

## Matrices, Vector Spaces, and Information Retrieval\*

Michael W. Berry<sup>†</sup>  
Zlatko Drmač<sup>‡</sup>  
Elizabeth R. Jessup<sup>§</sup>

**Abstract.** The evolution of digital libraries and the Internet has dramatically transformed the processing, storage, and retrieval of information. Efforts to digitize text, images, video, and audio now consume a substantial portion of both academic and industrial activity. Even when there is no shortage of textual materials on a particular topic, procedures for indexing or extracting the knowledge or conceptual information contained in them can be lacking. Recently developed information retrieval technologies are based on the concept of a vector space. Data are modeled as a matrix, and a user's query of the database is represented as a vector. Relevant documents in the database are then identified via simple vector operations. Orthogonal factorizations of the matrix provide mechanisms for handling uncertainty in the database itself. The purpose of this paper is to show how such fundamental mathematical concepts from linear algebra can be used to manage and index large text collections.

**Key words.** information retrieval, linear algebra, QR factorization, singular value decomposition, vector spaces

**AMS subject classifications.** 15-01, 15A03, 15A18, 65F50, 68P20

PII: S0036144598347035

**1. Problems in Indexing Large Text Collections.** Traditional indexing mechanisms for scientific research papers are constructed from information such as their titles, author lists, abstracts, key word lists, and subject classifications. It is not necessary to read any of these items in order to understand a paper: they exist primarily to enable researchers to find the paper in a literature search. For example, the key words and subject classifications listed above enumerate what we consider to be the major mathematical topics covered in this paper. In particular, the subject classification 68P20 identifies this paper as one concerned with information retrieval (IR). Before the advent of modern computing systems, researchers seeking particular information could only search through the indexing information manually, perhaps

\*Received by the editors January 12, 1998; accepted for publication (in revised form) November 4, 1998; published electronically April 23, 1999.

<sup>†</sup><http://www.siam.org/journals/sirew/41-2/34703.html>

<sup>‡</sup>Department of Computer Science, University of Tennessee, Knoxville, TN, 37996-1301 (berry@cs.utk.edu). The work of this author was supported in part by National Science Foundation grant ACI-95-11394.

<sup>§</sup>Department of Computer Science, University of Colorado, Boulder, CO 80509-0330 (zlatko@cs.colorado.edu, jessup@cs.colorado.edu). The work of the second author was supported by National Science Foundation grants ACIR-83-57812 and ACIR-96-25912. The work of the third author was supported by National Science Foundation grant ACIR-95-57812 and Department of Energy grant DE-FC03-97ER25325.

in a card catalog. If an abstract or key word list were not provided, a professional indexer or cataloger could have written one.

These manual methods of indexing are succumbing to problems of both capacity and consistency. At the time of this writing, about 156,000 periodicals are published in print worldwide, with roughly 12,000 periodicals being added to that number each year [58]. There are nearly 1.4 million books in print in the United States alone, with approximately 60,000 new titles appearing there annually [15, 16]. The Library of Congress maintains a collection of more than 17 million books and receives new items at a rate of 7,000 per working day [39]. While these numbers imply daunting indexing problems, the scale is even greater in the digital domain. There are currently about 300 million Web pages on the Internet [19, 35], and a typical search engine updates or acquires pointers to as many as 10 million Web pages in a single day [53]. Because the pages are indexed at a much slower rate, the indexed collection of the largest search engine presently totals about 100 million documents [13, 53, 35].

Even when subsets of data can be managed manually, it is difficult to maintain consistency in human-generated indexes: the extraction of concepts and key words from documentation can depend on the experiences and opinions of the indexer. Decisions about important key words and concepts can be based on such attributes as age, cultural background, education, language, and even political bias. For instance, while we chose to include only higher-level concepts in this paper's key word list, a reader might think that the words *vector* and *matrix* should also have been selected. Our editor noted that the words *cosynergy* and *application* did not appear in the list even though they describe the main purpose of this paper. Experiments have shown that there is a 20% disparity on average in the terms chosen as appropriate to describe a given document by two different professional indexers [28].

These problems of scale and consistency have fueled the development of automated IR techniques. When implemented on high-performance computer systems, such methods can be applied to extremely large databases, and they can, without prejudice, model the concept-document association patterns that constitute the *semantic structure* of a document collection. Nonetheless, while automated systems are the answer to some concerns of information management, they have their own problems. Disparities between the vocabulary of the systems' authors and that of their users poses difficulties when information is processed without human intervention. Complexities of language itself present other quandaries. Words can have many meanings: a *bank* can be a section of computer memory, a financial institution, a steep slope, a collection of some sort, an airplane maneuver, or even a billiard shot. It can be hard to distinguish those meanings automatically. Similarly, authors of medical literature may write about *myocardial infarctions*, but the person who has had a minor *heart attack* may not realize that the two phrases are synonymous when using the public library's on-line catalog to search for information on treatments and prognosis. Formally, *polysemy* (words having multiple meanings) and *synonymy* (multiple words having the same meaning) are two major obstacles to retrieving relevant information from a database.

Polysemy and synonymy are two of the fundamental problems that any conceptual indexing scheme must overcome. Other issues such as the breadth and depth of concept extraction, zoning (indexing of parts of a document like its title, abstract, or first few paragraphs, as opposed to the entire document), and term or phrase weighting may also affect retrieval performance [33]. Indexing approaches (automated and otherwise) are generally judged in terms of their recall and precision ratings.

*Recall* is the ratio of the number of relevant documents retrieved to the total number of relevant documents in the collection, and *precision* is the ratio of the number of relevant documents retrieved to the total number of documents retrieved.

Standardized evaluation of IR began in 1992 with the initiation of the annual Text Retrieval Conference (TREC) sponsored by the Defense Advanced Research Projects Agency (DARPA) and the National Institute of Standards and Technology (NIST) [26]. TREC participants competitively index a large text collection (*guidelines in size*) and are provided search statements and relevancy judgments in order to judge the success of their approaches. Another DARPA-sponsored effort in standardization is being led by the TIPSTER working group [57]. The focus of this group is to specify an *architecture* of an IR system (a set of protocols for document processing) without legislating how that architecture should be implemented. Participants try to determine ways of integrating new methods of IR using a consistent interface.

The purpose of this paper is to show how linear algebra can be used in automated IR. The most basic mechanism is the *vector space model* [50, 18] of IR, in which each document is encoded as a vector, where each vector component reflects the importance of a particular term in representing the semantics or meaning of that document. The vectors for all documents in a database are stored as the columns of a single matrix. In section 2 of this paper, we show how to translate a collection of documents into a matrix and how to compare a user's query to those documents through basic vector operations. The SMART (System for the Mechanical Analysis and Retrieval of Text) system, introduced in 1983 [50], was one of the first to use the vector space model. The SMART system [18], tuned using sophisticated heuristic techniques, has been a top performer at TREC conferences.

The newer method of latent semantic indexing (LSI) or latent semantic analysis (LSA) is a variant of the vector space model in which a low-rank approximation to the vector space representation of the database is employed [9, 19]. That is, we replace the original matrix by another matrix that is as close as possible to the original matrix but whose column space is only a subspace of the column space of the original matrix. Reducing the *rank* of the matrix is a means of removing extraneous information or *noise* from the database it represents. Rank reduction is used in various applications of linear algebra and statistics [14, 27, 31] as well as in image processing [2], data compression [46], cryptography [43], and seismic tomography [17, 52]. LSI has achieved average or above average performance for several TREC collections [21, 22].

In this paper, we do not review LSI but rather show how to apply the vector space model directly to a low-rank approximation of the database matrix. The operations performed in this version of the vector space model admit an easier geometric interpretation than do those underlying LSI. We refer the interested reader to [9] and [19] for the details of LSI.

We begin our exposition with the QR factorization the orthogonal factorization with which most students are familiar. While this factorization has not actually been used in IR methods tested to date, it suffices for showing the features of rank reduction, white being simpler than the singular value decomposition (SVD). In section 3, we show how the QR factorization of the database matrix can be used to provide a geometric interpretation of the vector space model. In section 4, we demonstrate how using the factorization to reduce the rank of the matrix can help to account for uncertainties in the database.

In sections 5 and 6, we move on to combine the vector space model with the SVD. The SVD is a form of orthogonal matrix factorization that is more powerful

This is why  
QR is  
considered here

than the QR factorization. Although the SVD is not often included in introductory linear algebra courses, students comfortable with the QR factorization should be able to read and understand these sections. We first introduce the SVD and compare the low-rank approximations computed from the SVD and the QR factorization. We then explain how to formulate the comparisons of queries and documents via the SVD.

In section 7, we explain what motivates the use of the SVD in place of the QR factorization in practice by showing how relationships between terms can be discovered in the vector space model. Such comparisons aid in the refinement of searches based on the vector space model. The SVD allows such comparisons of terms with terms as well as documents with documents, while the QR factorization permits only the latter. In section 8, we depart from the basic mathematics to cover the more advanced techniques necessary to make vector space and SVD-based models work in practice. Finally, in section 9, we provide a brief outline of further reading material in IR.

Sections 2–7 of this paper should be accessible to anyone familiar with orthogonal factorization (like the QR factorization). Section 8 is more difficult; the recent research results and questions about the practical implementation details of SVD-based models in it may be challenging reading for students.

## 2. The Vector Space Model.

**2.1. A Vector Space Representation of Information.** In the vector space IR model, a vector is used to represent each item or document in a collection. Each component of the vector reflects a particular concept, key word, or term associated with the given document. The value assigned to that component reflects the importance of the term in representing the semantics of the document. Typically, the value is a function of the frequency with which the term occurs in the document or in the document collection as a whole [20, 55]. Suppose a document is described for indexing purposes by the three terms *linear*, *algebra*, and *matrix*. It can then be represented by a vector in the three corresponding dimensions. Figure 2.1 depicts that vector when the terms have respective weights 0.5, 2.5, and 5.0. In this case, the word *algebra* is the most significant term in the document, with *linear* of secondary importance and *matrix* of even less importance.

A database containing a total of  $d$  documents described by  $t$  terms is represented as a  $t \times d$  term-by-document matrix  $A$ . The  $d$  vectors representing the  $d$  documents form the columns of the matrix. Thus, the matrix element  $a_{ij}$  is the weighted frequency at which term  $i$  occurs in document  $j$  [9]. In the parlance of the vector space model, the columns of  $A$  are the *document vectors*, and the rows of  $A$  are the *term vectors*. The semantic content of the database is wholly contained in the column space of  $A$ , meaning that the document vectors span that content. Not every vector represented in the column space of  $A$  has a specific interpretation in terms of the document collection itself (i.e., a linear combination of vectors corresponding to two document titles may not translate directly into a meaningful document title). What is important from an IR perspective, however, is that we can exploit geometric relationships between document vectors to model similarities and differences in content. We can also compare term vectors geometrically in order to identify similarities and differences in term usage.

A variety of schemes are available for weighting the matrix elements. The elements  $a_{ij}$  of the term-by-document matrix  $A$  are often assigned two-part values  $a_{ij} = i_j g_j$ . In this case, the factor  $g_j$  is a *global* weight that reflects the overall value of term  $i$  as an indexing term for the entire collection. As one example, consider a very common term like *computer* within a collection of articles on personal computers.



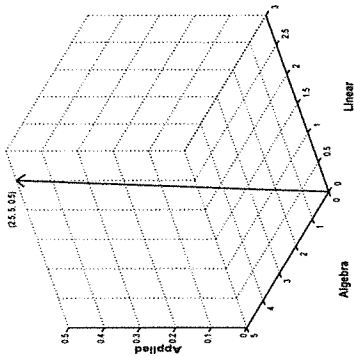


Fig. 2.1 Vector representation of applied linear algebra.

It is not important to include that term in the description of a document as all of the documents are known to be about computers (whether or not they use the actual term *computer*) so a small value of the global weight  $g_i$  is appropriate.

The factor  $l_{ij}$  is a *local* weight that reflects the importance of term  $i$  within document  $j$  itself. Local weights range in complexity from simple binary values (0 or 1) to functions involving logarithms of term frequencies. The latter functions have a smoothing effect in that high-frequency terms having limited discriminatory value are assigned low weights. Global weighting schemes range from simple normalizations to advanced statistics-based approaches. See [20] and [55] for more details about term weighting.

For text collections spanning many contexts (e.g., an encyclopedia), the number of terms is often much greater than the number of documents:  $t \gg d$ . In the case of the Internet, the situation is reversed. A term-by-document matrix using the content of the largest English language dictionary as terms and the set of all Web pages as documents would be about  $300,000 \times 300,000,000$  [4, 13, 35]. As a document generally uses only a small subset of the entire dictionary of terms generated for a given database, most of the elements of a term-by-document matrix are zero.

In a vector space IR scheme, a user queries the database to find relevant documents, somehow using the vector space representation of those documents. The query is a set of terms, perhaps with weights, represented just like a document. Again, it is likely that many of the terms in the database do not appear in the query, meaning that many of the query vector components are zero. Query matching is finding the documents most similar to the query in use and weighting of terms. In the vector space model, the documents selected are those geometrically closest to the query according to some measure.

One common measure of similarity is the cosine of the angle between the query and document vectors. If the term-by-document matrix  $A$  has columns  $a_j, j = 1, \dots, d$ ,

these  $d$  cosines are computed according to the formula

$$\cos \theta_j = \frac{a_j^T q}{\|a_j\|_2 \|q\|_2} = \frac{\sum_{i=1}^t a_{ij} q_i}{\sqrt{\sum_{i=1}^t a_{ij}^2} \sqrt{\sum_{i=1}^t q_i^2}} \quad (2.1)$$

for  $j = 1, \dots, d$ , where the Euclidean vector norm  $\|x\|_2$  is defined by  $\|x\|_2 = \sqrt{x^T x} = \sqrt{\sum_{i=1}^t x_i^2}$  for any real  $t$ -dimensional vector  $x$ . Because the query and document vectors are typically sparse, the dot product and norms in (2.1) are generally inexpensive to compute. Furthermore, the document vector norms  $\|a_j\|_2$  need be computed only once for any given term-by-document matrix. Note that multiplying either  $a_j$  or  $q$  by a constant does not change the cosine value. Thus, we may scale the document vectors or queries by any convenient value. Other similarity measures are reviewed in [30].

**2.2. An Example.** Figure 2.2 demonstrates how a simple collection of five titles described by six terms leads to a  $6 \times 5$  term-by-document matrix. Because the content of a document is determined by the relative frequencies of the terms and not by the total number of times particular terms appear, the matrix elements in this example are scaled so that the Euclidean norm of each column is 1. That is,  $\|a_j\|_2 = 1$  for columns  $a_j, j = 1, \dots, 5$ . In this way, we use *term frequency* as the local weight  $l_{ij}$  and apply no global weighting (i.e.,  $g_i = 1$ ).

The choice of terms used to describe the database determines not only its size but also its utility. In our example, we used only the terms directly related to cooking, meaning that the reader interested in *French cooking* in particular would have no way of retrieving relevant documents. In this case, adding the terms *French* and *Viennoise* to describe the nationalities covered would broaden the representation of the database semantics in a helpful way. On the other hand, including very common terms like *to* or *the* would do little to improve the quality of the term-by-document matrix. The process of excluding such high-frequency words is known as *stoplisting* [25].

In constructing a term-by-document matrix, terms are usually identified by their word stems [33]. In our example, the word *pastries* counts as the term *pastry*, and the word *baking* counts as the term *bake*. The use of *stemming* in IR dates back to the 1960s [40]. Stemming reduces storage requirements by decreasing the number of words maintained [48].

**2.3. Query Matching.** Using the small collection of titles from Figure 2.2, we can illustrate query matching based on angles in a six-dimensional vector space. Suppose that a user in search of cooking information initiates a search for books about *baking bread*. The corresponding query would be written as the vector

$$q^{(1)} = (1 \ 0 \ 1 \ 0 \ 0 \ 0)^T$$

with nonzero entries for the terms *baking* and *bread*. The search for relevant documents is carried out by computing the cosines of the angles  $\theta_j$  between the query vector  $q^{(1)}$  and the document vectors  $a_j$  by (2.1). A document is returned as relevant only if the cosine of the angle it makes with the query vector is greater than some threshold or cutoff value. A practical implementation might use a stringent cutoff like 0.9 [9], but for our small example we use a cosine threshold of 0.5.

For the query  $q^{(1)}$ , the only nonzero cosines are  $\cos \theta_1 = 0.8165$  and  $\cos \theta_4 = 0.5774$ . Hence, all of the documents concerning baking bread (the first and fourth)

enough to  
describe to  
see 120.55

The  $t = 6$  terms:

- T1: book(s)ing)
- T2: recipes
- T3: bread
- T4: cake
- T5: pastri(y)ies)
- T6: pie

The  $d = 5$  document titles:

- D1: How to Bake Bread Without Recipes
- D2: The Classic Art of Viennese Pastry
- D3: Numerical Recipes: The Art of Scientific Computing
- D4: Breads, Pastries, Pies and Cakes: Quantity Baking Recipes
- D5: Pastry: A Book of Best French Recipes

The  $6 \times 5$  term-by-document matrix before normalization, where the element  $a_{ij}$  is the number of times term  $i$  appears in document title  $j$ :

$$\hat{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The  $6 \times 5$  term-by-document matrix with unit columns:

$$A = \begin{pmatrix} 0.5774 & 0 & 0 & 0.4082 & 0 \\ 0.5774 & 0 & 1.0000 & 0.4082 & 0.7071 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4082 & 0 \\ 0 & 1.0000 & 0 & 0.4082 & 0.7071 \\ 0 & 0 & 0 & 0.4082 & 0 \end{pmatrix}$$

Fig. 2.2 The construction of a term-by-document matrix  $A$ .

are returned as relevant. The second, third, and fifth documents, which concern neither of these topics, are correctly ignored.

If the user had simply requested books about *baking*, however, the results would have been markedly different. In this case, the query vector is given by

$$q^{(2)} = (1 \ 0 \ 0 \ 0 \ 0 \ 0)^T,$$

and the cosines of the angles between the query and five document vectors are, in

order, 0.5774, 0, 0, 0.4082, and 0. Only the first document, a book about baking bread, makes the cosine cutoff. The fourth document, which is in fact a more comprehensive reference about baking, is not returned as relevant.

The IR community has developed a variety of approaches to respond to such failures of the basic vector space model. These techniques typically affect how the data are represented in the term-by-document matrix. Examples include term weighting schemes, use of a *controlled vocabulary* (a specified set of allowed terms [33]), and replacing the exact term-by-document matrix by a low-rank approximation to that matrix. The latter approach is the basis of the method of LSI [9, 19] and of the new method described in this paper. In sections 3 to 6 of this paper, we work through the process of rank reduction, first using the QR factorization and then proceeding to the perhaps less familiar SVD used in LSI. In both cases, we use the factorizations to illustrate the geometry of query matching and to explain what can constitute a reasonable approximation to the term-by-document matrix. The latter is important as proponents of LSI argue that the rank-reduction step serves to remove noise from the database representation.

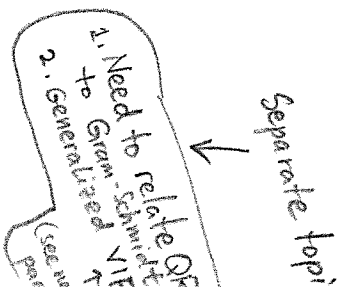
**3. The QR Factorization.** In this section, we show how the QR factorization can be used to identify and remove redundant information in the matrix representation of the database. In linear algebra terms, we identify the rank of the term-by-document matrix. This process leads us directly to a geometric interpretation of the vector space model. In section 4, we show how to lower the rank of the matrix further by removing components that can be attributed to the natural uncertainties present in any large database. The rank-reduction steps allow us to set portions of the matrix to zero and thus to ignore them in subsequent computations. Doing so lowers the cost of query matching and helps to recoup some of the expense of computing the factorization.

Note that the  $6 \times 5$  term-by-document matrix of the example in Figure 2.2 is of rank 4 because column 5 is the sum of columns 2 and 3. Even greater dependence can be expected in practice: a database of Internet sites can contain different editions of the same book, and a database of Internet sites can contain several mirrors of the same Web page. As in our example, dependencies can also involve more than simple copies of information: binary vectors representing the documents applied linear algebra and computer graphics sum to the binary vector representing linear algebra applied to computer graphics (where the proposition to is not considered to be a term), so any database containing all three documents would have dependencies among its columns.

**3.1. Identifying a Basis for the Column Space.** Our first step in the rank-reduction process is to identify dependence between the columns or rows of the term-by-document matrix. For a rank  $r_A$  matrix, the  $r_A$  basis vectors of its column space serve in place of its  $d$  column vectors to represent its column space. One set of basis vectors is found by computing the QR factorization of the term-by-document matrix

$$A = QR$$

where  $R$  is a  $t \times d$  upper triangular matrix and  $Q$  is a  $t \times t$  orthogonal matrix. A square matrix  $Q$  is *orthogonal* if its columns are orthonormal. That is, if  $q_j$  represents a column of an orthogonal matrix  $Q$ , it has unit Euclidean norm ( $\|q_j\| = \sqrt{q_j^T q_j} = 1$  for  $j = 1, \dots, t$ ) and it is orthogonal to all other columns of  $Q$  ( $\sqrt{q_i^T q_j} = 0$  for all  $i \neq j$ ). The rows of  $Q$  are also orthonormal, meaning that  $Q^T Q = Q Q^T = I$ .



This factorization exists for any matrix  $A$ . See [26] for methods of computing the QR factorization. The relation  $A = QR$  shows that the columns of  $A$  are all linear combinations of the columns of  $Q$ . Thus, a subset of  $r_A$  of the columns of  $Q$  forms a basis for the column space of  $A$ .

We now demonstrate how to identify the basis vectors of the example term-by-document matrix  $A$  from Figure 2.2 by using the QR factorization. If  $A = QR$ , the factors are

$$(3.1) \quad Q = \begin{pmatrix} -0.5774 & 0 & -0.4082 & 0 & -0.7071 & 0 \\ -0.5774 & 0 & 0.8165 & 0 & 0.0000 & 0 \\ -0.5774 & 0 & -0.4082 & 0 & 0.7071 & 0 \\ 0 & 0 & 0 & -0.7071 & 0 & -0.7071 \\ 0 & -1.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.7071 & 0 & 0.7071 \end{pmatrix}$$

$$(3.2) \quad R = \begin{pmatrix} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ 0 & 0 & 0 & 0 & -0.5774 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In (3.1), we have partitioned the matrix  $Q$  to separate the first four column vectors from the remaining columns. In (3.2), we have partitioned the matrix  $R$  to separate the nonzero part from the  $2 \times 5$  zero part. We now rewrite the factorization  $A = QR$  as

$$(3.3) \quad A = \begin{pmatrix} Q_A & Q_A^\perp \end{pmatrix} \begin{pmatrix} R_A \\ 0 \end{pmatrix} = Q_A R_A + Q_A^\perp \cdot 0 = Q_A R_A,$$

where  $Q_A$  is the  $6 \times 4$  matrix holding the first four columns of  $Q$ ,  $Q_A^\perp$  is the  $6 \times 2$  remaining submatrix of  $Q$ , and  $R_A$  covers the nonzero rows of  $R$ . This partitioning clearly reveals that the columns of  $Q_A^\perp$  do not contribute to the value of  $A$  and that the ranks of  $A$ ,  $R$ , and  $R_A$  are equal. Thus, the four columns of  $Q_A$  constitute a basis for the column space of  $A$ .

It is important to note here that the clear partitioning of  $R$  into zero and nonzero parts is a feature of the particular matrix  $A$ . In general, it is necessary to use column pivoting during the QR factorization to ensure that the zeros appear at the bottom of the matrix [26]. When column pivoting is used, the computed factorization is  $AP = QR$ , where  $P$  is a permutation matrix. With column pivoting, the first  $r_A$  columns of  $Q$  form a basis for the column space of the rank  $r_A$  matrix  $A$ , and the elements of the first  $r_A$  rows of  $R$  provide the coefficients for the linear combinations of those basis vectors that constitute the columns of  $A$ . In particular, if  $Q_A$  is the  $r_A \times r_A$  matrix having the basis vectors as columns and if  $r_j$  represents the  $j$ th column of the matrix  $R$ , the  $j$ th column of  $AP$  is given by the matrix-vector product  $AP_{c_j} = Q_A r_j$ . The remaining columns of  $Q$  (the columns of  $Q_A^\perp$ ) are a basis for the orthogonal complement of the column space of  $A$  and so of the column space of  $A$ . Column pivoting provides important numerical advantages without changing the database, as permuting the columns of  $A$  results only in a reordering of the document vectors. Because they describe the same constructs, we henceforth use the matrix  $A$  in place of  $AP$  for clarity of presentation.

*Need a deeper strategy.*  
*This is a generalization for rank  $r_A$ .*

*The rank*

For  $(YA)_5 = \begin{pmatrix} -0.707 & & & & \\ -0.408 & & & & \\ 0 & & & & \\ -0.577 & & & & \end{pmatrix}; \quad Q_A^T \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -0.577 \\ 0.908 \\ -0.408 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -0.577 \\ 0 \\ 0 \\ -0.408 \\ 0 \end{pmatrix}$

$\|(YA)_5\|_2 = 1, \quad \|q^{(0)}\|_2 = \sqrt{2}; \quad (YA)_5^T \cdot (QA q^{(0)}) = 0.707 \cdot 2 \cdot 0.577 = \sqrt{2} \cdot 0.577.$

The semantic content of a database is fully described by any basis for the column space of the associated term-by-document matrix, and query matching proceeds with the factors  $QR$  in place of the matrix  $A$ . The cosines of the angles  $\theta_j$  between a query vector  $q$  and the document vectors  $a_j$  are then given by

$$(3.4) \quad \cos \theta_j = \frac{a_j^T q}{\|a_j\| \|q\|} = \frac{(QA r_j)^T q}{\|QA r_j\| \|q\|} = \frac{r_j^T (Q_A^T q)}{\|r_j\| \|q\|}$$

for  $j = 1, \dots, d$ . In this relation, we have used the fact that multiplying a vector by any matrix with orthonormal columns leaves the vector norm unchanged; that is,

$$\|Q_A r_j\|_2 = \sqrt{(Q_A r_j)^T Q_A r_j} = \sqrt{r_j^T Q_A^T Q_A r_j} = \sqrt{r_j^T I r_j} = \sqrt{r_j^T r_j} = \|r_j\|_2.$$

We now revisit the example term-by-document matrix from Figure 2.2 using the query vector  $q^{(0)}$  (baking bread) and observe that there is no loss of information in using its factored form. As expected, the cosines computed via (3.4) are the same as those computed using (2.1): 0.8165, 0, 0, 0.5774, and 0.

**3.2. The Geometry of the Vector Space Model.** The partitioned representation of the term-by-document matrix in (3.3) also aids in a geometric interpretation of the query matching procedure. Note that, for the orthogonal matrix  $Q$ ,

$$I = QQ^T = (Q_A \quad Q_A^\perp) (Q_A \quad Q_A^\perp)^T = Q_A Q_A^T + Q_A^\perp (Q_A^\perp)^T.$$

Therefore, we can write the query vector  $q$  as the sum of its components in the column space of  $A$  and in the orthogonal complement of the column space as follows:

$$(3.5) \quad \begin{aligned} q &= Iq = QQ^T q \\ &= (Q_A Q_A^T + Q_A^\perp (Q_A^\perp)^T) q \\ &= Q_A Q_A^T q + Q_A^\perp (Q_A^\perp)^T q \\ &\equiv q_A + q_A^\perp. \end{aligned}$$

*defines  $q_A, q_A^\perp$*

The column space component  $q_A = Q_A Q_A^T q$  is called the *orthogonal projection* of  $q$  into the space spanned by the columns of  $Q_A$ . Note that  $q_A$  is in fact the closest approximation of the query vector  $q$  in the column space of  $A$ . More precisely,

$$\|q - q_A\|_2 = \min(\|q - x\|_2, \quad x \text{ from the column space of } A).$$

The proof relies on the fact that if the vectors  $q_A$  and  $x$  are both in the column space of  $A$ , the vector  $q_A - x$  is also. The vector  $q - q_A \equiv q_A^\perp$  is orthogonal to any vector in that space by definition. Using the Pythagorean theorem,

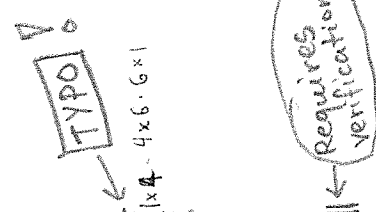
$$\|q - x\|_2^2 = \|q - q_A + q_A - x\|_2^2 = \|q - q_A\|_2^2 + \|q_A - x\|_2^2 \geq \|q - q_A\|_2^2.$$

Substituting (3.5) into (3.4) reveals that only the component  $q_A$  actually contributes to the dot products used to compute the cosines between the query and document vectors:

$$\cos \theta_j = \frac{a_j^T q_A + a_j^T q_A^\perp}{\|a_j\| \|q\|} = \frac{a_j^T q_A + a_j^T Q_A^\perp (Q_A^\perp)^T q}{\|a_j\| \|q\|} = \frac{a_j^T q_A}{\|a_j\| \|q\|}$$

Because  $a_j$  is a column of  $A$ , it is orthogonal to the columns of  $Q_A^\perp$ , which implies that  $a_j^T Q_A^\perp = 0$  and that the cosine formula simplifies to

$$\cos \theta_j = \frac{a_j^T q_A + 0 \cdot (Q_A^\perp)^T q}{\|a_j\| \|q\|} = \frac{a_j^T q_A}{\|a_j\| \|q\|}.$$



One interpretation of this result is that the user's imperfect query is automatically replaced in the dot product computation with its best approximation from the content of the database. The component  $q_1^*$ , which cannot share content with any part of the column space of  $A$ , is ignored. If we take that observation one step further, we can replace  $q$  with its projection altogether and compute a new measure of similarity:

$$(3.6) \quad \cos \theta_j' = \frac{q_1^* q_1}{\|q_1\| \|q_1^*\|}$$

That is, we compare the projection of the user's query to the document vectors. For a given index  $j$ , the two cosines are related by

$$(3.7) \quad \cos \theta_j = \cos \theta_j' \frac{\|q_1\|_2}{\|q\|_2} = \cos \theta_j' \frac{\|q_1\|_2}{\sqrt{\|q_1\|_2^2 + \|q_1^*\|_2^2}}$$

As the ratio of norms on the right-hand side of (3.7) is bounded above by 1, the cosines computed using  $q$  are always less than or equal to those computed using  $q_1$ . As a result, a query vector nearly orthogonal to the column space of  $A$  is more likely to be judged relevant when using  $q_1$  than when using  $q$ , even though such a vector has only a tiny component in that space. In other words, while use of (3.6) may help to identify more of the relevant documents, it may also increase the number of irrelevant ones. In IR terminology, this phenomenon is referred to as an increase in recall at the risk of reduced precision [33].

**4. The Low-Rank Approximation.** Up to this point, we have used the QR factorization to explain the geometry of the query matching procedure. In addition, the QR factorization gives us a means of dealing with uncertainties in the database. Just as measurement errors can lead to uncertainty in experimental data, the very process of indexing the database can lead to uncertainty in the term-by-document matrix. A database and its matrix representation may be built up over a long period of time, by many people with different experiences and different opinions about how the database content should be categorized. For instance, in the example of Figure 2.2, one could argue that the fifth document is relevant to *baking pastry* since it is about *pastry recipes*, which are simply instructions for *baking pastry*. Under that interpretation the (un-normalized) term-by-document matrix  $A$  would have the entry  $A_{55} = 1$ . Because the best transition from data to matrix is subject to interpretation, a term-by-document matrix  $A$  might be better represented by a matrix sum  $A + E$ , where the *uncertainty* matrix  $E$  may have any number of values reflecting missing or incomplete information about documents or even different opinions on the relevancy of documents to certain subjects.

Now, if we accept the fact that our matrix  $A$  is only one representative of a whole family of relatively close matrices representing the database, it is reasonable to ask if it makes sense to attempt to determine its rank exactly [56]. For instance, if we find the rank  $r_A$  and, using linear algebra, conclude that changing  $A$  by adding a *small* change  $E$  would result in a matrix  $A + E$  of lesser rank  $k$ , then we may as well argue that our problem has a rank- $k$  matrix representation and that the column space of  $A$  is not necessarily the best representation of the semantic content of the database. Next we show how lowering the rank may help to remove extraneous information or noise from the matrix representation of the database.

To proceed, we need a notion of the size of a matrix. In particular, we need to be able to say when a matrix is *small* in comparison to another matrix. If we generalize

Let's explain this in detail or skip this part altogether

i.e. the basis of columns (the documents)

the Euclidean vector norm to matrices, the result is the so-called Frobenius matrix norm, which is defined for the real  $t \times d$  matrix  $X$  by

$$(4.1) \quad \|X\|_F = \sqrt{\sum_{i=1}^t \sum_{j=1}^d x_{ij}^2}$$

The Frobenius norm can also be defined in terms of the matrix trace  $\text{Trace}(X)$ , which equals the sum of the diagonal elements of the matrix  $X^T X$ :

$$(4.2) \quad \|X\|_F = \sqrt{\text{Trace}(X^T X)} = \sqrt{\text{Trace}(X X^T)}$$

Using the latter definition, we show that premultiplying the matrix  $X$  by a  $t \times t$  orthogonal matrix  $O$  leaves the Frobenius norm unchanged:

$$\|OX\|_F = \sqrt{\text{Trace}(O^T O X^T O X)} = \sqrt{\text{Trace}(X^T O^T O X)} = \sqrt{\text{Trace}(X^T X)} = \|X\|_F$$

Similarly,  $\|XV\|_F = \|X\|_F$  for any orthogonal  $d \times d$  matrix  $V$ .

Our aim is to find a reasonable low-rank approximation to the matrix  $A$ . We focus on the upper triangular matrix  $R$ , recalling that the ranks of  $R$  and  $A$  are equal. While the rank of  $A$  is not generally obvious, the rank of  $R$  is easy to determine, as it is equal to the number of nonzero entries on the diagonal. The QR factorization with column pivoting aids us in manipulating the rank of  $R$ , as it tends to separate the large and small parts of the matrix, pushing the larger entries toward the upper left corner of the matrix and the smaller ones toward the lower right. If this separation is successful, the matrix  $R$  can be partitioned to isolate the small part. For example, the factor  $R$  for our example problem can be partitioned as follows:

$$R = \begin{pmatrix} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

Under this partitioning, the submatrix  $R_{22}$  is a relatively small part of the matrix  $R$ . Specifically,  $\|R_{22}\|_F / \|R\|_F = 0.5774/2.2361 = 0.2582$ .

We now create a new upper triangular matrix  $\hat{R}$  by setting the small matrix  $R_{22}$  equal to the zero matrix. The new matrix  $\hat{R}$  has rank 3, as does the matrix  $A + E = \hat{Q}\hat{R}$ . The uncertainty matrix  $E$  is then given by the difference

$$\begin{aligned} E &= (A + E) - \hat{A} \\ &= Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} - Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \\ &= Q \begin{pmatrix} 0 & 0 \\ 0 & -R_{22} \end{pmatrix}. \end{aligned}$$

Note that

$$\|E\|_F = \left\| \begin{pmatrix} 0 & 0 \\ 0 & -R_{22} \end{pmatrix} \right\|_F = \|R_{22}\|_F.$$

Verify that  $\|R_{11}\|_F < \|R\|_F$

← since  $R$  is upper-tri

Because  $\|A\|_F = \|R\|_F$ ,  $\|E\|_F/\|A\|_F = \|R_{22}\|_F/\|R\|_F = 0.2582$ . In words, making a 26% relative change in the value of  $R$  makes the same-sized change in  $A$ , and that change reduces the ranks of both matrices by 1. Recall that uncertainties of roughly this order may be introduced simply by disagreement between indexers [28]. Thus, we may deem it acceptable to use the rank-3 approximation  $A + E$  in place of the original term-by-document matrix  $A$  for query matching. If we compute the cosines using (3.4), we need never compute the matrix  $A + E$  explicitly but rather can use, from its QR factors, the first three columns of  $Q$  and the triangular matrix  $R$  which has three zero rows.

To verify that we have not caused undue loss of accuracy, we return to the example of Figure 2.2 using the matrix  $A + E$  in place of the original term-by-document matrix  $A$ . The cosines computed for query  $q^{(1)}$  (baking bread) are 0.8165, 0, 0, 0.7071, and 0, and the cosines computed for query  $q_2$  (baking) are 0.5774, 0, 0, 0.5000, and 0. In both of these cases, the results are actually improved, meaning that our rank-3 approximation  $A + E$  appears to be a better representation of our database than is the original term-by-document matrix  $A$ .

To push the rank reduction farther, we repartition the matrix  $R$  so that its third row and column are also included in  $R_{22}$ . In this case,  $\|R_{22}\|_F/\|R\|_F = 0.5146$ , and discarding  $R_{22}$  to create a rank-2 approximation of the term-by-document matrix introduces a 52% relative change in that matrix. The cosines for  $q^{(1)}$  are now 0.8165, 0, 0.8165, 0.7071, and 0.4082, and for  $q_2$ , they are 0.5774, 0, 0.5774, 0.5000, and 0.2887. In both cases, some irrelevant documents are incorrectly identified, meaning that the 52% relative change in  $R$  and  $A$  is unacceptably large.

In general, it is not possible to explain why one variant of the term-by-document matrix works better than another for a given set of queries. We have seen, however, that it can be possible to improve the performance of the method by reducing the rank of the term-by-document matrix. Note that even the 26% change that we've chosen as acceptable in our example is quite large in the context of scientific or engineering applications where accuracies of three or more decimal places (0.1% error or better) are typically required.

**5. The Singular Value Decomposition.** In sections 3 and 4, we show how to use the QR factorization in the context of a vector space model of IR. Note that while that approach gives us a reduced-rank basis for the column space of the term-by-document matrix, it gives us no such information about its row space. Thus, there is no mechanism for term-term comparison (as described in section 7 of this paper). In this section, we introduce an alternate approach, based on the SVD, that, while more expensive computationally than the QR factorization [26], simultaneously gives us reduced-rank approximations to both spaces. Furthermore, the SVD has the additional mathematical feature of allowing us to find a rank- $k$  approximation to a matrix  $A$  with minimal change to that matrix for a given value of  $k$ .

The fundamental mathematical construct underlying this new approach is the SVD of the term-by-document matrix  $A$ . That decomposition is written

$$A = U\Sigma V^T,$$

where  $U$  is the  $t \times t$  orthogonal matrix having the left singular vectors of  $A$  as its columns,  $V$  is the  $d \times d$  orthogonal matrix having the right singular vectors of  $A$  as its columns, and  $\Sigma$  is the  $t \times d$  diagonal matrix having the singular values  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(t,d)}$  of  $A$  in order along its diagonal. This factorization exists for any matrix  $A$ . See [26] and the papers cited in section 8.3 of this paper for methods for computing

we do not verify this requires using see (3.4) verify this requires using

need: what is the rank of A? defined on first two lines of next page

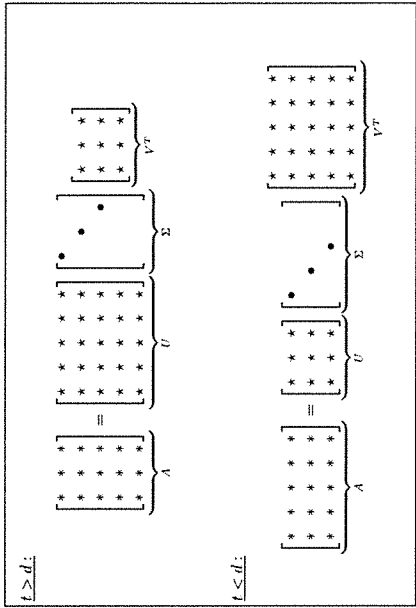


Fig. 5.1 The singular value and singular vector matrices.

the SVD. Figure 5.1 shows the relative sizes of the matrices  $U$ ,  $\Sigma$ , and  $V$  when  $t > d$  and when  $t \leq d$ . All entries not explicitly listed in the singular value matrices are zero.

The rank  $r_A$  of the matrix  $A$  is equal to the number of nonzero singular values. It then follows directly from the orthogonal invariance of the Frobenius norm that  $\|A\|_F$  is defined in terms of those values:

$$\|A\|_F = \|U\Sigma V^T\|_F = \|\Sigma\|_F = \sqrt{\sum_{j=1}^{r_A} \sigma_j^2}.$$

There are many parallels between the SVD  $A = U\Sigma V^T$  and the QR factorization  $AP = QR$ . Just as the rank  $r_A$  of the matrix  $A$  equals the number of nonzero diagonal elements of  $R$ , so does it equal the number of nonzero diagonal elements of  $\Sigma$ . Just as the first  $r_A$  columns of  $Q$  are a basis for the column space of  $A$ , so are the first  $r_A$  columns of  $U$ . (In addition, the first  $r_A$  rows of  $V^T$  are a basis for the row space of  $A$ .) Just as we created a rank- $k$  approximation to  $A$ , where  $k \leq r_A$ , by setting all but the first  $k$  rows of  $R$  equal to zero, so can we create a rank- $k$  approximation  $A_k$  to the matrix  $A$  by setting all but the  $k$  largest singular values of  $A$  equal to zero.

A fundamental difference between the two factorizations is in the theoretical underpinnings of that approximation. More precisely, a classic theorem by Eckart and Young [23, 42] states that the distance between  $A$  and its rank- $k$  approximation is minimized by the approximation  $A_k$ . The theorem further shows how the norm of that distance is related to singular values of  $A$ . It reads

$$\|A - A_k\|_F = \min_{\text{rank}(X) \leq k} \|A - X\|_F = \sqrt{\sigma_{k+1}^2 + \dots + \sigma_d^2}.$$

rank of A is defined on first two lines of next page

Here  $A_k = U_k \Sigma_k V_k^T$ , where  $U_k$  is the  $t \times k$  matrix whose columns are the first  $k$  columns of  $U$ ,  $V_k$  is the  $d \times k$  matrix whose columns are the first  $k$  columns of  $V$ , and  $\Sigma_k$  is the  $k \times k$  diagonal matrix whose diagonal elements are the  $k$  largest singular values of  $A$ .

We return now to the example matrix from Figure 2.2. The SVD of that matrix is  $A = U \Sigma V^T$ , where

$$U = \begin{pmatrix} 0.2670 & -0.2567 & 0.5308 & -0.2847 & -0.7071 & 0 \\ 0.7479 & -0.3981 & -0.5249 & 0.0816 & 0 & 0 \\ 0.2670 & -0.2567 & 0.5308 & -0.2847 & 0.7071 & 0 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 & 0 & -0.7071 \\ 0.5198 & 0.8423 & 0.0838 & -0.1158 & 0 & 0 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 & 0 & 0.7071 \end{pmatrix},$$

$$\Sigma = \begin{pmatrix} 1.6950 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.1158 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.8403 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4195 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$V = \begin{pmatrix} 0.4366 & -0.4717 & 0.3688 & -0.6715 & 0 & 0 \\ 0.3067 & 0.7549 & 0.0998 & -0.2760 & -0.5000 & 0 \\ 0.4412 & -0.3568 & -0.6247 & 0.1945 & -0.5000 & 0 \\ 0.4909 & -0.0346 & 0.5711 & 0.6571 & 0 & 0 \\ 0.5288 & 0.2815 & -0.3712 & -0.0577 & 0.7071 & 0 \end{pmatrix}.$$

This rank-4 matrix has four nonzero singular values, and the two zero rows of  $\Sigma$  signal that the first four columns of  $U$  constitute a basis for the column space of  $A$ .

Using (5.1), we establish that  $\|A - A_3\|_F = \sigma_4 = 0.4195$  and that, since  $\|A\|_F = 2.2461$ ,  $\|A - A_3\|_F / \|A\|_F \approx 0.1876$ . Similarly,  $\|A - A_2\|_F / \|A\|_F \approx 0.4200$ . Therefore, only a 19% relative change is required to reduce the rank of  $A$  from 4 to 3, while it would take a 42% relative change to reduce it from 4 to 2. If we consider 19% a reasonably small change and 42% too large compared to the initial uncertainty in our model, then we can accept rank 3 as the best for our model.

The columns of  $A_3$  span a three-dimensional subspace of the column space of  $A$ . This subspace is a good choice because a relatively small change is required to lower the rank of  $A$  by 1 (thereby obtaining  $A_2$ ), while a relatively large one is required to lower its rank one unit farther. In our model, we believe that this subspace and the corresponding matrix  $A_3$  represent the structure of the database well. In other words, we assume that the *true content* of the database cannot be easily reduced to a lower dimensional subspace. Thus, in our small example, we are able to identify a rank ( $k = 3$ ) that provides a reasonable compromise between accuracy and problem size. How to choose the rank that provides optimal performance of LSI for any given database remains an open question and is normally decided via empirical testing [9]. For very large databases, the number of dimensions used usually ranges between 100 and 300 [38], a choice made for computational feasibility as opposed to accuracy. Using the SVD to find the approximation  $A_k$ , however, guarantees that the approximation is the best we can create for any given choice of  $k$ .

As expected, the relative changes of 19% and 42% required to reduce the rank of the matrix via the SVD are less than the corresponding changes of 26% and 52%

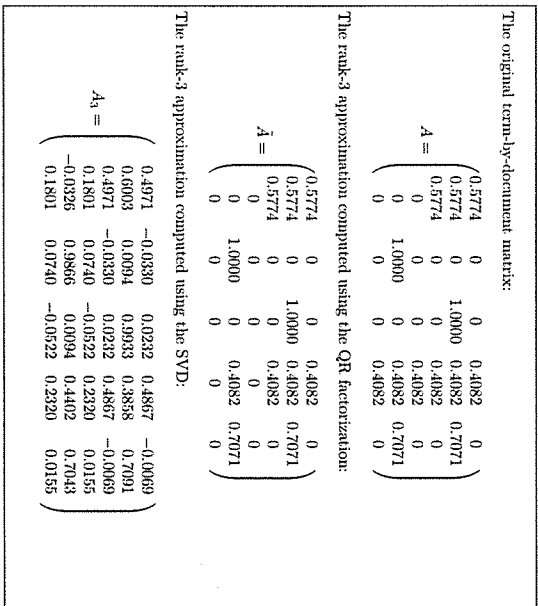


Fig. 5.2 The term-by-document matrix  $A$  and its two rank-3 approximations.

required to do it by the QR factorization. Keeping these numbers in mind, it is interesting to make a visual comparison of the original term-by-document matrix  $A$  and the two rank-3 approximations. As shown in Figure 5.2, the QR-based approximation  $\bar{A}$  looks a lot more like the original matrix than does the more accurate SVD-based approximation  $A_3$ . These results demonstrate the danger of making assumptions about accuracy based on appearance.

Recall that the original term-by-document matrix  $A$  was constructed from term frequencies, thus all of its entries are nonnegative. The presence of negative elements in  $A_3$  is not a problem but rather a reflection of the fact that the entries are linear combinations of the entries of  $A$ . Keep in mind that the database content is modeled by the geometric relationships between the document vectors (columns of  $A_k$ ), not by the individual components of those vectors.

**6. The Reduced-Rank Vector Space Model.** Just as we did for the QR factorization, we can develop a formulation of query matching based on the SVD. One form of the vector space model that uses the SVD is LSI as defined in [9] and [19]. In this section, we introduce a new SVD-based variant of the vector space model that follows



more directly from the preceding discussion than does LSI. We compare a query vector  $q$  to the columns of the approximation  $A_k$  to the term-by-document matrix  $A$ . If we define  $e_j$  to be the  $j$ th canonical vector of dimension  $d$  (the  $j$ th column of the  $d \times d$  identity matrix), the  $j$ th column of  $A_k$  is given by  $A_k e_j$ . The cosines of the angles between the query vector  $q$  and approximate document vectors are then computed by

$$\cos \theta_j = \frac{(A_k e_j)^T q}{\|A_k e_j\| \|q\|} = \frac{(U_k \Sigma_k V_k^T e_j)^T q}{\|U_k \Sigma_k V_k^T e_j\| \|q\|} = \frac{e_j^T V_k \Sigma_k (U_k^T q)}{\| \Sigma_k V_k^T e_j \| \|q\|}$$

for  $j = 1, \dots, d$ . If we define the vector  $s_j = \Sigma_k V_k^T e_j$ , the formula reduces to

$$\cos \theta_j = \frac{s_j^T (U_k^T q)}{\|s_j\| \|q\|}, \quad j = 1, \dots, d, \tag{6.1}$$

and the cosines can be computed without explicitly forming the  $t \times d$  matrix  $A_k$ . The norms  $\|s_j\|$  are computed once for each term-by-document matrix and subsequently used for all queries.

Equation (6.1) is rich in geometry. The  $k$  elements of the vector  $s_j$  are the coordinates of the  $j$ th column of  $A_k$  in the basis defined by the columns of  $U_k$ . In addition, the  $k$  elements of the vector  $U_k^T q$  are the coordinates in that basis of the projection  $U_k U_k^T q$  of the query vector  $q$  into the column space of  $A_k$ . These observations parallel those made for the QR factorization in section 3.2 and so imply that an alternative formula for the comparison could be formed solely from the projected query vector:

$$\cos \theta_j = \frac{s_j^T (U_k^T q)}{\|s_j\| \|U_k^T q\|}, \quad j = 1, \dots, d. \tag{6.2}$$

In this case, the cosine computation uses only  $k$ -dimensional vectors after the one-time computation of  $U_k^T q$ . Because the query vector  $q$  is typically very sparse, the cost of computing  $U_k^T q$  is itself low. For all document vectors,  $\cos \theta_j \geq \cos \theta_j$ , so that recall may be improved at the expense of precision if (6.2) is used in place of (6.1).

Just as was the case for the QR factorization, lowering the rank lowers the cost of query matching. In practice (for LSI), lowering the rank also lowers the cost of the factorization itself. It is never necessary to compute the full SVD of  $A$ —it is sufficient to compute only the select singular values and singular vectors that make up  $\Sigma_k$ ,  $U_k$ , and  $V_k$ .

Let us now revisit the query  $q^{(1)}$  for books about **baking bread** in the example of Figure 2.2. Using the rank-3 approximation  $A_3$  ( $k = 3$ ) and (6.1), the cosines are 0.7327, -0.0469, 0.0330, 0.7161, and -0.0097. The first and fourth books are still correctly identified—this time with nearly equal relevance ratings. The remaining cosines are no longer zero but are still tiny with respect to the cutoff of 0.5, meaning that no irrelevant books are incorrectly returned. Using the second query vector  $q^{(2)}$  about **baking** and  $A_3$  results in the cosines 0.5181, -0.0332, 0.0233, 0.5064, and -0.0069, so that both books about baking are returned, again with very similar relevance ratings.

Running the same tests with the rank-2 approximation  $A_2$  leads to the cosines 0.5181, -0.1107, 0.5038, 0.3940, and 0.2362 for the query vector  $q^{(1)}$  (**baking bread**). Now the unlikely title *Numerical Recipes* is ranked as highly as the appropriate first document, but the fourth document (perhaps the best match) does not make the 0.5 cutoff. The cosines computed for the query vector  $q^{(2)}$  (**baking**) are all less than

1  
 Geometrically  
 The last two  $s_j$ 's  
 are perpendicular  
 to each other  
 so perpendicularity  
 is preserved  
 Skipped, it can be  
 part on p. 348  
 Covered Note

0.5. These nonsensical results confirm our suspicion that  $A_2$  is not a reasonable approximation to  $A$ .

We note again that, unlike the QR factorization, the SVD provides us with a mechanism for low-rank representation of both the row and column spaces of the term-by-document matrix  $A$ . Thus, the SVD-based method can be used both for the query-document comparison just described and for a reduced-rank version of the term-term comparison that is the subject of section 7.

**7. Term-Term Comparison.** Up to this point, we have been concerned with the vector space model as a mechanism for comparing queries with documents. With minor variation, the model can also be used to compare terms with terms. When implemented as part of a search engine, term-term comparison provides a tool to help refine the results of a search automatically. An example is presented in Figure 7.1. The five titles listed in that figure are the results of a search of a large and diverse collection of book titles using the single polysemous key word **run** as the query.

The titles reflect three of the many meanings of the key word. We use term-term comparison to help focus the result.

To begin, we create a new  $7 \times 5$  term-by-document matrix  $G$  using the document vectors returned in the search. Term-term comparison is then carried out by computing the cosines of the angles  $\omega_j$  between all pairs of term vectors  $i$  and  $j$ :

$$(7.1) \quad \cos \omega_j = \frac{(G^T e_i)(G^T e_j)}{\|G^T e_i\| \|G^T e_j\|}$$

for  $i, j = 1, \dots, 7$ , where  $e_i$  denotes the  $i$ th canonical vector of dimension  $t$  (the  $i$ th column of the  $t \times t$  identity matrix). The cosines are listed in the matrix  $C$ , where  $C_{ij} = \cos \omega_j$ . For clarity of presentation, only the entries in the top half of the symmetric matrix  $C$  are shown.

The entry  $C_{ij}$  reveals how closely term  $i$  is associated with term  $j$ . If the entry is near 1, the term vectors for the two terms are nearly parallel and the terms are closely correlated. In this case, the terms are similarly used across the collection of documents and so have similar functions in describing the semantics of these documents. Geometrically, the vectors occupy nearby locations in the row-space of the term-by-document matrix. If the entry  $C_{ij}$  is near zero, the vectors are nearly orthogonal and the corresponding terms are not related. As expected, the cosine matrix  $C$  shows a nonzero cosine for the angles between the term vector for **run** and all other term vectors. It is the cosines of the angles between the remaining term vectors that are more interesting. They show that the remaining term vectors divide into three geometrically distinct groups, where the first group corresponds to terms 2 through 4, the second to terms 5 and 6, and the third to term 7 alone.

The geometric separation of the term vectors translates into three semantically independent groups of terms. The terms **bike**, **endurance**, and **training** identify documents about the sport of running, while the other two groups of terms are associated with other meanings of the term **run**. The process of grouping terms according to their related content in this way is known as *clustering* [33]. Using these results, an automated indexing tool can prompt the user to identify which of the three meanings of **run** is of interest and so help the user to refine the search. Clustering thus serves as one mechanism for dealing with polysemy.

It is this application of the vector space model that justifies use of the SVD for the reduced-rank method when  $A_k = U_k \Sigma_k V_k^T$  replaces  $A$ . Recall that the columns of  $U_k$  formed a basis for the column space of  $A_k$ , and so those columns could be used

p. 336

1th row of G demonstrate

emphasize

The  $t = 7$  terms:

- T1: run(ming)
- T2: bike
- T3: endurance
- T4: training
- T5: hand
- T6: music
- T7: fishes

The  $d = 5$  document titles:

- D1: Complete Triathlon Endurance Training Manual: Swim, Bike, Run
- D2: Lake, River and Sea-Run Fishes of Canada
- D3: Middle Distance Running: Training and Competition
- D4: Music Law: How to Run Your Band's Business
- D5: Running: Learning, Training, Competing

The  $7 \times 5$  term-by-document matrix with unit columns:

$$G = \begin{pmatrix} 0.5000 & 0.7071 & 0.7071 & 0.5774 & 0.7071 \\ 0.5000 & 0 & 0 & 0 & 0 \\ 0.5000 & 0 & 0.7071 & 0 & 0.7071 \\ 0 & 0 & 0 & 0 & 0.5774 \\ 0 & 0 & 0 & 0 & 0.5774 \\ 0 & 0.7071 & 0 & 0 & 0 \end{pmatrix}$$

Cosines of angles between term vectors:

$$C = \begin{pmatrix} 1.0000 & 0.3464 & 0.3464 & 0.7746 & 0.4000 & 0.4000 & 0.4899 \\ 1.0000 & 1.0000 & 0.4472 & 0 & 0 & 0 & 0 \\ 1.0000 & 0.4472 & 1.0000 & 0 & 0 & 0 & 0 \\ 1.0000 & 0 & 0 & 1.0000 & 1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 1.0000 & 1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 & 0 & 1.0000 & 0 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 & 1.0000 \end{pmatrix}$$

Fig. 7.1 Identifying polygamy via term comparison.

In place of the columns of  $A_k$  for query matching. In the same way, the rows of  $V_k$  are a basis for the row space of  $A_k$  and so can replace the rows of  $A_k$  in (7.1). Thus, in a reduced-rank approximation, the cosine becomes

$$\cos w_{ij} = \frac{(e_j^T U_k \Sigma_k V_k^T)(V_k \Sigma_k U_k^T e_i)}{\|V_k \Sigma_k U_k^T e_i\|_2 \|V_k \Sigma_k U_k^T e_j\|_2} = \frac{(e_j^T U_k \Sigma_k)(\Sigma_k U_k^T e_i)}{\|\Sigma_k U_k^T e_i\|_2 \|\Sigma_k U_k^T e_j\|_2}$$



for  $i = 1, \dots, t$  and  $j = 1, \dots, d$ . Defining  $b_j = \Sigma_k U_k^T e_j$ , we have

$$\cos w_{ij} = \frac{U_k^T b_j}{\|b_j\|_2 \|b_j\|_2}$$

for  $i = 1, \dots, t$  and  $j = 1, \dots, d$ .

In addition to the geometric measure of similarity used in Figure 7.1, techniques based on graph-theoretic concepts (e.g., links, circles, connected components) are sometimes used to produce approximate clusters directly from the term frequencies stored in the original term-by-document matrix. Clusters formed by any method may be merged or split depending on the level of similarity among clusters. Clustering information can be used to generate *ad hoc* clusters in which terms are grouped according to how often they re-occur under the assumption that terms that typically appear together should be associated with similar concepts. Such clusters are used for query expansion and refinement. See [33] for a more thorough discussion of term (and document) clustering techniques.

**8. What We Do to Really Make IR Work.** Scientific computing is rarely a straightforward application of textbook algorithms. Implementing vector space methods in IR is no different. In this section, we discuss some techniques used in practice to index and manage large collections of documents using LSI. They also apply directly to the SVD-based method described in section 6. Most of the material in this section is more advanced than that covered in the preceding sections, so we review only the basic ideas and provide references to more detailed explanations.

**8.1. Relevance Feedback.** An ideal IR system would achieve high precision for high levels of recall. That is, it would identify all relevant documents without also returning any irrelevant ones. Unfortunately, due to problems such as polysemy and synonymy (described in section 1), a list of documents retrieved for a given query is almost never perfect, and the user has to ignore some of the items.

In practice, precision can be improved using *relevance feedback* [49], that is, specifying which documents from a returned set are most relevant to the information sought and using those documents to clarify the intent of the original query. The term-term comparison procedure described in section 7 provides one mechanism for relevance feedback by which the user can improve a query based on term clustering information. Relevance feedback can also be carried out in the column space of the term-by-document matrix. In particular, the query can be supplemented with or replaced by the vector sum of the most relevant documents returned in order to focus the search nearer to those document vectors. We now provide a mathematical description of the latter relevance feedback procedure.

Because we are concerned only with information contained within the column space of  $A_k$ , we assume that the original query vector  $q$  lies within that space. If not, we replace the query vector with its projection into that space ( $q \leftarrow U_k U_k^T q$ ). To provide a common ground for the query and document vectors, it is convenient to describe them both in terms of the same basis for the rank- $k$  approximation to the column space. We choose the basis given by the columns of  $U_k$  and then work directly with the  $k$ -dimensional vectors that provide the coordinates of the vectors in that basis instead of working directly with the much larger  $t$ -dimensional query and document vectors. The coordinates of the query vector are the elements of the vector  $U_k^T q$ , and the coordinates of the  $j$ th column  $U_k \Sigma_k V_k^T e_j$  of the matrix  $A_k$  are the elements of the vector  $\Sigma_k V_k^T e_j$ .

can skip