and collaborative filtering, which provide interesting mechanisms for the capture of human judgement in an interactive manner, are also discussed in this context. In Section 14.4 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 14.5 reviews basic concepts in sequence retrieval. As the 1-dimensional analog of image retrieval, very similar representational and invariance issues arise for sequences as for image data. Section 14.6 and Section 14.7 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 its accuracy (or more generally its loss) on unseen test data. This makes comparisons of different models and algorithms relatively 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 how well a retrieval system is performing is determined by how useful the retrieved information is 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-centred, interactive process, which makes performance evaluation difficult. This is an important point to keep in mind below.

Although it may be very difficult to directly measure how useful a particular retrieval system is to an average user, 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, there is the assumption 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 judgements can be a formidable obstacle.

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 non-relevant.

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 is one which 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

	Truth: Relevant	Truth: Not-Relevant
Algorithm: Relevant	n_a	n_b
Algorithm: Not-Relevant	n_c	n_d

Table 14.1: A schematic of the 4 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. A perfect retrieval algorithm would produce a diagonal matrix with $n_b = n_c = 0$. This form of reporting classification results is sometimes referred to as a *confusion matrix*.

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 which 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 terms of which 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 $N = n_a + n_b + n_c + n_d$ is the total number of labeled objects, $n_a + n_b = K_T$ is the number of objects returned by the algorithm, and $n_a + n_c$ is the total number of relevant objects. *Precision* is defined as the fraction of retrieved objects which are relevant, i.e., $n_a/(n_a + n_b)$. *Recall* is defined as the proportion of relevant objects which are retrieved relative to the total number of relevant objects in the data set, i.e., $n_a/(n_a + n_c)$. There is a natural trade-off here. As the number of returned objects K_T is increased (i.e., as we lower 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 only return relevant objects). If we run the retrieval algorithm for different values of 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, the average recall-precision performance over a set of queries is estimated (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.

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.

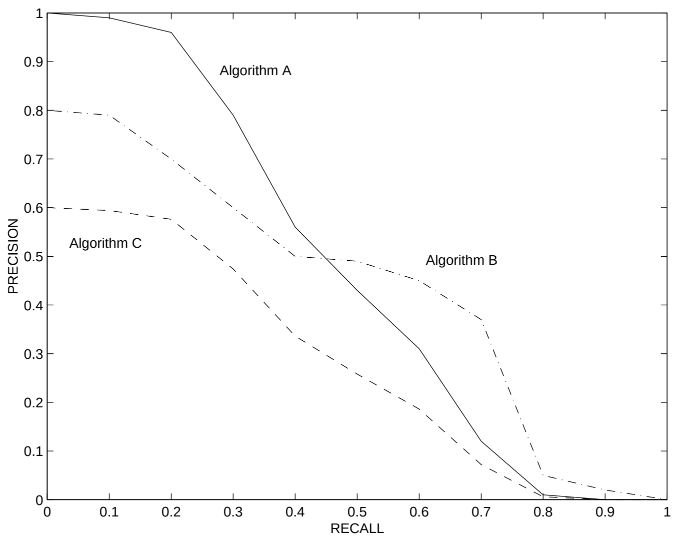

Figure 14.1: A simple (synthetic) example of precision-recall curves for three hypothetical query algorithms. Algorithm A has the highest precision for low recall values, while Algorithm B has the highest precision for high recall values. Algorithm C is universally worse than A or B, but one cannot make a clear distinction between A or B unless (for example) one 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.

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 one can use to summarize precision-recall performance with a single number, e.g., precision at some fixed value of recall, or precision at the point where recall and precision are equal.

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 US 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 judgements 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 judgements 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 relevant documents which are potentially available. Sampling techniques can in principle be used, but combined with the fact that subjective human judgement is involved in determining relevance in the first-place, precision-recall experiments on a large-scale can be extremely non trivial 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 more generally it is a used as a name for any structured segment of text such as chapters, sections, paragraphs, or even email messages, Web pages, computer source code, and so forth. A *term* is a word or a phrase within a document, e.g., the word "data" or the phrase "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 which 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, one of the primary goals of text retrieval is to find a general *representation* for documents which supports both

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

A user who is using a text retrieval system (such as a search engine on the Web) wants to retrieve documents which 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 which 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 rely on relatively simple term matching and counting techniques, where the content of a document is implicitly and approximately captured (at least in theory) by various word occurrence counts. Assume that a set of terms t_j for retrieval, $1 \leq j \leq T$, 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 $D_i, 1 \leq i \leq N$, is then represented as a *term vector*:

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

where d_{ij} represents some information about the occurrence of the jth term in the ith 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 or not 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.

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

- t1 = database

	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

Table 14.2: A toy document-term matrix for 10 documents and 6 terms. Each ijth entry contains the number of times that term j appears in document i.

- 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 terms j contained in document i. We can clearly see that the first 5 documents $d1$ to $d5$ contain mainly database terms (combinations of *query, SQL, index*) while the last 5 documents $d6$ to $d10$ contain mainly regression terms (combinations of *regression, likelihood, linear*). We will return to this example later in this chapter.

Given a particular vector-space representation, it is relatively 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 ???? 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

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

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.

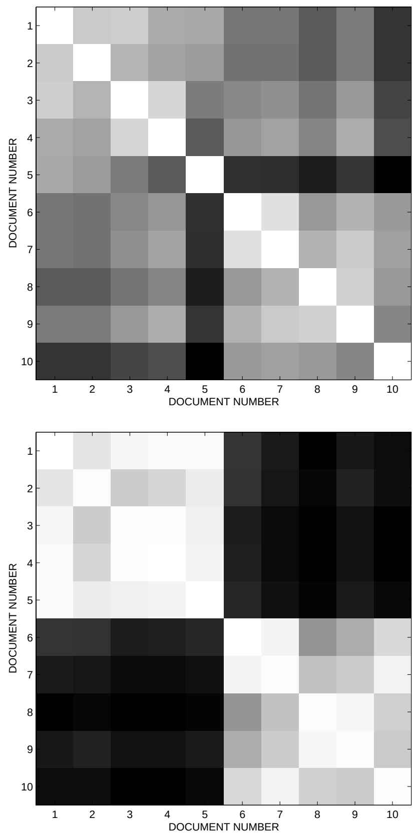

Figure 14.2: Pairwise document distances for the toy document-term matrix in the text, for Euclidean distance (top) and cosine distance (bottom). The pixel intensities for the Euclidean distance have been plotted so that white corresponds to zero (e.g., on the diagonals) and black is the maximum distance 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).

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 relatively 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).

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 on the order of 0.03% of cells being non-zero 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 non-trivial, 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 for a particular application.

14.3.2 Matching Queries and Documents

Queries can also be expressed using the same term-based representation as used for documents. In effect, the query is itself represented as a document, albeit one that typically contains very few terms.

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 which 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 is 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 both the query as well as the weights for the terms in query.

Let the $\underline{Q} = (q_1, \ldots, q_T)$ be a vector of query weights. For simplicity and efficiency reasons, it is common in practice for the query weights to be simply either 1 (the term is in the query) or 0 (the term is not in the query). For example, consider three queries, each consisting of the single terms *database, SQL, regression.* For our toy example these 3 queries would be represented as three vectors: $(1,0,0,0,0,0), (0,1,0,0,0,0), (0,0,0,1,0,0)$. Using the cosine distance to match these 3 queries to our toy example data set of Table 14.2 results in documents $d2, d3$, and $d9$ 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, $1 \leq k \leq T$ 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 larger documents are more likely to include a given query term somewhere in the document.

One particular weighting scheme has proven extremely useful in practice, known as the TF-IDF weighting. "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 which occur relatively frequently in a given document. However, if this same term occurs frequently in many documents in the document set, then the TF method may have little discriminative power, i.e., it will increase recall but may have poor precision. The toy document-term matrix of Table ??? is expresed in TF form.

The *inverse-document-frequency* (IDF) weight is defined as $\log(N/n_j)$, i.e., the log of the inverse of the fraction of documents in the whole set which 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 which occur in relatively few documents, i.e., it has discriminative power. The use of the log IDF rather than IDF directly is motivated by the desire to make the weight relatively insensitive to the exact number of documents N.

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

Table 14.3: 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 $d5$ is closest; for the TF-IDF matrix, $d2$ is closest.

The TF-IDF weight is simply the product of TF and IDF for a particular term in a particular document. Like 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 to comparable alternative weighting schemes (there are enhancements to the basic TF-IDF method, but it is still considered the default baseline method for comparative purposes).

The toy example 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 a TF-IDF document-term matrix of the following form:

```
2.5287   14.5561    4.5974         0         0    2.0794
3.3715    6.9315    2.5541         0    1.0700         0
1.2643   11.0904    2.5541         0         0         0
0.6322    4.8520    1.0217         0         0         0
4.5305   21.4876   10.2165         0    1.0700         0
0.6322         0         0   11.7835    1.4267   15.9424
0.2107         0         0   12.4766    2.4967   11.0904
     0         0    0.5108   22.1807    4.2801         0
0.3161         0         0   15.2492    1.4267    1.3863
0.1054         0         0   23.5670    9.6302   17.3287
```

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

- represent queries as term vectors with 1's for terms occurring in the query and 0's everywhere else,

- represent term-vectors for documents using TF-IDF weights for the vector components

- use the cosine distance measure to rank the documents in terms of distance to the query

Table 14.3 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 which include at least one relevant term.

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 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) which 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 explained in Chapter ????, 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 relatively 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 which 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 or not 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 only contain 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 an SVD decomposition for the toy document-term matrix M in Table 14.2 such that $\mathbf{M} = \mathbf{USV^T}$. Here $\mathbf{U}$ is a 10×6 matrix of weights for each document, $\mathbf{S}$ is a 6×6 diagonal matrix of eigenvalues for each principal component direction, and the columns of the 6×6 matrix $\mathbf{V}^T$ provide a new orthogonal basis for the data, often referred to here as the principal component directions.

The diagonal elements of the S matrix for M are $\lambda_1, \ldots, \lambda_6 = \{77.4, 69.5, 22.9, 13.5, 12.1, 4.8\}$. 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 only to retain 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* which are in effect linear combinations of the original six terms.

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

$$\mathbf{v}_1 \quad = (0.74, 0.49.0.27, 0.28, 0.18, 0.19) \tag{14.3}$$

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

These are the two directions (a plane) in the original 6-dimensional term space where the data is most "spread out" (has the most variance). The first direction is emphasizes the first two terms (*query, SQL*) more than the others: this is in effect the direction which characterizes documents relating to databases. The second direction emphasizes the last three terms *regression, likelihood, linear*, and can be considered the direction which 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 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 and that symbols D1 and D2 are discussed later. The distances of the points from the origin reflect the magnitude

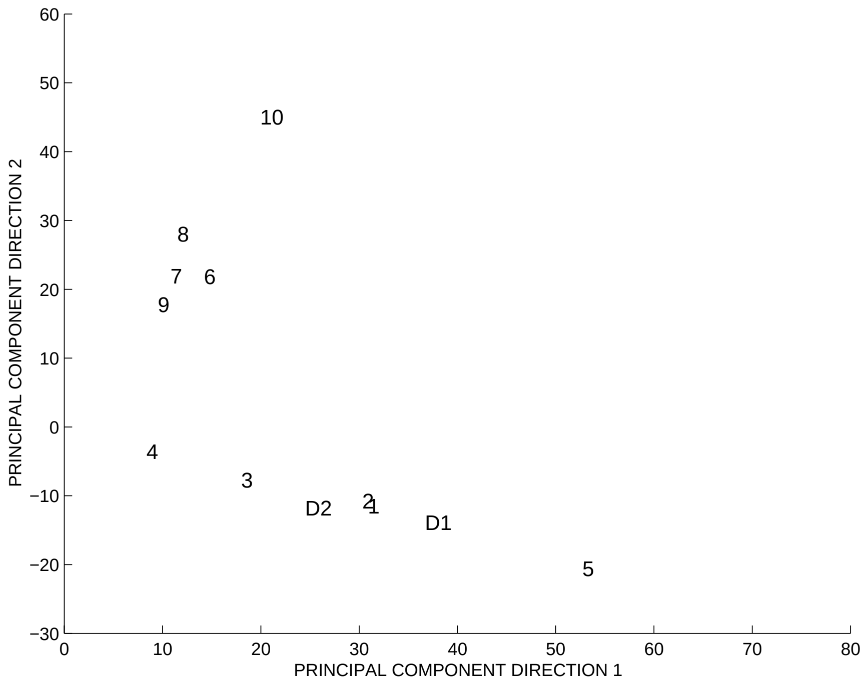

Figure 14.3: Projected locations of the 10 documents (Table 14.2) in the two dimensional plane spanned by the first two principal components of the document-term matrix M.

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 will is clearly a very useful indicator of similarity, since regression and database documents are each clustered around two different angles in the plane.

An advantage of principal components in this context can be seen by considering a new document D1 which contains (say) 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 they are each 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 which do not contain the term *SQL* at all, but which are related in content.

There are many variations on this basic idea. The application of principal components for text retrieval is often referred to as latent semantic indexing (LSI); however, the underlying principles are the same as discussed above. The use of this technique has been shown to consistently increase retrieval performance in systematic tests, based on its ability to match documents and queries with no terms in common.

14.3.4 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 "non-relevant" 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

$$Q_{optimal} = \frac{1}{|R|} \sum_{\underline{d} \in R} \underline{d} - \frac{1}{|NR|} \sum_{\underline{d} \in NR} \underline{d}. \tag{14.5}$$

where $\underline{d}$ represents a document whose labeling (by the user) is 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 be suboptimal relative to $Q_{optimal}$. The algorithm uses this initial query to return a small set of documents which are then labeled by the user as being relevant R' or not NR'. Rocchio's algorithm then refines the query in the following manner:

$$Q_{new} = \alpha Q_{current} + \frac{\beta}{|R'|} \sum_{\underline{d} \in R'} \underline{d} - \frac{\gamma}{|NR'|} \sum_{\underline{d} \in NR'} \underline{d}. \tag{14.6}$$

Thus, the query is modified by (in effect) moving the current query towards 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 α, β and γ are positive constants (chosen heuristically) which 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. Clearly there are many details which must be specified to implement this method in practice, such as how many documents should be presented to the user, the relative number of relevant and irrelevant documents to be used, how the irrelevant documents are chosen, and so forth, leading to a large number of variations on this basic theme.

14.3.5 Collaborative Filtering

Collaborative filtering extends the notion of modeling the preferences of a single user to the situation where a system has information about multiple users and their preferences in terms of a large database of objects. The basic idea is quite simple. For example, imagine you are interested in a certain musical group and buy a CD for this group at an online Web site. There may be several hundred other people who have also 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. The well-known online bookseller `Amazon.com` uses this system to automatically recommend books to online customers. Clearly one can generalize the basic idea in various directions. For example, if users are willing to provide more detailed information about their specific interests (in the form of user profiles) then one 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 in a sense is an attempt to capture the expert judgement 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 user-profiles who 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 which is known about each user and the size of the user database. The technique tends to work best with relatively large numbers of users. Acquiring a large number of user profiles can be relatively difficult in practice, since users are naturally reluctant to take the time to provide detailed personal information. Capturing a user's preferences by their actions (e.g., by what they buy, or by what Web pages they visit) is a less intrusive approach which implicitly tries to estimate user preferences, and is commonly employed in internet-based recommendation systems (for example).

14.4 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 who 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 non-trivial problem!

Thus, there is considerable motivation to develop efficient and accurate query-by-content algorithms for image databases, i.e., to develop interactive systems which 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.

14.4.1 Image Understanding

Querying image data comes with an important caveat. Finding images which 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 has 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). Very 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.

14.4.2 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 are may be invariant to only relatively small changes in lighting, shading, viewpoint, etc.

Typically the features on 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 $N \times d$ data matrix, where each image is now represented as a d-dimensional vector in feature space.

Spatial information can be introduced in a coarse manner by computing features in localized sub-regions of an image. For example, one could compute color information in each 32×32 subregion of a 1024×1024 pixel image. This allows relatively coarse spatial constraints to be specified in image queries, such as "find images which 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, one can also extract primitive properties which 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.

For example, the QBIC system (REFER TO SIDEBAR HERE) uses a variety of features and a variety of associated distance measures:

- A 3-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 3-dimensional texture vector consisting of features which 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.

14.4.3 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 either be expressed in two basic forms. In "query by example" one provides a sample image of what one is looking for, or sketches 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 pre-computed database of feature vectors. Alternatively, the query can be expressed directly in terms of the feature representation itself, e.g., "find images which 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) then only that subset of features is used in the distance calculations.

Clearly, depending on the application, one 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 which take advantage of spatial relations (such as "find images with object 1 above object 2") or sequential relations (such as "find video sequences with landscape scenes followed by ocean scenes").

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.

14.4.4 Image Invariants

For retrieval by content with images it is important to keep in mind that (with current techniques at least) we can realistically only work with a restricted notion of semantic content, based on relatively simple "low-level" measurements such as color, texture, and simple geometric properties of objects. It is important to keep in mind that there are many common distortions in visual data such as translations, rotations, non-linear 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 described for content retrieval above typically are *not* invariant under such distortions, i.e., changes in scale, illumination, 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 non-linear transformations of non-rigid objects.

14.4.5 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) we have used above. 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.

A Real World Application of Retrieval by Content: The QBIC System:

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 1990's. The system is based on the general ideas described in Section 14.4, 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.

One application of the system is at the Fine Arts Museum of San Francisco. The museum's Web page (`http://www.thinker.org/imagebase/index-2.html`*) uses QBIC to allow online interactive searching of more than 3,000 Japanese prints. This is a typical example of the interactive user-centered nature of retrieval-by-content, namely many different users with different potential interests using a standard query engine to explore a large image database in real-time.*

14.5 Time Series and Discrete Sequence Retrieval

The problem of efficiently and accurately locating patterns of interest in time series and sequence data sets is an important and non-trivial 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, e.g.,

- 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, and
- 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 are measured over time, such that each observation is indexed by a time variable t. Typically, these measurements occur 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, atmospheric and ocean science, control theory, signal processing, and ecology.

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 which 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 or not similar behavior had been observed in past missions.

14.5.1 Global Models for Time Series Data

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

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

where the α_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 which are very similar in spirit to the linear regression techniques of Chapter 12 can be used to estimate the parameters α_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 α's 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 which 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 non-stationary component (such as a trend), the non-stationary 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 non-stationarity 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.

Non-linear global models generalize Equation 14.7 to allow (for example) a non-linear dependence of $y(t)$ on the past:

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

where $g(.)$ is a nonlinearity.

There are a large number of extensions of both the linear and non-linear models of this general form. One key aspect they have in common is that the 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) is completely determined, i.e., these models provide global aggregate descriptions of the expected behavior of the time series.

In terms of data mining, if one assumes that such a global model is an adequate description of the underlying time-series, one 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 different stocks), one 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 workaround here 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 was 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 which monitors and records international telephone conversations (lets 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 hundred's 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, one could construct (from training data) a different 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 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.

14.5.2 Structure and Shape in Time Series

Consider a real valued subsequence of time-series values, $Q = [q(t), \ldots, q(t+m)]$, which we will call the query pattern and a much larger archived time series $X = [x(1), \ldots, x(T)]$. The goal is find a subsequence within X which 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 only provides 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. (IDEALLY INCLUDE A FIGURE HERE).

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.

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 typically 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 importantly, it can extract structural information in a form which 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. This type of representation is particularly useful for the types of signals which are not easily handled by the global statistical models, i.e., non-stationary signals containing transients, step functions, trends, and various other shape-like patterns.

14.6 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 which is competitive with the human brain.

Nonetheless, given that in many applications it is impossible to manually analyze the data due to its sheer volume, researchers nonetheless 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 relatively low-level features such as color or texture, and retrieve text based on word co-occurrences.

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

1. Determine a relatively 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 which exists for multivariate data analysis.

We might term such systems as “first-generation retrieval by content systems.” Certainly they can be very useful, in evidenced by 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.

14.7 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 et al (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).

Deerwester et al (1990) is one of the first papers to clearly demonstrate the utility of LSI in information retrieval. In a more recent paper, Landauer and Dumais (1997) provide a thought-provoking discussion of the general implications of the LSI approach for cognitive models of language and knowledge acquisition.

Rocchio (1971) introduced the original algorithm for relevance feedback. Salton and Buckley (1990) provide experimental evidence on the effectiveness of relevance feedback in improving recall-precision performance and Buckley and Salton (1995) discuss optimality aspects of Rocchio's algorithm. Resnick et al (1994) and Shardanand and Maes (1995) describe the initial work on collaborative filtering. Breese, Heckerman, and Cadie (1998) contains a more recent empirical evaluation of large-scale collaborative filtering algorithms.

The QBIC system is described in some detail in Faloutsos et al (1994) and Flickner et al (1995). The first paper contains a reasonably detailed discussion of the features, distance measures, and indexing schemes used, while the second paper focuses somewhat more on user interface issues. Other query by content systems for image and video retrieval are described in Kato, Kurita, and Shimogaki (1991), Smoliar and Zhang (1994), Pentland, Picard, and Sclaroff (1996), Smith and Chang (1997), Viola and Bonet (1998). Rui et al (1997) discuss the use of relevance feedback in an image-retrieval context. The edited collection by Maybury (1997) provides a useful overview of recent work in the general area of retrieval of multimedia objects such as images and video.

Box and Jenkins (1976) is a comprehensive and classic text on the fundamentals of linear global models for time series data. Chatfield (1989) provides a somewhat broader perspective and is particularly noteworthy as a gentle introduction to time series concepts for readers unfamiliar with the field in general. MacDonald and Zucchini (1997) provide a comprehensive description of statistical approaches to modeling discrete-valued time-series, and Durbin et al (1998) illustrate the application of sequence modeling and pattern recognition techniques to protein sequences and related problems in computational biology.

There are a large number of different techniques for efficient subsequence matching. The work of Faloutsos, Ranganathan, and Manolopolous (1994) is fairly typical. Sequences are decomposed into windows, features are extracted from each window (locally estimated spectral coefficients in this case), and efficient matching is then performed using an R^*-tree structure in feature space. Agrawal et al. (1995) proposed an alternative approach which can handle amplitude scaling, offset translation, and "don't care" regions in the data, where distance is determined from the envelopes of the original sequences. Berndt and Clifford (1994) use dynamic time-warping approach to allow for "elasticity" in the temporal axis when matching a query Q to a reference sequence R. Another popular approach is to abstract the notion of shape. Relational trees can be used to capture the hierarchy of peaks (or valleys) in a sequence and tree matching algorithms can then be used to compare two time series (Shaw and DeFigueiredo, 1990; Wang, et al., 1994). Keogh and Smyth (1997) illustrate the use of elastic templates in for matching time series shapes in a probabilistic framework, and Keogh and Pazzani (1999) extend this work to incorporate relevance feedback in retrieval of time-series patterns.

References

Agrawal, R., Lin, K. I., Sawhney , H. S and Shim, K. (1995).Fast similarity search in the presence of noise, scaling, and translation in time-series databases. *Proceedings of VLDB-95* 490–501.

Berndt, D. J., Clifford, J. (1996). Finding patterns in time-series, a dynamic programming approach. In *Advances in Knowledge Discovery and Data Mining*, Fayyad, U. M, Piatetsky-Shapiro, G., Smyth, P., and Uthurasamy, R. (eds), Menlo Park, CA: AAAI/MIT Press.

Blum, T., Keislaer, D., Wheaton, J., and Wold, E. (1997). Audio databases with content-based retrieval. In Maybury, M. T. (ed.), *Intelligent Multimedia Information Retrieval*, Menlo Park, CA: AAAI Press, pp.113–135.

Box, G. E. P., and Jenkins, F. M. (1976). *Time Series Analysis: Forecasting and Control*, Oakland, CA: Holden Day.

Breese J.S., Heckerman D. and Kadie C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings 14th Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA: Morgan Kaufman.

Buckley, C. and Salton, G. (1995). Optimization of relevance feedback weights. *Proceedings of SIGIR 1995.*

Chatfield, C. (1989). *The Analysis of Time Series*, 4td ed., London: Chapman and Hall.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41, 391–407.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge, UK: Cambridge University Press.

Faloutsos, C., Ranganathan, M., Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *SIGMOD–Proceedings of Annual Conference*, Minneapolis..

Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., and Equitz, W. (1994). Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3, 231–262.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., and Yanker, P. (1995). Query by image and video content. *IEEE Computer*, September 1995, 23–31.

Frakes, W. B. et al. (1992). *Information Retrieval: Data Structure and Algorithms*, New Jersey: Prentice Hall.

Harman, D. K. (1993). The First Text REtrieval Conference (TREC-1), NIST SP 500-207, National Institute of Standards and Technology, Gaithersburg, MD (annual series, 1993–1999).

Harman, D. K., (1995). In R. Kuhlen and M. Rittberger (eds.), *Hypertext — Information Retrieval — Multimedia: Synergieeffekte Elektronischer Informationssysteme, Proceedings of HIM' 95*, pp.9–28, Konstanz: Germany, Universitaetsforlag Konstanz.

Journal of the American Society for Information Science (1996) Special Issue on Evaluation, 47, 1–105.

Kato, T., Kurita, T., Shimogaki, H. (1991). Intelligent visual interaction with image database systems — towards the multimedia personal interface. *Information Processing* (Japan), 14, 134–143.

Keogh, E. and Smyth, P. (1997). A probabilistic approach to fast pattern matching in time-series databases. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA: AAAI Press, 24–30.

Keogh, E. and Pazzani, M. (1999). Relevance feedback retrieval of time series data. *Proceedings of the ACM Conference on Information Retrieval*, ACM Press.

Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato's problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2), 211–240.

MacDonald, I. L. and Zucchini, W. (1997). *Hidden Markov and other models for discrete-valued time-series*, London: Chapman and Hall.

Maybury, M. T. (ed.) (1997). *Intelligent Multimedia Information Retrieval*, Menlo Park, CA: AAAI Press.

Pentland, A., Picard, R. W., and Sclaroff, S. (1994). Photobook: tools for content-based manipulation of image databases. *International Journal of Computer Vision*, 18, pp.233–254.

Raghavan, V. V. and Wong, S. K. M. (1986). A critical analysis of the vector space model for information retrieval. *Journal of the American Society for Information Science*, 37(5), 100–124.

Resnick P, Iacovou N., Suchak M., Bergstrom, and Riedl J. (1994). GroupLens: An open architecture for collaborative filtering of netnews. *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, Chapel Hill, NC: ACM, 175-186.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In *The SMART Retrieval System: Experiments in Automatic Document Processing*, Salton, G. (ed.), Englewood Cliffs, NJ: Prentice Hall, 313–323.

Rui, Y., Huang, T. S., Mehrotra, S., and Ortega, M. (1997). A relevance feedback architecture in content-based multimedia information retrieval systems. *Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries.*

Salton, G. (ed.) (1971) *The SMART Retrieval System: Experiments in Automatic Document Processing*, Englewood Cliffs, NJ: Prentice Hall.

Salton, G. and Buckley, C. (1988) Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24, 513-523.

Salton, G. and Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society of Information Science*, 41(4), 288-297.

Salton, G. and McGill, M. (1983). *Introduction to Modern Information Retrieval*, New York: McGraw Hill.

Shardanand U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth." *Proceedings of CHI'95–Human Factors in Computing Systems*, 210-217.

Shaw, S. W., Defigueiredo, R. J. P. (1990). Structural processing of waveforms as trees. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 38(2), 328–338.

Smith, J. R. and Chang, S. (1997). Querying by color regions using VisualSEEk content-based visual query system. In Maybury, M. T. (ed.), *Intelligent Multimedia Information Retrieval*, Menlo Park, CA: AAAI Press, pp.23–41.

Smoliar, S., and Zhang, H. (1994). Content-based video indexing and retrieval. *IEEE Multimedia*, 1, pp.62–72.

Sparck Jones, K. and Willett, P. (1997) *Readings in Information Retrieval*, San Francisco, CA: Morgan Kaufmann.

Van Rijsbergen, C. J. (1979). *Information Retrieval*, London: Butterworth Press.

Viola and Bonet, NIPS 97 paper ???

Wang, J. T., Zhang, K., Jeong, K and Shasha. D. (1994). A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(2).