SIAM REVIEW
Vol. 41, No. 2, pp. 335–362

Matrices, Vector Spaces, and Information Retrieval*

Michael W. Berry†
Zlatko Drmač‡
Elizabeth R. Jessup‡

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 those 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.
http://www.siam.org/journals/sirev/41-2/34703.html

†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 ACIR-94-11394.

‡Department of Computer Science, University of Colorado, Boulder, CO 80309-0430 (zlatko@cs.colorado.edu, jessup@cs.colorado.edu). The work of the second author was supported by National Science Foundation grants ACIR-93-57812 and ACIR-96-25912. The work of the third author was supported by National Science Foundation grant ACIR-93-57812 and Department of Energy grant DE-FG03-97ER25325.

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 [13, 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 *expository* 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 pose 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) [29]. TREC participants competitively index a large text collection (gigabytes 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 lead 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, while 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

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 comparison 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 **applied**, **linear**, and **algebra**. 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 **applied** 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 $a_{ij} = l_{ij}g_i$. In this case, the factor g_i 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.

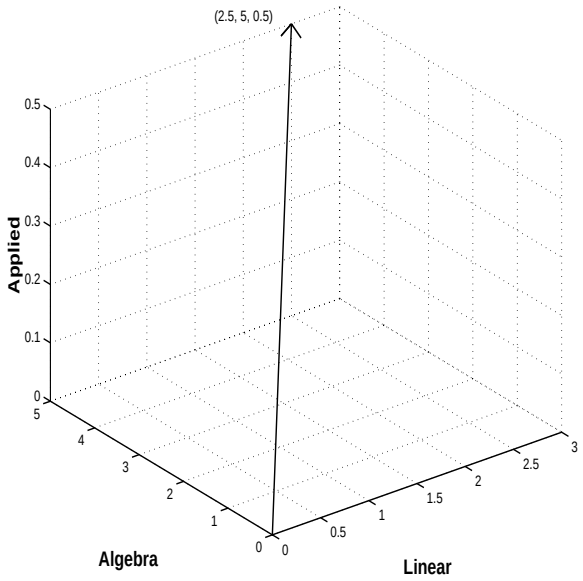

Fig. 2.1 *Vector representation of* **applied linear algebra**.

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 $a_j, j = 1, \ldots, d$,

those d cosines are computed according to the formula

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

for $j = 1, \ldots, d$, where the Euclidean vector norm $\| x \|_2$ is defined by $\| x \|_2 = \sqrt{x^T x} = \sum_{i=1}^t x_i^2$ 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 6×5 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, $j = 1, \ldots, 5$. 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 **Viennese** 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

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

with nonzero entries for the terms **baking** and **bread**. The search for relevant documents is carried out by computing the cosines of the angles θ_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)

The $t = 6$ **terms**:

T1: bak(e,ing)
T2: recipes
T3: bread
T4: cake
T5: pastr(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 6×5 term-by-document matrix before normalization, where the element $\hat{a}_{ij}$ is the number of times term i appears in document title j:

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

The 6×5 term-by-document matrix with unit columns:

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

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

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

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. Those 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 6×5 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 library materials 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 preposition *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 ($\| q_j \|_2 = \sqrt{q_j^T q_j} = 1$ for $j = 1, \ldots, t$) and it is orthogonal to all other columns of Q ($\sqrt{q_j^T q_i} = 0$ 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

$$Q = \left(\begin{array}{rrrr|rr} -0.5774 & 0 & -0.4082 & 0 & -0.7071 & 0 \\ -0.5774 & 0 & 0.8165 & 0 & 0.0000 & 0 \\ -0.5774 & 0 & -0.4082 & 0 & 0.7071 & 0 \\ 0 & 0 & 0 & -0.7071 & 0 & -0.7071 \\ 0 & -1.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.7071 & 0 & 0.7071 \end{array}\right), \tag{3.1}$$

$$R = \left(\begin{array}{rrrrr} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ 0 & 0 & 0 & -0.5774 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right). \tag{3.2}$$

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 2×5 zero part. We now rewrite the factorization $A = QR$ as

$$\begin{aligned} A &= \begin{pmatrix} Q_A & Q_A^{\perp} \end{pmatrix} \begin{pmatrix} R_A \\ 0 \end{pmatrix} \\ &= Q_A R_A + Q_A^{\perp} \cdot 0 = Q_A R_A, \end{aligned} \tag{3.3}$$

where Q_A is the 6×4 matrix holding the first four columns of Q, $Q_A^{\perp}$ is the 6×2 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 $t \times r_A$ matrix having the basis vectors as columns and if r_j represents the jth column of the matrix R, the jth column of AP is given by the matrix–vector product $APe_j = Q_A r_j$. The remaining columns of Q (the columns of $Q_A^{\perp}$) are a basis for the orthogonal complement of the column space of AP 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.

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 θ_j between a query vector q and the document vectors a_j are then given by

$$\cos\theta_j = \frac{a_j^T q}{\| a_j \|_2 \| q \|_2} = \frac{(Q_A r_j)^T q}{\| Q_A r_j \|_2 \| q \|_2} = \frac{r_j^T (Q_A^T q)}{\| r_j \|_2 \| q \|_2} \tag{3.4}$$

for $j = 1, \ldots, d$. 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 $q^{(1)}$ (**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,

$$I = QQ^T = \begin{pmatrix} Q_A & Q_A^\perp \end{pmatrix} \begin{pmatrix} Q_A & Q_A^\perp \end{pmatrix}^T = Q_A Q_A^T + Q_A^\perp (Q_A^\perp)^T.$$

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:

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

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:

$$\cos\theta_j = \frac{a_j^T q_A + a_j^T q_A^\perp}{\| a_j \|_2 \| q \|_2} = \frac{a_j^T q_A + a_j^T Q_A^\perp (Q_A^\perp)^T q}{\| a_j \|_2 \| q \|_2}.$$

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

$$\cos\theta_j = \frac{a_j^T q_A + 0 \cdot (Q_A^\perp)^T q}{\| a_j \|_2 \| q \|_2} = \frac{a_j^T q_A}{\| a_j \|_2 \| q \|_2}.$$

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 $q_A^{\perp}$, which cannot share content with any part of the column space of A, is ignored. If we take that observation one step farther, we can replace q with its projection altogether and compute a new measure of similarity:

$$\cos\theta_j{}' = \frac{a_j^T q_A}{\| a_j \|_2 \| q_A \|_2}. \tag{3.6}$$

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

$$\cos\theta_j = \cos\theta_j{}' \frac{\| q_A \|_2}{\| q \|_2} = \cos\theta_j{}' \frac{\| q_A \|_2}{\sqrt{\| q_A \|_2^2 + \| q_A^{\perp} \|_2^2}}. \tag{3.7}$$

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_A. As a result, a query vector nearly orthogonal to the column space of A is more likely to be judged relevant when using q_A 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* since it is about **pastry recipes**, which are simply instructions for **baking pastry**. Under that interpretation the (unnormalized) term-by-document matrix $\hat{A}$ would have the entry $\hat{A}_{15} = 1$. Because the best translation 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

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

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

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

$$\|X\|_F = \sqrt{\mathrm{Trace}(X^T X)} = \sqrt{\mathrm{Trace}(X X^T)}. \tag{4.2}$$

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

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

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 its 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:

$$R = \left(\begin{array}{rrr|rr} -1.0001 & 0 & -0.5774 & -0.7070 & -0.4082 \\ 0 & -1.0000 & 0 & -0.4082 & -0.7071 \\ 0 & 0 & 0.8165 & 0 & 0.5774 \\ \hline 0 & 0 & 0 & -0.5774 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}.$$

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

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

Note that

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

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 $\tilde{R}$ 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 Σ is the $t \times d$ diagonal matrix having the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(t,d)}$ 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

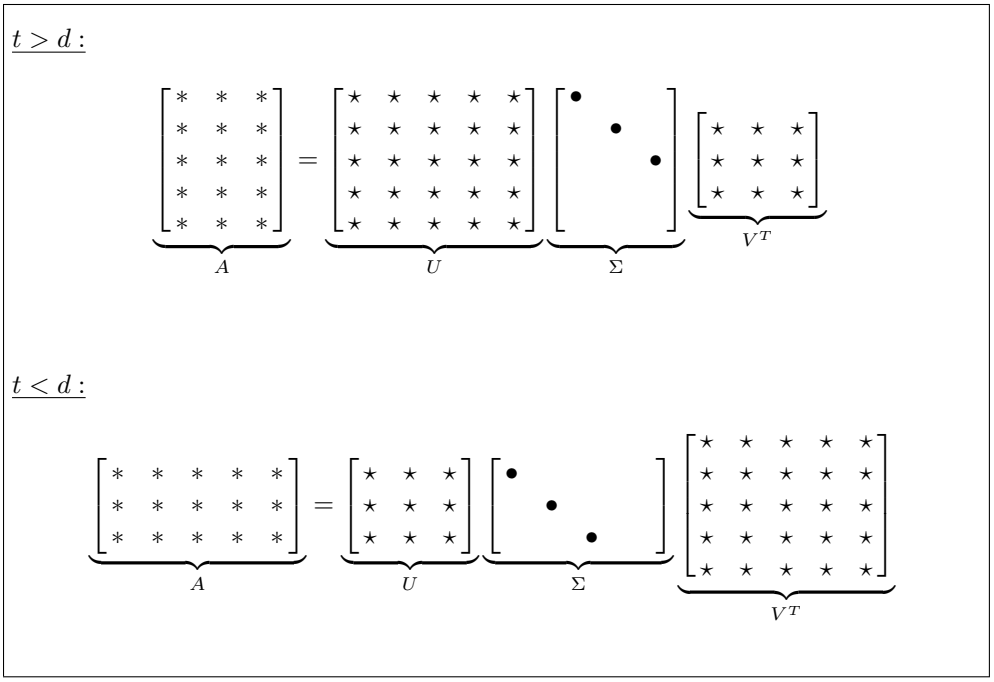

Fig. 5.1 *The singular value and singular vector matrices.*

the SVD. Figure 5.1 shows the relative sizes of the matrices U, Σ, and V when $t > d$ and when $t < d$. 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:

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

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 Σ. 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 approximations 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

$$\|A - A_k\|_F = \min_{\mathrm{rank}(X) \leq k} \|A - X\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_{r_A}^2}. \tag{5.1}$$

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 Σ_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

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

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

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

This rank-4 matrix has four nonzero singular values, and the two zero rows of Σ 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 $\|A\|_F = 2.2361$, $\|A - A_3\|_F / \|A\|_F \approx 0.1876$. 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_3), 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%

The original term-by-document matrix:

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

The rank-3 approximation computed using the QR factorization:

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

The rank-3 approximation computed using the SVD:

$$A_3 = \begin{pmatrix} 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.6003 & 0.0094 & 0.9933 & 0.3858 & 0.7091 \\ 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \\ -0.0326 & 0.9866 & 0.0094 & 0.4402 & 0.7043 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \end{pmatrix}$$

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 $\tilde{A}$ *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_k 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 jth canonical vector of dimension d (the jth column of the $d \times d$ identity matrix), the jth 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

$$\cos\theta_j = \frac{(A_k e_j)^T q}{\| A_k e_j \|_2 \| q \|_2} = \frac{(U_k \Sigma_k V_k^T e_j)^T q}{\| U_k \Sigma_k V_k^T e_j \|_2 \| q \|_2} = \frac{e_j^T V_k \Sigma_k (U_k^T q)}{\| \Sigma_k V_k^T e_j \|_2 \| q \|_2}$$

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

$$\cos\theta_j = \frac{s_j^T (U_k^T q)}{\| s_j \|_2 \| q \|_2}, \quad j = 1, \ldots, d, \tag{6.1}$$

and the cosines can be computed without explicitly forming the $t \times d$ matrix A_k. The norms $\| s_j \|_2$ 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 jth 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:

$$\cos\theta_j{}' = \frac{s_j^T (U_k^T q)}{\| s_j \|_2 \| U_k^T q \|_2}, \quad j = 1, \ldots, d. \tag{6.2}$$

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 Σ_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

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 7×5 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 ω_{ij} between all pairs of term vectors i and j:

$$\cos\omega_{ij} = \frac{(e_i^T G)(G^T e_j)}{\| G^T e_i \|_2 \| G^T e_j \|_2} \tag{7.1}$$

for $i, j = 1, \ldots, 7$, where e_l denotes the lth canonical vector of dimension t (the lth column of the $t \times t$ identity matrix). The cosines are listed in the matrix C, where $C_{ij} = \cos\omega_{ij}$. 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 those 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

The $t = 7$ **terms**:

T1: run(ning)
T2: bike
T3: endurance
T4: training
T5: band
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 7×5 term-by-document matrix with unit columns:

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

Cosines of angles between term vectors:

$$C = \left(\begin{array}{c|cc|c|cc|c} 1.0000 & 0.3464 & 0.3464 & 0.7746 & 0.4000 & 0.4000 & 0.4899 \\ \hline & 1.0000 & 1.0000 & 0.4472 & 0 & 0 & 0 \\ & & 1.0000 & 0.4472 & 0 & 0 & 0 \\ & & & 1.0000 & 0 & 0 & 0 \\ \hline & & & & 1.0000 & 1.0000 & 0 \\ & & & & & 1.0000 & 0 \\ \hline & & & & & & 1.0000 \end{array}\right)$$

Fig. 7.1 *Identifying polysemy 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

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

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

$$\cos \omega_{ij} = \frac{b_i^T b_j}{\| b_i \|_2 \| b_j \|_2}$$

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

In addition to the geometric measure of similarity used in Figure 7.1, techniques based on graph-theoretic concepts (e.g., links, cliques, 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 *statistical thesauri* in which terms are grouped according to how often they co-occur under the assumption that terms that typically appear together should be associated with similar concepts. Such thesauri 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 jth column $U_k \Sigma_k V_k^T e_j$ of the matrix A_k are the elements of the vector $\Sigma_k V_k^T e_j$.

Suppose that the most relevant result of a user's search is the single document a_j. The new and improved query is then the sum

$$\begin{aligned} q_{new} &= U_k U_k^T q + a_j \\ &= U_k U_k^T q + U_k \Sigma_k V_k^T e_j \\ &= U_k (U_k^T q + \Sigma_k V_k^T e_j). \end{aligned}$$

(If the document vector a_j is not in the column space of A_k, it should also be replaced with its projection into that space, $a_j \leftarrow U_k U_k^T a_j$.) If a larger collection of documents is relevant, the new query can be written as

$$q_{new} = q + \sum_{j=1}^{d} w_j a_j = U_k (U_k^T q + \Sigma_k V_k^T w), \tag{8.1}$$

where the vector element w_j is 1 if a_j is relevant and 0 otherwise. If the query vector is replaced by the sum of document vectors, the vector q is replaced by zero in (8.1). The vector $U_k^T q$ was formed in the original cosine computation, so the new query is formed efficiently via the sum of vectors of dimension k. We can now repeat the comparison using q_{new} in place of q. If (as we did for (6.1)) we define the vector $s_j = \Sigma_k V_k^T e_j$, the cosine formula is now

$$\cos \theta_j = \frac{s_j^T (U_k^T q_{new})}{\| s_j \|_2 \, \| q_{new} \|_2}$$

for $j = 1, \ldots, d$. There is empirical evidence that replacing the query with a combination of a few of the most relevant documents returned can markedly improve the performance of LSI in some cases [20].

8.2. Managing Dynamic Collections. Like the weather, databases rarely stay the same. Information is constantly added or removed, meaning that catalogs and indexes become obsolete or incomplete (sometimes in a matter of seconds). For the LSI model, the most obvious approach to accommodating additions (new terms or documents) is to recompute the SVD of the new term-by-document matrix, but, for large databases, this procedure is very costly in time and space. Less expensive alternatives, folding-in and SVD-updating, have been examined in [9], [44], and [54]. The first of these procedures is very inexpensive computationally but results in an inexact representation of the database. It is generally appropriate to fold documents in only occasionally. Updating, while more expensive, preserves (or restores) our representation of the database. In this section, we briefly review both procedures.

8.2.1. Folding-In. Folding a new document vector into the column space of an existing term-by-document matrix amounts to finding coordinates for that document in the basis U_k. The first step in folding a new $t \times 1$ document vector $\hat{p}$ into the column space is to project it onto that space. Let p represent the projection of $\hat{p}$; then, following the discussions in section 6,

$$p = U_k U_k^T \hat{p}. \tag{8.2}$$

This equation shows that the coordinates of p in the basis U_k are given by the elements of the vector $U_k^T \hat{p}$.

The new document is then folded in by appending the k-dimensional vector $U_k^T \hat{p}$ as a new column of the $k \times d$ matrix $\Sigma_k V_k^T$. Because the latter matrix product is not

actually computed, the folding-in is carried out implicitly by appending $\hat{p}^T U_k \Sigma_k^{-1}$ as a new row of V_k to form a new matrix ${V_k}'$. The implicit product $\Sigma_k {V_k}'$ is then the desired result. Note that the matrix ${V_k}'$ is no longer orthonormal. In addition, the row space of the matrix ${{V_k}'}^T$ does not represent the row space of the new term-by-document matrix. Furthermore, if the new document p is nearly orthogonal to the columns of U_k, most information about that document is lost in the projection step.

Similarly, to fold in a $d \times 1$ term vector $\hat{w}$ whose elements specify the documents associated with a term, $\hat{w}$ is projected into the row space of A_k. Let w represent the term projection of $\hat{w}$; then

$$w = V_k V_k^T \hat{w}.$$

The coordinates $V_k^T \hat{w}$ of the projected vector w are then appended to the matrix U_k as a new row. In this case, the orthogonal representation of the column space of the term-by-document matrix breaks down [9, 44].

8.2.2. SVD-Updating. An alternative to folding-in that accounts for the effects that new terms and documents might have on term–document associations while still maintaining orthogonality was first described in [9] and [44]. This approach comprises the following three steps: updating terms, updating documents, and updating term weights. As pointed out by Simon and Zha [54], the operations discussed in [9] and [44] may not produce the exact SVD of the modified reduced-rank LSI model (i.e., A_k from section 4.) Those authors provide alternative algorithms for all three steps of SVD-updating, and we now review them. For consistency with our earlier discussion, we use column pivoting in the QR factorizations, although it is not used in [9], [44], and [54].

Updating Terms. Suppose that r term vectors are added to an existing LSI database of d documents. If T is the $r \times d$ matrix of new term vectors, the new term-by-document matrix is formed by appending T to the rows of the rank-k $t \times d$ matrix $A_k = U_k \Sigma_k V_k^T$. The result is the $(t + r) \times d$ matrix

$$B = \begin{pmatrix} A_k \\ T \end{pmatrix}.$$

By construction, the rank of B is greater than or equal to the rank k of A_k. An approximation to the SVD of B is used to obtain a new rank-k factorization $B_k = U_B \Sigma_B V_B^T$, reflecting the change in semantic structure of the database caused by the addition of terms.

The factorization of B proceeds in two stages. In the first, we create a block trapezoidal matrix pre- and postmultiplied by matrices with orthonormal columns and rows, respectively. In the second, we modify that factorization to produce the approximate SVD of B. To begin, we replace A_k by its SVD and factor out the singular vector matrices:

$$\begin{aligned} B = \begin{pmatrix} A_k \\ T \end{pmatrix} &= \begin{pmatrix} U_k \Sigma_k V_k^T \\ T \end{pmatrix} \\ &= \begin{pmatrix} U_k & 0 \\ 0 & I \end{pmatrix} \left[\begin{pmatrix} \Sigma_k \\ TV_k \end{pmatrix} V_k^T + \begin{pmatrix} 0 \\ T(I - V_k V_k^T) \end{pmatrix} \right] \\ &= \begin{pmatrix} U_k & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_k & 0 \\ TV_k & I \end{pmatrix} \begin{pmatrix} V_k^T \\ T(I - V_k V_k^T) \end{pmatrix} \\ &= \begin{pmatrix} U_k & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_k & 0 \\ TV_k & I \end{pmatrix} \begin{pmatrix} V_k & (I - V_k V_k^T) T^T \end{pmatrix}^T. \end{aligned} \tag{8.3}$$

At this point, the interior matrix in the matrix product is triangular and the left exterior matrix is orthogonal, but the right exterior matrix is not. While the columns of V_k are orthonormal, the columns of $\tilde{V}_k = (I - V_k V_k^T)T^T$ are not. It is, however, the case that the columns of $\tilde{V}_k$ belong to the orthogonal complement of the column space of V_k. (Each column of that matrix is formed by subtracting the projection of a column of T^T into the column space of V_k from that column of T^T.) The remedy comes in the form of a variant of the QR factorization. Let r_V be the rank of $\tilde{V}_k$; then $\tilde{V}_k$ can be factored into the product $\tilde{V}_k \Pi_V = \hat{V}_k R_r$, where Π_V is an $r \times r$ permutation matrix, $\hat{V}_k$ is a $d \times r_V$ matrix having orthonormal columns, and R_r is an $r_V \times r$ upper trapezoidal matrix.

Then, in (8.3),

$$\begin{pmatrix} V_k & (I - V_k V_k^T)T^T \end{pmatrix} = \begin{pmatrix} V_k & \hat{V}_k R_r \Pi_V^T \end{pmatrix},$$

so that

$$\begin{aligned} B &= \begin{pmatrix} U_k & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_k & 0 \\ TV_k & I \end{pmatrix} \begin{pmatrix} V_k & \hat{V}_k R_r \Pi_V^T \end{pmatrix}^T \\ &= \begin{pmatrix} U_k & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_k & 0 \\ TV_k & \Pi_V R_r^T \end{pmatrix} \begin{pmatrix} V_k^T \\ \hat{V}_k^T \end{pmatrix}. \end{aligned}$$

In this factorization, the left matrix is a $(t + r) \times (k + r)$ matrix with orthonormal columns, the interior matrix is a $(k + r) \times (k + r_V)$ block lower trapezoidal matrix, and the right matrix is a $(k + r_V) \times d$ matrix having orthonormal rows. (Note that when column pivoting is used, the small elements of R_r are pushed toward the lower right corner. As discussed in section 3, it may be possible to ignore those elements and thereby reduce the cost of the SVD computation in the following step.)

If we define the SVD of the interior matrix by

$$\hat{B} = \begin{pmatrix} \Sigma_k & 0 \\ TV_k & \Pi_V R_r^T \end{pmatrix} = \begin{pmatrix} P_k & P_k^\perp \end{pmatrix} \begin{pmatrix} \hat{\Sigma}_k & 0 \\ 0 & \hat{\Sigma}_r \end{pmatrix} \begin{pmatrix} Q_k & Q_k^\perp \end{pmatrix}^T,$$

where P_k is a $(k + r) \times k$ matrix with orthonormal columns, Q_k is a $(k + r_V) \times k$ matrix with orthonormal columns, and $\hat{\Sigma}_k$ is a $k \times k$ diagonal matrix, then the best rank-k approximation of B is

$$B_k = \begin{pmatrix} U_k^T & 0 \\ 0 & I_r \end{pmatrix} P_k \ \hat{\Sigma}_k \begin{pmatrix} V_k & \hat{V}_k \end{pmatrix} Q_k \Big)^T. \tag{8.4}$$

Using (8.4), the new factors of rank-k approximation of the updated term-by-document matrix B are

$$U_B = \begin{pmatrix} U_k^T & 0 \\ 0 & I_r \end{pmatrix} P_k \ \text{ and } \ V_B = \begin{pmatrix} V_k & \hat{V}_k \end{pmatrix} Q_k,$$

respectively, and its singular values are $\Sigma_B = \hat{\Sigma}_k$.

Updating Documents. Adding s document vectors to a database is similar to adding r term vectors. Let D denote the $t \times s$ document vectors, where t is the number of terms. The matrix D is appended to the columns of the original A_k matrix so that

$$B = \begin{pmatrix} A_k & D \end{pmatrix},$$

where B is a $t \times (d+s)$ matrix of rank at least k. An approximation to the SVD of B given by $B_k = U_B \Sigma_B V_B^T$ then accounts for changes in the document set [9]. If the QR factorization of $(I - U_k U_k^T)D$ is represented as

$$(I - U_k U_k^T)D\Pi_U = \hat{U}_k R_s,$$

where R_s is an $s \times s$ upper triangular (or trapezoidal) matrix, then it follows [54] that

$$B = \begin{pmatrix} A_k & D \end{pmatrix} = \begin{pmatrix} U_k & \hat{U}_k \end{pmatrix} \begin{pmatrix} \Sigma_k & U_k^T D \\ 0 & R_s \Pi_U^T \end{pmatrix} \begin{pmatrix} V_k^T & 0 \\ 0 & I_s \end{pmatrix}.$$

Using the SVD of

$$\begin{pmatrix} \Sigma_k & U_k^T D \\ 0 & R_s \Pi_U^T \end{pmatrix} = \begin{pmatrix} P_k & P_k^\perp \end{pmatrix} \begin{pmatrix} \hat{\Sigma}_k & 0 \\ 0 & \hat{\Sigma}_s \end{pmatrix} \begin{pmatrix} Q_k & Q_k^\perp \end{pmatrix}^T,$$

where P_k and Q_k are $(s+k) \times k$ matrices with orthonormal columns and $\hat{\Sigma}_k$ is again a $k \times k$ diagonal matrix, the best rank-k approximation of B is given by

$$B_k = \begin{pmatrix} U_k & \hat{U}_k \end{pmatrix} P_k \ \hat{\Sigma}_k \left[\begin{pmatrix} V_k & 0 \\ 0 & I_s \end{pmatrix} Q_k \right]^T. \tag{8.5}$$

Using (8.5), the term and document vectors for the updated LSI model are

$$U_B = \begin{pmatrix} U_k & \hat{U}_k \end{pmatrix} P_k \ \text{ and } \ V_B = \begin{pmatrix} V_k & 0 \\ 0 & I_s \end{pmatrix} Q_k,$$

respectively, and the updated singular values are the diagonal elements of $\Sigma_B = \hat{\Sigma}_k$.

Updating Term Weights. The weights assigned to j of the terms can be changed by computing the matrix sum

$$B = A_k + Y Z^T,$$

where the elements of the $d \times j$ matrix Z specify the differences between the old and new weights for the j terms, and the elements of the $t \times j$ matrix Y are either 0 or 1, depending on which elements of A_k are to be updated [9].

This updating procedure also depends on the QR factorization. In particular, by substituting the two factorizations

$$(I - U_k U_k^T)Y\Pi_Y = Q_Y R_Y,$$
$$(I - V_k V_k^T)Z\Pi_Z = Q_Z R_Z,$$

we can write the updated matrix as the product [54]

$$\begin{aligned} B &= \begin{pmatrix} U_k & Q_Y \end{pmatrix} \left[\begin{pmatrix} \Sigma_k & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} U_k^T Y \\ R_Y \Pi_Y^T \end{pmatrix} \begin{pmatrix} Z^T V_k & \Pi_Z R_Z^T \end{pmatrix} \right] \begin{pmatrix} V_k^T \\ Q_Z^T \end{pmatrix} \\ &= \begin{pmatrix} U_k & Q_Y \end{pmatrix} \hat{B} \begin{pmatrix} V_k^T \\ Q_Z^T \end{pmatrix}. \end{aligned}$$

In this product, the left matrix is a $t \times (k+j)$ matrix with orthonormal columns, the interior matrix $\hat{B}$ is a rank-j update to a $(k+j) \times (k+j)$ diagonal matrix, and the right matrix is a $(k+j) \times d$ matrix having orthonormal rows. Π_Y and Π_Z are $j \times j$ permutation matrices. If the SVD of $\hat{B}$ is

$$\hat{B} = \begin{pmatrix} \hat{U}_k & \hat{U}_k^\perp \end{pmatrix} \begin{pmatrix} \hat{\Sigma}_k & 0 \\ 0 & \hat{\Sigma}_j \end{pmatrix} \begin{pmatrix} \hat{V}_k & \hat{V}_k^\perp \end{pmatrix}^T,$$

the best rank-k approximation of the SVD of B is found by setting $\hat{\Sigma}_j$ to zero, resulting in

$$B_k = ((U_k \quad Q_Y)\hat{U}_k)\hat{\Sigma}_k((V_k \quad Q_Z)\hat{V}_k)^T = U_B\Sigma_B V_B^T.$$

The new singular values and singular vectors computed from any of the updating steps reflect changes in the vector space due to the addition of terms and documents. SVD-updating, in one sense, is a dynamic IR model that can accurately reflect the impact that new or modified information can have on a current index of terms and documents.

8.2.3. Downdating. Using SVD-downdating, the LSI model can be modified to reflect the *removal* of terms and documents and any subsequent changes to term weights. Downdating can be useful for information filtering [24] (e.g., parental screening of Internet sites) and evaluating the importance of a term or document with respect to forming or breaking clusters of semantically related information. See [12] and [60] for more details on the effects of downdating and how it can be implemented.

8.3. Sparsity. The sparsity of a term-by-document matrix is a function of the word usage patterns and topic domain associated with the document collection. The more new terms each document brings to the global dictionary, the sparser is the matrix overall. The sample IR matrices studied in [5] are typically no more than 1% dense; that is, the ratio of nonzeros to the product of the row and column dimensions is barely 0.01. Experience has shown that these matrices typically lack any regular nonzero pattern, but some recent efforts in the use of both spectral (based on the eigendecomposition or SVD) and nonspectral (usually graph-theoretic) approaches to generate banded or *envelope* matrix forms are promising [11].

In order to compute the SVD of sparse term-by-document matrices, it is important to store and use only the nonzero elements of the matrix. Special matrix storage formats (e.g., Harwell–Boeing) have been developed for this purpose (see [3]). Special techniques for computing the SVD of a sparse matrix include iterative methods such as Arnoldi [37], Lanczos [34, 45], subspace iteration [47, 45], and trace minimization [51]. All of these methods reference the sparse matrix A only through matrix–vector multiplication operations, and all can be implemented in terms of the sparse storage formats.

Implementations of the aforementioned methods are available at www.netlib.org. These include software for Arnoldi-based methods (ARPACK) as discussed in [36] and [37] and implementations of Lanczos, subspace iteration, and trace minimization (SVDPACK (Fortran 77) [6] and SVDPACKC (ANSI C) [8]), as discussed in [5]. Simple descriptions of Lanczos-based methods with MATLAB examples are available in [3], and a good survey of public-domain software for Lanczos-type methods is available in [7]. Whereas most of the iterative methods mentioned thus far are serial in nature, an interesting asynchronous technique for computing several of the largest singular triplets of a sparse matrix on a network of workstations is described in [59].

For relatively small-order term-by-document matrices, it may be most convenient to ignore sparsity altogether and consider the matrix A as dense. One Fortran library including the SVD of dense matrices is LAPACK [1]. MATLAB also provides a dense SVD routine called by `[U,Sigma,V]=svd(A)` if A is stored as a dense matrix or by `[U,Sigma,V]=svd(full(A))` if A is stored as a sparse matrix. MATLAB (version 5.1) also provides a function to compute a few of the largest singular values and corresponding singular vectors of a sparse matrix. If the k largest singular values and corresponding left and right singular vectors are required, the MATLAB call

is `[Uk,Sigmak,Vk] = SVDS(A,k)`. The sparse SVD function `SVDS` is based on the Arnoldi methods described in [36]. Note that, for practical purposes, less expensive factorizations such as QR or ULV may suffice in place of the SVD [10].

Presently, no effort is made to preserve sparsity in the SVD of the sparse term-by-document matrices. Since the singular vector matrices are often dense, the storage requirements for U_k, Σ_k, and V_k can vastly exceed those of the original term-by-document matrix. For example, a sparse $5,526 \times 1,033$ term-by-document matrix A generated from the MEDLINE collection [41] of medical abstracts requires 0.4 Mbytes to store the original matrix, whereas the storage needed for the corresponding single precision matrices U_k,Σ_k,V_k is 2.6 Mbytes when $k = 100$.

The semidiscrete decomposition (or SDD) [32] provides one means of reducing the storage requirements of LSI. In SDD, only the three values $-1, 0, 1$ (represented by two bits each) are used to define the elements of U_k and V_k, and an integer programming problem is solved to produce the decomposition. Another possible way to remedy the problem of fill is to replace the singular vector matrix with a less accurate but more compact form. In particular, we can replace small elements of the matrices U_k and V_k with zeros and store the results in sparse formats. In the MEDLINE example, if we replace all entries of less than 0.0025 with zero we reduce U_k from 100% dense to 61% dense. It can be shown that the error in computing the cosines using the sparser approximation in place of U_k is equal to 0.0976, which may be acceptable in some circumstances.

9. Further Reading. In addition to the numerous LSI-related journal articles and technical reports cited in this paper, two recommended sources of background material on IR systems are the textbooks by Frakes and Baeza-Yates [25] and Kowalski [33]. Both of these books are used in undergraduate and graduate courses in IR, and both provide good references on the design and performance of IR systems. While the book by Frakes and Baeza-Yates [25] does provide some accompanying C software, Kowalski's [33] does not elaborate on the computational (or software) issues associated with automatic indexing. Foundational concepts in IR are covered in Salton's book [48]. Salton and McGill later published a more modern study of IR methods in [50].

Certainly, more data and information management tutorials and handbooks will be available in the near future, as the need for skills in information-based technologies continues to grow. We hope that our presentation of the more mathematical side of information modeling will spur new interest in computational mathematics and attract students and faculty to pursue interdisciplinary research in linear algebra and information science.

Acknowledgments. The authors thank Mark Anderson, Laura Mather, and Jim Martin for their reviews of a draft of this paper. The Website www.amazon.com was a source of material for the examples in this paper. The numerical results were computed using MATLAB.

REFERENCES

[1] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, AND D. SORENSEN, *LAPACK Users' Guide*, 2nd ed., SIAM, Philadelphia, PA, 1995.

[2] H. ANDREWS AND C. PATTERSON, *Outer product expansions and their uses in digital image processing*, Amer. Math. Monthly, 82 (1975), pp. 1–13.

[3] R. BARRETT, M. BERRY, T. F. CHAN, J. DEMMEL, J. DONATO, J. DONGARRA, V. EIJKHOUT, R. POZO, C. ROMINE, AND H. VAN DER VORST, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[4] D. Berg, *A Guide to the Oxford English Dictionary*, Oxford University Press, Oxford, UK, 1993.

[5] M. Berry, *Large scale singular value computations*, Internat. J. Supercomputer Applications, 6 (1992), pp. 13–49.

[6] M. Berry, *SVDPACK: A Fortran* 77 *Software Library for the Sparse Singular Value Decomposition*, Tech. Rep. CS–92–159, University of Tennessee, Knoxville, TN, June 1992.

[7] M. Berry, *Survey of public-domain Lanczos-based software*, in Proceedings of the Cornelius Lanczos Centenary Conference, J. Brown, M. Chu, D. Ellison, and R. Plemmons, eds., 1997, SIAM, Philadelphia, PA, pp. 332–334.

[8] M. Berry, T. Do, G. O'Brien, V. Krishna, and S. Varadhan, *SVDPACKC: Version* 1.0 *User's Guide*, Tech. Rep. CS–93–194, University of Tennessee, Knoxville, TN, October 1993.

[9] M. W. Berry, S. T. Dumais, and G. W. O'Brien, *Using linear algebra for intelligent information retrieval*, SIAM Rev., 37 (1995), pp. 573–595.

[10] M. Berry and R. Fierro, *Low-rank orthogonal decompositions for information retrieval applications*, Numer. Linear Algebra Appl., 3 (1996), pp. 301–328.

[11] M. Berry, B. Hendrickson, and P. Raghavan, *Sparse matrix reordering schemes for browsing hypertext*, in Lectures in Appl. Math. 32: The Mathematics of Numerical Analysis, J. Renegar, M. Shub, and S. Smale, eds., AMS, Providence, RI, 1996, pp. 99–123.

[12] M. Berry and D. Witter, *Intelligent information management using latent semantic indexing*, in Proceedings of Interface'97, Interface of North America Foundation, Fairfax Station, VA, 1997.

[13] K. Bharat and A. Broder, *Estimating the relative size and overlap of public web search engines*, in 7th International World Wide Web Conference, paper FP37, Elsevier Science, New York, 1998.

[14] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA, 1996.

[15] D. Bogard, ed., *The Bowker Annual Library and Book Trade Almanac*, 43rd ed., R.R. Bowker, New Providence, NJ, 1998.

[16] *Books in Print*, R.R. Bowker, New York, 1997/1998.

[17] R. Bording, A. Gertsztenkorn, L. Lines, J. Scales, and S. Treitel, *Applications of seismic travel-time tomography*, Geophys. J. R. Astr. Soc., 90 (1987), pp. 285–304.

[18] C. Buckley, G. Salton, J. Allan, and A. Singhanl, *Automatic query expansion using SMART: TREC* 3, in Overview of the Third Text REtrieval Conference, D. Harman, ed., National Institute of Standards and Technology Special Publication 500-226, NIST, Gaithersburg, MD, April 1995, pp. 69–80.

[19] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, *Indexing by latent semantic analysis*, J. American Society for Information Science, 41 (1990), pp. 391–407.

[20] S. Dumais, *Improving the retrieval of information from external sources*, Behavior Research Methods, Instruments, & Computers, 23 (1991), pp. 229–236.

[21] S. Dumais, *LSI meets TREC: A status report*, in The First Text REtrieval Conference, D. Harman, ed., National Institute of Standards and Technology Special Publication 500-207, NIST, Gaithersburg, MD, March 1993, pp. 137–152.

[22] S. Dumais, *Latent Semantic Indexing (LSI) and TREC*-2, in The Second Text REtrieval Conference, D. Harman, ed., National Institute of Standards and Technology Special Publication 500-215, NIST, Gaithersburg, MD, March 1994, pp. 105–116.

[23] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[24] P. Foltz and S. Dumais, *Personalized information delivery: An analysis of information filtering methods*, Comm. ACM, 35 (1992), pp. 51–60.

[25] W. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures & Algorithms*, Prentice–Hall, Englewood Cliffs, NJ, 1992.

[26] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[27] G. H. Golub, V. Klema, and G. W. Stewart, *Rank Degeneracy and Least Squares Problems*, Tech. Rep. TR-456, Department of Computer Science, University of Maryland, College Park, 1976.

[28] D. Harman, *Overview of the third text REtrieval conference (TREC-3)*, in Overview of the Third Text REtrieval Conference, D. Harman, ed., National Institute of Standards and Technology Special Publication 500-226, NIST, Gaithersburg, MD, April 1995, pp. 1–21.

[29] D. Harman and E. Voorhees, *Overview of the fifth text REtrieval conference (TREC-5)*, in Information Technology: The Fifth Text REtrieval Conference (TREC-5), D. Harman and E. Voorhees, eds., National Institute of Standards and Technology Special Publication 500-238, NIST, Gaithersburg, MD, November 1996, pp. 1–28.

[30] W. Jones and G. Furnas, *Pictures of relevance: A geometric analysis of similarity measures*, J. American Society for Information Science, 38 (1987), pp. 420–442.
[31] W. Kahan, *Conserving Confluence Curbs Ill-Conditioning*, Tech. Rep. 6, Computer Science Department, University of California, Berkeley, 1972.
[32] T. Kolda and D. O'Leary, *A semi-discrete matrix decomposition for latent semantic indexing in information retrieval*, ACM Trans. Inform. Systems, 16 (1998), pp. 322–346.
[33] G. Kowalski, *Information Retrieval Systems: Theory and Implementation*, Kluwer Academic Publishers, Boston, 1997.
[34] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255–282.
[35] S. Lawrence and C. Giles, *Searching the world wide web*, Science, 280 (1998), pp. 98–100.
[36] R. Lehoucq, *Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration*, Ph.D. Thesis, Rice University, Houston, TX, 1995.
[37] R. Lehoucq and D. Sorensen, *Deflation techniques for an implicitly restarted Arnoldi iteration*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 789–821.
[38] T. Letsche and M. Berry, *Large-scale information retrieval with latent semantic indexing*, Inform. Sci., 100 (1997), pp. 105–137.
[39] Library of Congress, http://lcweb.loc.gov/, April 24, 1998.
[40] J. Lovins, *Development of a stemming algorithm*, Mechanical Translation and Computational Linguistics, 11 (1968), pp. 22–31.
[41] MEDLINE, ftp://ftp.cs.cornell.edu/pub/smart/med/, May 27, 1998.
[42] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math., 11 (1960), pp. 50–59.
[43] C. Moler and D. Morrison, *Singular value analysis of cryptograms*, Amer. Math. Monthly, 90 (1983), pp. 78–87.
[44] G. O'Brien, *Information Management Tools for Updating an SVD-Encoded Indexing Scheme*, Master's Thesis, University of Tennessee, Knoxville, TN, 1994.
[45] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice–Hall, Englewood Cliffs, NJ, 1980.
[46] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*, 1st ed., Prentice–Hall, Englewood Cliffs, NJ, 1978.
[47] H. Rutishauser, *Simultaneous iteration method for symmetric matrices*, Numer. Math., 16 (1970), pp. 205–223.
[48] G. Salton, *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, 1968.
[49] G. Salton and C. Buckley, *Improving retrieval performance by relevance feedback*, J. American Society for Information Science, 41 (1990), pp. 288–297.
[50] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.
[51] A. H. Sameh and J. A. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM J. Numer. Anal., 19 (1982), pp. 1243–1259.
[52] J. Scales, P. Dochery, and A. Gerszternkorn, *Regularization of nonlinear inverse problems: Imaging the near-surface weathering layer*, Inverse Problems, 6 (1990), pp. 115–131.
[53] Search Engine Watch, http://www.searchenginewatch.com/, March 25, 1998.
[54] H. Simon and H. Zha, *On Updating Problems in Latent Semantic Indexing*, Tech. Rep. CSE-97-011, The Pennsylvania State University, University Park, PA, 1997.
[55] K. Sparck Jones, *A statistical interpretation of term specificity and its applications in retrieval*, J. Documentation, 28 (1972), pp. 11–21.
[56] G. W. Stewart, *Rank degeneracy*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 403–413.
[57] TIPSTER, http://www.tipster.org, May 28, 1998.
[58] *Ulrich's International Periodicals Directory*, R.R. Bowker, New York, 1998.
[59] S. Varadhan, M. Berry, and G. Golub, *Approximating dominant singular triplets of large sparse matrices via modified moments*, Numer. Algorithms, 13 (1996), pp. 123–152.
[60] D. Witter, *Downdating the Latent Semantic Indexing Model for Information Retrieval*, Master's Thesis, University of Tennessee, Knoxville, TN, 1997.