Folding-up: A Hybrid Method for Updating the Partial Singular Value Decomposition in Latent Semantic Indexing

by

Jane Elizabeth Bailey Tougas

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December, 2005

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Folding-up: A Hybrid Method for Updating the Partial Singular Value Decomposition in Latent Semantic Indexing**" by **Jane Elizabeth Bailey Tougas** in partial fulfillment of the requirements for the degree of **Master of Computer Science**.

Dated: December 6, 2005

Supervisor:	Raymond J. Spiteri
Readers:	Patrick. Keast
	Evangelos. E. Milios

Table of Contents

Abstract

The tremendous size of the Internet and modern databases has made efficient searching and information retrieval (IR) an area of utmost importance. Latent semantic indexing (LSI) is an IR method that represents a dataset as a term-document matrix. LSI uses a matrix factorization method known as the partial singular value decomposition (PSVD) to reduce noise in the data by projecting the term-document matrix into a lower-dimensional vector space. Calculating the PSVD of a large term-document matrix is computationally expensive. In a rapidly expanding environment, a term-document matrix is altered often as new documents and terms are added. Recomputing the PSVD of the term-document matrix each time these slight alterations occur can be prohibitively expensive.

Folding-in is one method of adding new documents or terms to a term-document matrix; updating the PSVD of the existing term-document matrix is another method. The folding-in method is computationally inexpensive, but it may cause deterioration in the accuracy of the PSVD. The PSVD-updating method is computationally more expensive than the folding-in method, but it better maintains the accuracy of the PSVD. This thesis introduces folding-up, a new method which combines folding-in and PSVD-updating. Folding-up offers a significant improvement in computation time when compared with either recomputing the PSVD, or PSVD-updating, while avoiding the degradation in the PSVD that can occur when the folding-in method is used on its own.

Acknowledgements

"Just as a house needs a foundation in order to stand firm, so does a person."
—Jacquie McTaggart

I extend my heartfelt appreciation to my family, who have been unflagging in their extraordinary patience, support, and encouragement. They are my foundation. I also offer special thanks to my supervisor Dr. Raymond J. Spiteri, and to Dr. Patrick Keast; without them I would never have considered this path. I would also like to express my appreciation to Dr. Shepherd for the use of his computers, and to Trevor Smith for inadvertently providing the incentive to finish. Thanks also to Chris Jordan, Philip O'Brien, John Mason, and Singer Wang for their friendship, advice, and encouragement, and to Lydia Hill, not only for her example, but also for her enthusiasm when it was most needed. This thesis could not have been completed without the funding of the Killam Foundation and the Natural Sciences and Engineering Research Council of Canada (NSERC) for whose support I am also very grateful. This thesis is dedicated to Christopher Tougas (August 9, 1963 – October 19, 2005), who lived life with spirit, strength, and good humour. His courage inspired all who knew him.

"I must finish what I've started, even if, inevitably,
what I finish turns out not to be what I began..." —Salman Rushdie

Chapter 1

Introduction

1.1 Motivation

In recent years, there has been a tremendous increase in the size of both the Internet and modern databases. With this growth comes a corresponding increase in the need for efficient information retrieval (IR) methods. Latent Semantic Indexing (LSI) is a method for the automatic indexing and retrieval of textual data [11]. LSI uses a mathematical approach known as the *vector-space model*, which relies heavily on techniques from numerical linear algebra. The vector-space model represents a document collection as a *term-document matrix* containing a column vector for each document in the text collection and a row vector for each semantically significant term [29]. Typically, terms that occur in only one document are not considered semantically significant; often, these words are simply spelling errors. Terms that occur in most of the documents in a text collection are also not considered semantically significant, because they are not useful in differentiating between documents. Words that occur in at least 80% of the documents are considered to be semantically insignificant *stopwords*; they are not included in the term-document matrix [1]. Common stopwords include words such as *as*, *or*, and *and*. A list of 571 such English stopwords [10] is used by the Cornell SMART system, one of the first vector-space information retrieval systems [29]. Such a list is not exhaustive however, and the use of a stopword list does not guarantee the removal of all semantically insignificant terms.

A term-document matrix $\mathbf{A}$ thus has t rows and d columns, where t is the number of semantically significant terms, or *index* terms, and d is the number of documents in the text collection. Each entry a_{ij} indicates the importance of term i relative to document j, where $1 \leq i \leq t$, and $1 \leq j \leq d$. The entries may simply be binary (1 if the term appears in the document and 0 otherwise), raw term frequencies (the number of times the term appears in the document), or more typically, weighted term frequencies [1, 28]. Two forms of weighting are commonly used in practice. These are *local weighting*, and *global weighting*. Local weighting indicates the importance of a term in a document; the frequency of the term in the document is used to calculate the local weight. Global weighting, on the other hand, indicates the importance of the term in the document collection as a whole; the frequency of a word across the entire document collection is used to calculate the global weight [3, 4]. If a word appears in relatively few documents it is considered to be a better index term than a word that appears in many documents, and so it will receive a correspondingly higher weight. Given a term-document matrix $\mathbf{A} \in \Re^{t \times d}$, a local weight l_{ij}, and a global weight g_j, the weight of index term i in document j (entry a_{ij} in the term document matrix) is given by

$$a_{ij} = l_{ij} \times g_j,$$

where $1 \leq i \leq t$, and $1 \leq j \leq d$. Because there may be considerable variance in the length of documents (and therefore of the frequency of terms within a document), term weights may be normalized [2]. With both local and global weighting, the higher the weight, the more important the term is considered to be. Typically, term weight formulas are constructed such that, for each term weight a_{ij}, $0 \leq a_{ij} \leq 1$. Term weighting is discussed in more detail in Chapter 4, with particular attention, in Section 4.1.1, to the term frequency $\times$ inverse document frequency ($tf \times idf$) weighting scheme.

The term-document matrix with t rows and d columns represents a t-dimensional space with d t-dimensional document vectors, where t is the number of semantically significant terms. Each document vector contains the coordinates of that document's location in the t-dimensional space. A user's search query is represented as a document (column) vector, a *pseudo-document*, using the same stopword removal and weighting scheme that have been applied to the document collection. The vectors of documents (and queries) with many terms in common are close together in the t-dimensional vector space, whereas the vectors of documents with few terms in common are far apart. The distance, or *similarity*, between each pair of vectors is typically measured using the cosine of the angle between them; this is known as the *cosine similarity measure* [1, 2]. Two vectors that are very close together have an angle with a large cosine value (close to 1); that is, they are almost parallel. Two vectors that are far apart have an angle with a small cosine value; that is, they are almost perpendicular.

Unfortunately even using the vector-space model, retrieving textual information in response to a search query is hampered by the fact that many words have similar meanings (they are *synonymous*). When a word that has a synonym is used in a search query, relevant documents containing that synonym, but not the specific word used in the query, may be overlooked. This is known as *recall failure.* A further complication arises from the fact that many words have more than one meaning (they are *polysemous*). When a polysemous word is used in a search query, irrelevant documents about the word's other meaning(s) may be retrieved. This is known as *precision failure.* The goal of efficient information retrieval is to have both high recall (all relevant documents are retrieved) and high precision (all retrieved documents are relevant). LSI assumes that, although there is an underlying semantic structure in the data, the term-document representation of the document collection includes *noise* due to factors such as synonymy and polysemy. LSI uses a matrix factorization method known as the *partial singular value decomposition* (PSVD) to reduce this noise, and estimate a more accurate representation of the data. By means of the PSVD, the data in the term-document matrix are projected into a lower-dimensional vector space. This lower-dimensional space is associated with concepts rather than specific index terms (i.e., the concepts that are spanned by both the terms and the documents are represented by the geometric relationships between the vectors in this lower-dimensional space) [2]. The more terms documents have in common, the more closely related they are considered to be, and the closer together their representative vectors will lie in the lower-dimensional vector space. Query vectors are also projected into the lower-dimensional space using the PSVD. Retrieving documents relevant to a particular query is based on the matching of concepts rather than the literal matching of index terms to query terms. When a query is processed, documents containing the index terms in the query are retrieved, but those documents that are closely related to the documents containing the query terms are also retrieved. Thus, LSI returns the documents that are most similar to a search query (those closest to the query in the vector space), even if the documents do not contain all (or any) of the terms contained in the query [1, 4]. Although research indicates that LSI is more successful in dealing with the problems caused by synonymy than those caused by polysemy [11], this does not detract from the importance of LSI in IR.

1.2 Statement of Problem

Although using the PSVD to project the term-document matrix into a lower-dimensional space has the benefit of removing noise from the data, it has the drawback of being computationally expensive. Even using numerical linear algebra methods, most of the computation time in LSI is spent in calculating the PSVD [3, 4]. In a dynamic environment, such as the Internet, the term-document matrix is changed often as new documents and terms are added. Given the potentially huge size of such term-document matrices, recomputing the PSVD of the matrix each time such changes occur can be prohibitively expensive. Traditionally, LSI uses a process known as *folding-in* to modify the PSVD if recomputing it would be too costly [11]. The folding-in of new documents or terms occurs based on the existing representation of the data (PSVD), and adds to, rather than modifies, the existing semantic structure [3, 4]. In the existing vector-space, new documents that are folded-in are placed at the centroid of their index terms, and new terms that are folded-in are placed at the centroid of the documents in which they are found [11]; details are discussed in Section 3.2.

Unfortunately, although the folding-in method, suggested by Deerwester et al. [11] and described by O'Brien [26], and Berry, Dumais, and O'Brien [4], is much faster than recomputing the PSVD, it may result in a significant degradation of retrieval performance [33]. A more recent and more accurate approach is to *update* the PSVD using updating algorithms. PSVD-updating algorithms for LSI were introduced by O'Brien in his Master's thesis [26] and published in Berry, Dumais, and O'Brien [4]. These methods for updating the PSVD when new documents or new terms are added to the LSI-database are a compromise between recomputing the PSVD and folding-in. Although these methods are much slower than folding-in, they are much faster than recomputing the PSVD, and unlike folding-in, they do not cause the significant degradation in the data representation that occurs with the folding-in methods. Details of these algorithms are discussed in Section 3.3.

Other PSVD-updating algorithms for use in LSI have been suggested by Zha and Simon [33]. Zha and Simon have shown that these methods give superior results when compared to the PSVD-updating algorithms of [4, 26]. In the absence of roundoff errors, the updating procedure of [33] produces the *exact* PSVD of the updated matrix. In practice roundoff errors do affect the results. Details of these algorithms are given in Section 3.4.

This thesis introduces *folding-up*, a new hybrid method that is a combination of folding-in and PSVD-updating. Folding-up is an attractive option because it offers a significant improvement in computation time when compared to either recomputing or PSVD-updating, and yet unlike the folding-in method, it results in little or no loss of accuracy. Although the folding-up experiments performed for this thesis use the PSVD-updating algorithms of Zha and Simon [33], the method can easily be adapted to use other PSVD-updating algorithms. Details of the folding-up method are given in Section 3.5.

1.3 Literature Survey

Although the focus of this thesis is specifically on updating the PSVD in LSI, the problem of updating the PSVD is not limited to the field of information retrieval. In numerical linear algebra, early work on incrementally updating the singular value decomposition (SVD) was carried out by Businger in 1970 [8] and by Bunch and Neilson in 1978 [7]; in 1994, Gu and Eisenstat introduced an algorithm for updating the SVD when a row or column is added to the original matrix [16]. The SVD is also used in image analysis, and an SVD-updating algorithm for use in this area was proposed by Chandrasekaran, Manjunath, Wang, Winkeler, and Zhang in 1997 [27]. In the area of signal processing, SVD-updating algorithms include those proposed by Moonen, van Dooren, and Vandewalle [23] and Ferzali and Proakis [13]. Techniques for parallelizing these algorithms are suggested by Moonen, van Dooren, and Vandewalle [24] and Sengupta, Cavallaro, and Aazhang [30]. Brand [5] discusses updating the PSVD in online recommender systems in which fragments of rows and columns need to be added in random order. Zha and Zhang [34] provide an analysis of matrices with a low-rank-plus-shift structure, motivated by computing and updating the PSVD. Recent work by Okša, Bečka, and Vajteršic [27] introduces an algorithm for the parallel computation of the SVD of matrices with a particular structure, specifically for use in the PSVD-updating algorithms proposed by Zha and Simon [33].

1.4 Overview

The remainder of this thesis proceeds as follows. Chapter 2 covers background information on matrix norms, the QR factorization, the SVD, the PSVD, and the comparison of terms and documents in LSI. The chapter concludes with an LSI example. Chapter 3 describes the methods of recomputing the PSVD, folding-in, PSVD-updating, and folding-up. Chapter 4 details term weighting and evaluation metrics, while Chapter 5 discusses the experiments performed and the experimental results. Finally, Chapter 6 gives conclusions and suggestions for future work.

Chapter 2

Background

2.1 Matrix Norms

A *matrix norm* is a scalar measure of the size of the elements of a matrix. In order for a function $\|\cdot\|$ to be a norm, it must be a real-valued function defined on the space of $t \times d$ matrices and meet the following three conditions:

1. $\|\mathbf{A}\| \geq 0$, with $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = \mathbf{0}$,
2. $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$,
3. $\|\alpha \mathbf{A}\| = |\alpha| \, \|\mathbf{A}\|$,

where $\mathbf{A} \in \Re^{t \times d}$, $\mathbf{B} \in \Re^{t \times d}$, and α is any scalar value. The infinity norm of a matrix, for example, is the maximum row sum of the matrix, defined for matrix $\mathbf{A} \in \Re^{t \times d}$ as

$$\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^{d} |a_{ij}|,$$

where $1 \leq i \leq t$, and $1 \leq j \leq d$. A matrix norm is *unitarily invariant* if, in addition to the above conditions, the following condition is also met:

$$\|\mathbf{Q}\mathbf{A}\| = \|\mathbf{A}\|,$$

where $\mathbf{Q} \in \Re^{t \times t}$ is a *unitary* matrix. A unitary matrix is a square, possibly complex matrix with orthonormal columns. Such a matrix, $\mathbf{Q}$, satisfies the property $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The *Frobenius norm* and the *2-norm* are both unitarily invariant matrix norms. For a matrix $\mathbf{A} \in \Re^{t \times d}$, the Frobenius norm is defined by

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{t} \sum_{j=1}^{d} |a_{ij}|^2},$$

and the 2-norm is defined by

$$\|\mathbf{A}\|_2 = \sigma_1,$$

where σ_i is the ith largest singular value of $\mathbf{A}$, and r is the rank of $\mathbf{A}$.

2.2 QR Factorization

A *QR factorization* is the decomposition of a matrix $\mathbf{A}$ such that $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is an *orthogonal matrix* and $\mathbf{R}$ is an upper-triangular matrix. An orthogonal matrix is a square matrix that has orthonormal columns; geometrically, this means that the columns are mutually perpendicular, and that each has a Euclidean length of one. When a *full* QR factorization is performed on a matrix $\mathbf{A} \in \Re^{t\times d}$, with $t \geq d$, then $\mathbf{Q} \in \Re^{t\times t}$ and $\mathbf{R} \in \Re^{t\times d}$ is upper-triangular, with its bottom $t - d$ rows containing zeros. Figure 2.1 gives a schematic representation of the full QR factorization with $t \geq d$, where the shaded area denotes zeros.

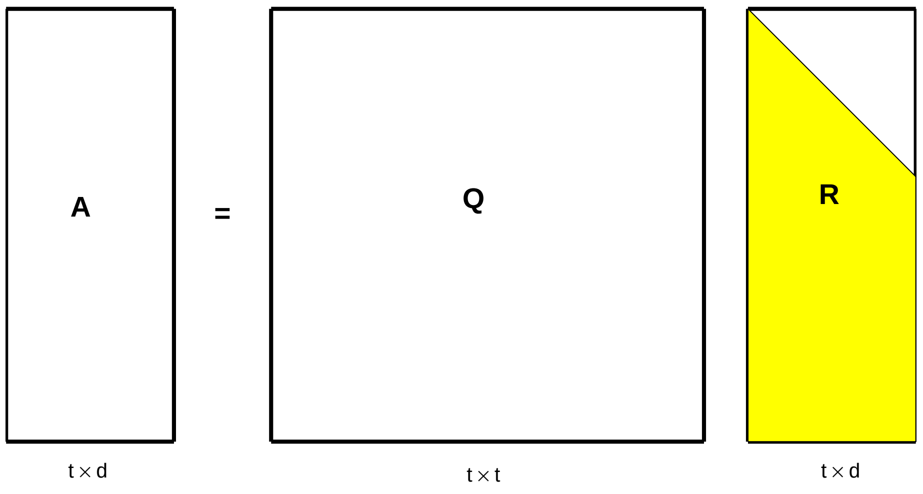

Figure 2.1: The Full QR Factorization of $\mathbf{A} \in \Re^{t\times d}, t \geq d$.

Figure 2.2 gives a schematic representation of the full QR factorization with $t < d$, where the shaded area denotes zeros.

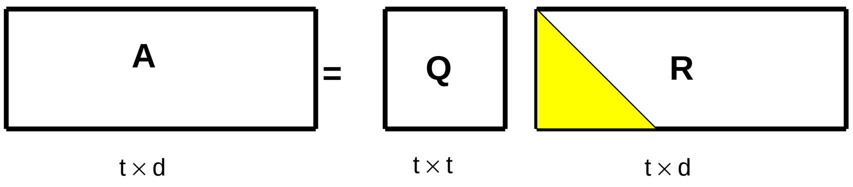

Figure 2.2: The Full QR Factorization of $\mathbf{A} \in \Re^{t\times d}, t < d$.

When a *reduced* QR factorization is performed under the same circumstances, such that $\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, then $\hat{\mathbf{Q}} \in \Re^{t \times d}$ is the first d columns of $\mathbf{Q}$, and $\hat{\mathbf{R}} \in \Re^{d \times d}$ is the first d rows of $\mathbf{R}$. In this case, none of the rows of $\hat{\mathbf{R}}$ are necessarily zero. Figure 2.3 gives a schematic representation of the reduced QR factorization, where the shaded area denotes zeros.

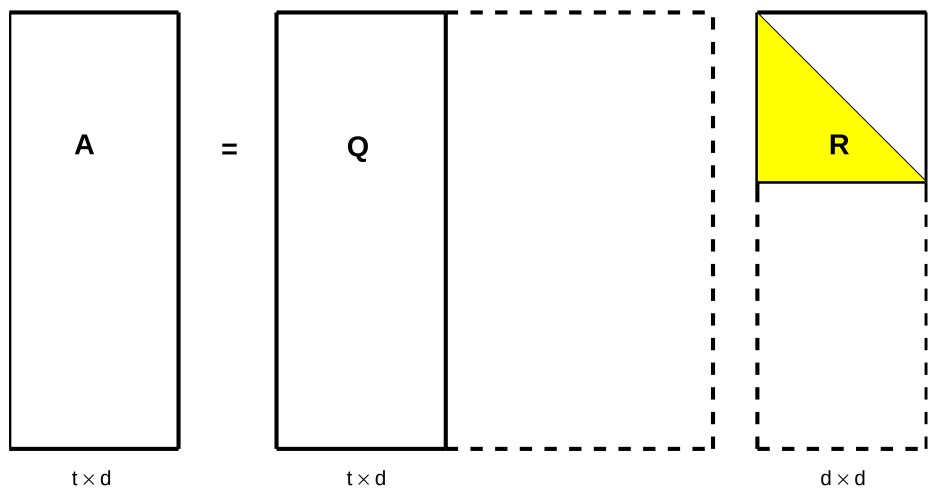

Figure 2.3: The Reduced QR Factorization of $\mathbf{A} \in \Re^{t \times d}, t > d$.

2.3 Singular Value Decomposition

The SVD is a matrix factorization that captures the most important characteristics of a matrix. The SVD is related to a number of mathematical techniques, such as spectral analysis and rank determination [11], and it is used for such purposes as dimensionality reduction, noise reduction, and clustering. Every matrix has an SVD, and the *singular values* $\{\sigma_j\}$ are always uniquely determined [31]. Given a real matrix $\mathbf{A}$ with t rows and d columns, its SVD has the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}, \tag{2.1}$$

where $\mathbf{U} \in \Re^{t \times t}$, $\mathbf{\Sigma} \in \Re^{t \times d}$, and $\mathbf{V} \in \Re^{d \times d}$. Matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices that contain the left and right *singular vectors* of $\mathbf{A}$ respectively. When $\mathbf{A}$ is a term-document matrix, the left singular vectors represent the term vectors, and the right singular vectors represent the document vectors (see Sections 2.5.1 and 2.5.2 for details). The matrix $\mathbf{\Sigma}$ has non-zero entries only on the diagonal, although not all diagonal entries are necessarily non-zero. These diagonal entries are the singular values of $\mathbf{A}$. The singular values are in non-increasing order, and are denoted by σ_j, for $j = 1, 2, \ldots, \min(t, d)$. The singular values are the non-negative square roots of the eigenvalues of the product $\mathbf{AA}^T$ [15]. The number of non-zero singular values is the rank, r, of the matrix. See Figure 2.4 for a schematic representation of the SVD.

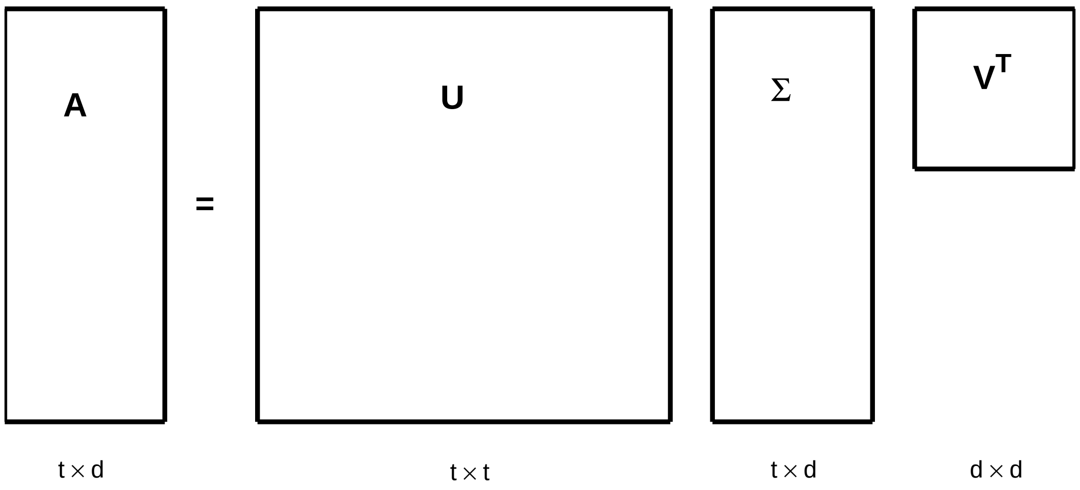

Figure 2.4: The SVD of $\mathbf{A} \in \Re^{t\times d}$.

2.4 Partial Singular Value Decomposition

Recall from Section 2.3 that the SVD of a matrix $\mathbf{A}$ is written as $\mathbf{A} = \mathbf{U\Sigma V}^T$. An alternative way to represent the SVD of a matrix is as the sum of r rank-one matrices

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T, \tag{2.2}$$

where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the jth columns of matrices $\mathbf{U}$ and $\mathbf{V}$, respectively. This representation of the SVD in Equation (2.2) allows the formation of lower-rank approximations of $\mathbf{A}$. Replacing r in this equation by any ν with $0 \leq \nu < r$ gives

$$\mathbf{A} \approx \sum_{j=1}^{\nu} \sigma_j \mathbf{u}_j \mathbf{v}_j^T .$$

Taking the first ν columns of $\mathbf{U}$ and $\mathbf{V}$ and the leading $\nu \times \nu$ submatrix of $\mathbf{\Sigma}$ gives a rank-ν approximation of $\mathbf{A}$; i.e., $\mathbf{A}_1$ is a rank-one approximation of $\mathbf{A}$, $\mathbf{A}_2$ is a rank-two approximation of $\mathbf{A}$, and so on. This can be expressed as

$$\mathbf{A}_\nu = \mathbf{U}_\nu \mathbf{\Sigma}_\nu \mathbf{V}_\nu^T. \tag{2.3}$$

In the Frobenius-norm

$$\|\mathbf{A} - \mathbf{A}_\nu\|_F^2 = \min_{\substack{\text{rank}(\mathbf{B}) \leq \nu \\ \mathbf{B} \in \Re^{t \times d}}} \|\mathbf{A} - \mathbf{B}\|_F^2 = \sigma_{\nu+1}^2 + \cdots + \sigma_r^2.$$

See [32] for a proof. This means that the rank-ν approximation given in Equation (2.3) is optimal in the sense that for a given rank, k, where $0 \leq k < r$, there is no matrix of rank at most k that is closer to $\mathbf{A}$, as measured in the Frobenius-norm [15]. However, this is true not only for the Frobenius norm, but for *any* unitarily invariant norm [4, 22]. In the 2-norm, for example,

$$\|\mathbf{A} - \mathbf{A}_\nu\|_2 = \min_{\substack{\text{rank}(\mathbf{B}) \leq \nu \\ \mathbf{B} \in \Re^{t \times d}}} \|\mathbf{A} - \mathbf{B}\|_2 = \sigma_{\nu+1}.$$

Let matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ be the first k columns of $\mathbf{U}$ and $\mathbf{V}$ respectively, and let matrix $\mathbf{\Sigma}_k$ be the leading submatrix of $\mathbf{\Sigma}$ with k rows and k columns. Then $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ is the optimal approximation (of rank at most k) of $\mathbf{A}$. This is the partial SVD (PSVD) of $\mathbf{A}$; see Figure 2.5 for a schematic representation of the PSVD. This approximation can be used to reduce the dimension of the term-document matrix, while eliciting the underlying structure of the data. In LSI, the effect of this dimensional reduction on the data is a muting of the noise in the data and an enhancing of the latent patterns that indicate semantically similar terms. This means that $\mathbf{A}_k$ can actually be a better representation of the data than the original term-document matrix. The optimal number of dimensions k (singular values and corresponding singular vectors) to keep in the reduced term-document matrix is an open question, and it is database dependent [2]. Deciding how many dimensions to use is based on empirical testing and, in the case of very large datasets, on the feasibility of computation [2, 21, 25]. It is important to note that, in general, having more dimensions does not necessarily provide better retrieval performance [11]. Typically, as the number of dimensions is increased, retrieval performance increases to a certain point, and then decreases as the number of dimensions is further increased [11, 4]. Experiments using databases containing between 1000 and 2000 documents have found that using between 50 and 100 dimensions is optimal [11]. For very large databases, containing hundreds of thousands of documents, the number of dimensions used is typically between 100 and 300 [21]. This tremendous dimensional reduction, given the potentially huge size of such term-document matrices, demonstrates the power of the PSVD as a method of data compression.

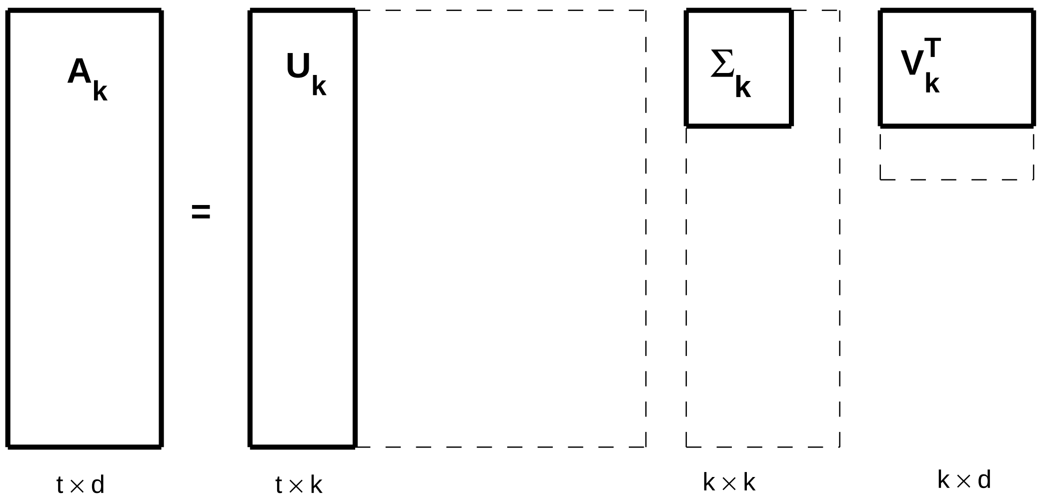

Figure 2.5: The PSVD of $\mathbf{A} \in \Re^{t \times d}$.

2.4.1 Computing the Partial Singular Value Decomposition

A term-document matrix is very sparse, containing relatively few non-zero entries and many zero entries, because any document typically contains only a small percentage of all the index terms [2]. When computing the PSVD of a term-document matrix $\mathbf{A}$, it makes sense to take advantage of this sparsity. In this thesis, all PSVD calculations are performed using the *Matlab* sparse matrix function `svds`, which returns a user-specified number k of the largest singular values of a matrix and, if requested, the corresponding left and right singular vectors. The function `svds` uses the function `eigs` to compute the k largest magnitude eigenvalues and corresponding eigenvectors of the following matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_t & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0}_d \end{bmatrix},$$

where $\mathbf{B} \in \Re^{(t+d)\times(t+d)}$, and $\mathbf{0}_t \in \Re^{t\times t}$ and $\mathbf{0}_d \in \Re^{d\times d}$ are zero matrices. $\mathbf{B}$ is a symmetric matrix. The positive eigenvalues of $\mathbf{B}$ are the singular values of $\mathbf{A}$, the first k eigenvectors of $\mathbf{B}$ correspond to the left singular vectors of $\mathbf{A}$, and the last k eigenvectors of $\mathbf{B}$ correspond to the right singular vectors of $\mathbf{A}$. The algorithm used by the *Matlab* function `eigs` to compute the k largest eigenvalues and corresponding eigenvectors is an implicitly restarted Arnoldi-Lanczos iterative method, implemented in the linear algebra library ARPACK. ARPACK is a collection of subroutines designed to solve large scale eigenvalue problems. The Arnoldi-Lanczos iterative method takes advantage of the existing sparsity of the matrix. Details of the algorithm can be found in the ARPACK Users' Guide [20].

2.5 Comparing Terms and Documents

In order to compare terms and/or documents, it is first helpful to define the *inner product* or *dot product* of two vectors. The dot product is a binary operation that takes two vectors and returns a scalar value. The dot product of two vectors $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$ is expressed as

$$\mathbf{x} \cdot \mathbf{y} = x_1y_1 + x_2y_2 + \ldots + x_ny_n = \sum_{i=1}^{n} x_iy_i = \mathbf{x}^T\mathbf{y}.$$

The dot product can be used to express the cosine of the angle between two vectors $\mathbf{x}$ and $\mathbf{y}$:

$$\cos\alpha = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2},$$

where $\|\cdot\|_2$ is the vector 2-norm. As discussed in Section 1.1, the cosine of the angle between a query vector and a document vector is used to measure the similarity between the two vectors; this is the cosine similarity measure.

2.5.1 Comparing Terms

To compare two terms in a term-document matrix $\mathbf{A}$, recall from Section 2.4 that the rank-k approximation of $\mathbf{A}$ is expressed as $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$. The dot product of the rows of $\mathbf{A}_k$ indicates the similarity of the occurrence pattern of the two terms represented by the rows, across all the documents (columns). This can be expressed as

$$\begin{aligned}
\mathbf{A}_k\mathbf{A}_k^T &= \left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right)\left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right)^T \\
&= \left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right)\left(\mathbf{V}_k\mathbf{\Sigma}_k\mathbf{U}_k^T\right) \\
&= \mathbf{U}_k\mathbf{\Sigma}_k\left(\mathbf{V}_k^T\mathbf{V}_k\right)\mathbf{\Sigma}_k\mathbf{U}_k^T \\
&= \mathbf{U}_k\mathbf{\Sigma}_k^2\mathbf{U}_k^T.
\end{aligned}$$

Thus, the measure of the comparison between the term represented by the ith row of $\mathbf{A}$ and that represented by the jth row of $\mathbf{A}$, is the element in row i, column j of $\mathbf{A}_k\mathbf{A}_k^T = \mathbf{U}_k\mathbf{\Sigma}_k^2\mathbf{U}_k^T$. The rows of $\mathbf{U}_k\mathbf{\Sigma}_k$ can be considered to be the coordinates for the terms [11]. For this reason, the matrix $\mathbf{U}_k$, containing the k left singular vectors of $\mathbf{A}$, is known as the term vector matrix.

2.5.2 Comparing Documents

To compare two documents in a term-document matrix $\mathbf{A}$, the dot product of the columns of $\mathbf{A}_k$ is taken. This product indicates how similar the pattern of terms is between the two documents. This can be expressed as

$$\begin{aligned}
\mathbf{A}_k^T\mathbf{A}_k &= \left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right)^T\left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right) \\
&= \left(\mathbf{V}_k\mathbf{\Sigma}_k\mathbf{U}_k^T\right)\left(\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T\right) \\
&= \mathbf{V}_k\mathbf{\Sigma}_k\left(\mathbf{U}_k^T\mathbf{U}_k\right)\mathbf{\Sigma}_k\mathbf{V}_k^T \\
&= \mathbf{V}_k\mathbf{\Sigma}_k^2\mathbf{V}_k^T.
\end{aligned}$$

Thus, the measure of the comparison between the document represented by the ith column of $\mathbf{A}$ and that represented by the jth column of $\mathbf{A}$, is the element in row i, column j of $\mathbf{A}_k^T\mathbf{A}_k = \mathbf{V}_k\mathbf{\Sigma}_k^2\mathbf{V}_k^T$. The rows of $\mathbf{V}_k\mathbf{\Sigma}_k$ can be considered the coordinates for the documents [11]. For this reason, the matrix $\mathbf{V}_k$, containing the k right singular vectors of $\mathbf{A}$, is known as the document vector matrix.

2.5.3 Comparing a Term and a Document

To compare a term i and a document j in a term-document matrix $\mathbf{A}$, the value of interest is that found in row i, column j of $\mathbf{A}$ (i.e., element a_{ij}). Recalling that the rank-k approximation of $\mathbf{A}$ is expressed as $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$, the dot product is taken between the ith row of $\mathbf{U}_k\mathbf{\Sigma}_k^{1/2}$ and the jth row of $\mathbf{\Sigma}_k^{1/2}\mathbf{V}_k^T$. Thus, the comparison of a term and a document uses coordinates from both $\mathbf{U}_k\mathbf{\Sigma}_k^{1/2}$ and $\mathbf{\Sigma}_k^{1/2}\mathbf{V}_k^T$ [11].

2.6 LSI Example

This example steps through the process of forming a small term-document matrix and projecting it into a 2-dimensional vector space using the PSVD.

Suppose that the document collection consists of a small number of book titles, given in Table 2.1. The semantically significant index terms are those terms not considered stopwords and that appear more than once in the document collection. These terms are underlined in Table 2.1.

ID	Document Title
D1	Going to the Net: A Girl's Guide to Cyberspace
D2	Internet for Kids: A Beginner's Guide to Surfing the Net
D3	Naked in Cyberspace: How to Find Personal Information Online
D4	Learning the Internet for Kids: A Voyage of Internet Treasures
D5	Internet Safety: Surfing the Net with Kids
D6	Online Kids: A Young Surfer's Guide to Cyberspace
D7	Feline Online: What Happens when a Smart Cat Surfs the Internet
D8	The Big Deep: Classic Big Wave Surfing
D9	Surf Science: An Introduction to Waves for Surfing
D10	Surf Riders: In Search of the Perfect Wave
D11	The Next Wave: The World of Surfing
D12	Big Wave: Stories of Riding the World's Wildest Water
D13	Good Things Love Water: A Collection of Surf Stories
D14	Surfing: The Big Wave

Table 2.1: Document IDs and Documents for Example 2.6.

Note that in processing the text, punctuation is removed, and a process known as *stemming* is performed to reduce words to their root form. Thus, for example, the words *surfing*, *surfer's*, and *surfs* are all stemmed to the root *surf*. The theory behind stemming is that by reducing words to a common concept, retrieval performance is enhanced. This is countered by the possibility that stemming may relate nonrelevant terms, and therefore nonrelevant documents may be retrieved (increasing precision failure). Reducing both the words *divide* and *divine* to the root *div* could cause this precision problem. Perhaps the greatest advantage to stemming, and the main reason for its use, is that it greatly reduces the number of index terms, and thus the size of the term-document matrix. For example, without stemming, the documents from Table 2.1 would generate 19 index terms, whereas with stemming, there are only 13 index terms. Thus, the term-document matrix is of size 13×14, rather than size 19×14. When the number of documents is very large, the saving in the storage needed for the term-document matrix becomes significant when stemming is performed. There are a number of automatic stemming algorithms; the most popular is Porter's algorithm [1].

Table 2.2 lists the stemmed index terms for the documents given in Table 2.1. Note that for the term *stories*, stemming has replaced the *ies* ending with *y*, giving the index term *story*.

Term ID	Stemmed Term	Term ID	Stemmed Term	Term ID	Stemmed Term
T1	net	T6	surf	T11	world
T2	guide	T7	online	T12	water
T3	cyberspace	T8	big	T13	story
T4	internet	T9	wave		
T5	kid	T10	ride		

Table 2.2: Term IDs and Stemmed Terms for Example 2.6.

The data in Tables 2.1 and 2.2 allow the formation of the term-document matrix in Table 2.3. For simplicity, global weighting is ignored in this example, and the local weight function is defined as the frequency of an index term in a document. The documents are similar in size, so normalization of the weights is not necessary. Note that the matrix is very sparse, containing few non-zero entries and many zero entries. This is typical of term-document matrices because any document is likely to contain only a small percentage of all the index terms.

ID	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
T1	1	1	0	0	1	0	0	0	0	0	0	0	0	0
T2	1	1	0	0	0	1	0	0	0	0	0	0	0	0
T3	1	0	1	0	0	1	0	0	0	0	0	0	0	0
T4	0	1	0	2	1	0	1	0	0	0	0	0	0	0
T5	0	1	0	1	1	1	0	0	0	0	0	0	0	0
T6	0	1	0	0	1	1	1	1	2	1	1	0	1	1
T7	0	0	1	0	0	1	1	0	0	0	0	0	0	0
T8	0	0	0	0	0	0	0	2	0	0	0	1	0	1
T9	0	0	0	0	0	0	0	1	1	1	1	1	0	1
T10	0	0	0	0	0	0	0	0	0	1	0	1	0	0
T11	0	0	0	0	0	0	0	0	0	0	1	1	0	0
T12	0	0	0	0	0	0	0	0	0	0	0	1	1	0
T13	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 2.3: Term-document matrix **A** for Example 2.6.

Let **A** be the 13×14 term-document matrix from Table 2.3. Having formed this matrix, the next step is to project it into a lower dimension using the PSVD. For convenience in plotting, let $k = 2$, where k is the number of dimensions. Recalling Equation (2.3), the projection is expressed as

$$\mathbf{A} \approx \mathbf{A}_2 = \mathbf{U}_2 \mathbf{\Sigma}_2 \mathbf{V}_2^T,$$

where $\mathbf{A}_2 \in \Re^{13\times14}$, $\mathbf{U}_2 \in \Re^{13\times2}$, $\mathbf{\Sigma}_2 \in \Re^{2\times2}$, and $\mathbf{V}_2^T \in \Re^{2\times14}$. Performing this calculation in *Matlab* gives the following results (where only 2 decimal places of accuracy are displayed).

$$
\begin{aligned}
\mathbf{A}_2 &= \mathbf{U}_2\mathbf{\Sigma_2}\mathbf{V}_2^T \\
&= \begin{bmatrix} -0.156 & 0.27 \\ -0.15 & 0.25 \\ -0.09 & 0.16 \\ -0.26 & 0.49 \\ -0.24 & 0.41 \\ -0.73 & -0.01 \\ -0.13 & 0.16 \\ -0.27 & -0.38 \\ -0.40 & -0.41 \\ -0.11 & -0.16 \\ -0.11 & -0.16 \\ -0.09 & -0.14 \\ -0.09 & -0.14 \end{bmatrix}
\begin{bmatrix} 4.62 & 0 \\ 0 & 3.48 \end{bmatrix}
\begin{bmatrix} -0.09 & 0.20 \\ -0.33 & 0.40 \\ -0.05 & 0.09 \\ -0.16 & 0.40 \\ -0.30 & 0.33 \\ -0.29 & 0.28 \\ -0.24 & 0.18 \\ -0.36 & -0.34 \\ -0.40 & -0.13 \\ -0.27 & -0.17 \\ -0.27 & -0.17 \\ -0.23 & -0.40 \\ -0.20 & -0.08 \\ -0.30 & -0.23 \end{bmatrix}^T .
\end{aligned}
$$

This rank-2 PSVD of the term-document matrix $\mathbf{A}$ can be used to plot the terms and documents in two dimensions. Given a standard x-axis and y-axis, the terms are plotted by scaling the first column of $\mathbf{U}_2$ by the first singular value to get the term x-coordinates, and by scaling the second column of $\mathbf{U}_2$ by the second singular value to get the term y-coordinates. Similarly, the documents are plotted by scaling the first column of matrix $\mathbf{V}_2$ by the first singular value to get the document x-coordinates, and by scaling the second column of $\mathbf{V}_2$ by the second singular value to get the document y-coordinates. Figure 2.6 gives the two-dimensional plot of the documents and terms from Tables 2.1 and 2.2.

Note that the plot shows two distinct clusters of terms and documents, corresponding to the two distinct topics (surfing the Internet, and surfing waves) in the documents. It is interesting to note that in two cases, two terms map to the same point (water and story, ride and world), and in one case, two documents map to the same point (D10 and D11). This is a function of the dimensionality reduction that

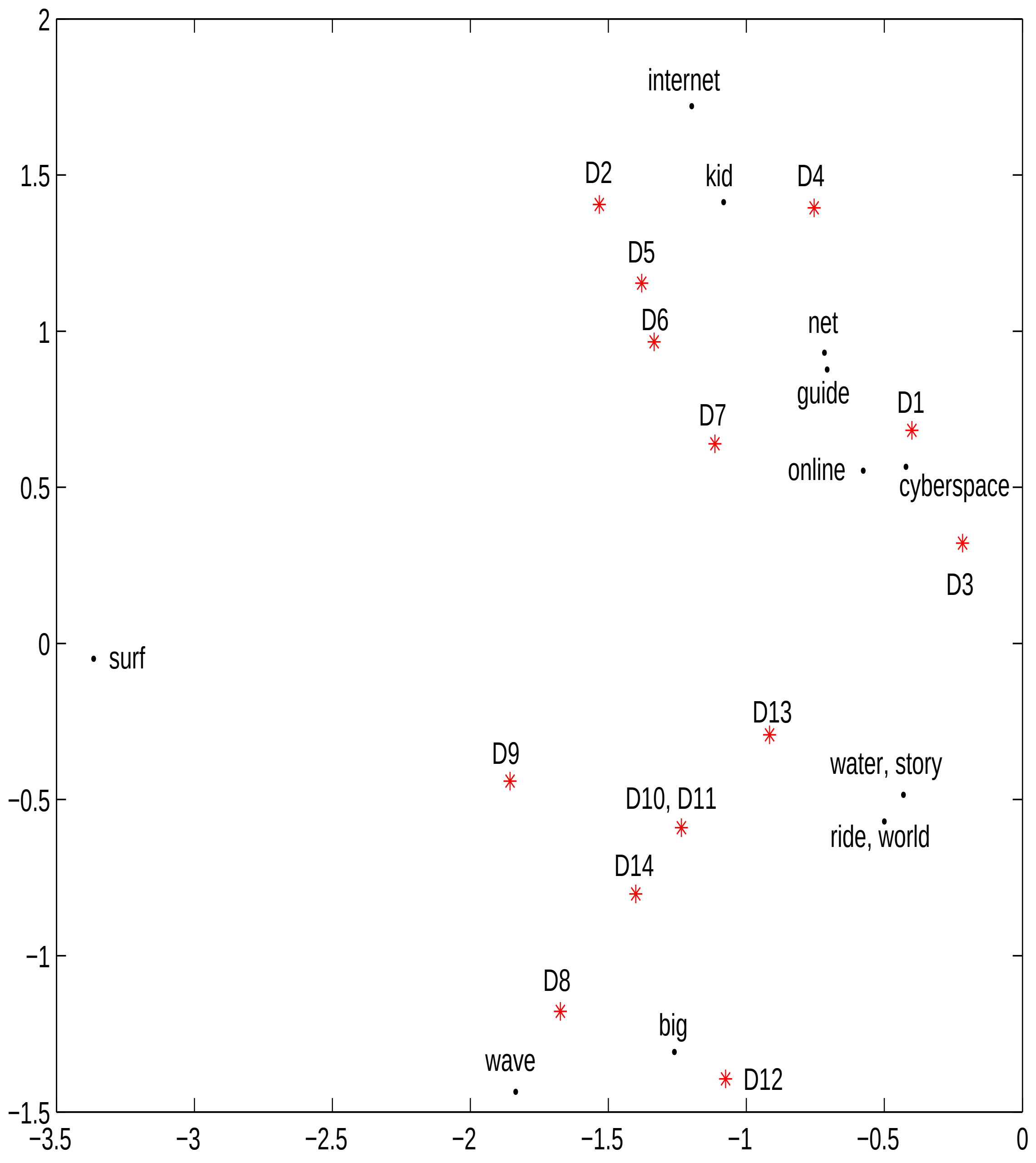

Figure 2.6: Two-dimensional plot of terms and documents for Example 2.6.

removes noise from the data [11].

As discussed in Section 1.1, search queries are treated as pseudo-documents, and are projected into the same lower-dimensional space as the documents, using the PSVD. Suppose, for the documents listed in Table 2.1, the search query *Information for kids about surfing the Internet* is posed. Removing stopwords and non-index terms and performing stemming reduces this query to *kid surf internet*. As a pseudo-document vector $\mathbf{q}$, this is expressed as $\mathbf{q} = [0,0,0,1,1,1,0,0,0,0,0,0,0]^T$.

To project the query $\mathbf{q}$ vector into the two-dimensional document space of Figure 2.6, giving the reduced dimensional query $\mathbf{q}_2$, the following multiplication is performed:

$$\begin{aligned}
\mathbf{q} \approx \mathbf{q}_2 &= \mathbf{q}^T \mathbf{U}_2 \boldsymbol{\Sigma}_2^{-1} \\
&= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} -0.156 & 0.27 \\ -0.15 & 0.25 \\ -0.09 & 0.16 \\ -0.26 & 0.49 \\ -0.24 & 0.41 \\ -0.73 & -0.01 \\ -0.13 & 0.16 \\ -0.27 & -0.38 \\ -0.40 & -0.41 \\ -0.11 & -0.16 \\ -0.11 & -0.16 \\ -0.09 & -0.14 \\ -0.09 & -0.14 \end{bmatrix} \begin{bmatrix} 4.62 & 0 \\ 0 & 3.48 \end{bmatrix}^{-1} \\
&= \begin{bmatrix} -0.27 & 0.25 \end{bmatrix}.
\end{aligned}$$

This projected query can be thought of as a probe into the document vector space [2]. The similarity of the query to the documents must be evaluated. As discussed in Section 1.1, the cosine similarity measure is used. The documents are ranked according to the value of the cosine of the angle between each document and the query. Using this ranked list, the top n documents (those closest to the query in the vector space) can be returned to the user as relevant to the query, or a *cosine threshold* can be set in which all documents above the given threshold are returned to the user as relevant. Let the angle between the query vector and a document vector be θ. The shaded area of Figure 2.7 represents the area in which document vectors which satisfy $\cos(\theta) \geq .90$ are found; this is the area spanning approximately 25.84° on either side of the query

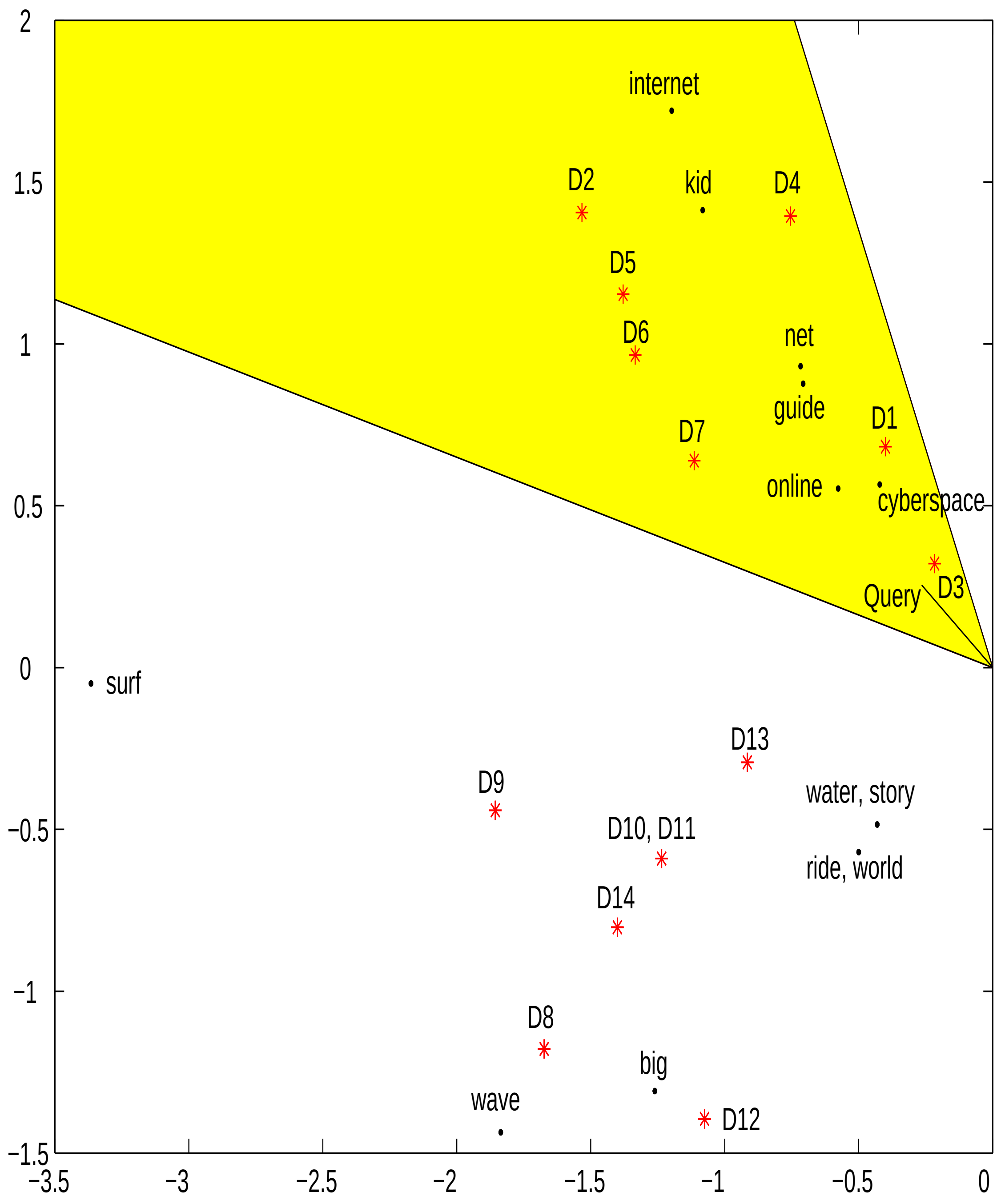

Figure 2.7: Two-dimensional plot of the terms and the documents from Example 2.6 and the query *Information for kids about surfing the Internet.*

vector (the line from the origin to the query). Note that all the documents related to surfing the Internet are in this area, whereas none of the documents related to surfing waves are in this area.

Chapter 3

Updating Options

As discussed in Chapter 1, calculating the PSVD of a matrix is a computationally intensive procedure that takes up most of the processing time in LSI [3, 4]. With a dynamic medium, such as the Internet, the PSVD needs to be modified often as new documents and terms are added to the existing LSI database. This chapter discusses options for incorporating changes to an existing term-document matrix in LSI. Section 3.1 describes recomputing the PSVD of the new term-document matrix from scratch, Section 3.2 describes the traditional method of folding-in new documents and terms, Section 3.3 details the PSVD-updating algorithms proposed by O'Brien [26] and Berry, Dumais, and O'Brien [4], and Section 3.4 covers the PSVD-updating algorithms proposed by Zha and Simon [33]. Finally, Section 3.5 introduces a new hybrid PSVD-updating method, folding-up.

Sections 3.1–3.4 include brief examples in which four documents and one term are added to the documents in Table 2.1 and the terms in Table 2.2 of the LSI example in Section 2.6. The documents to be added in each case are listed in Table 3.1. As in Table 2.1, the semantically significant index terms are those terms that are not to be considered stopwords and that appear more than once in the document collection. These terms are underlined in Table 3.1. The original documents have already been parsed, and the index terms selected as those that appeared more than once in these documents; therefore in this case, the only terms that are added when new documents are introduced are those terms that are considered semantically significant (i.e., they are not stopwords) and that appear more than once in the new documents. In the examples in this chapter, the term *web*, which appears twice in the new documents, is added to the existing term-document matrices.

ID	Document Title
D15	The Net: Effective Email, Web Surfing and Assessing Information
D16	The Complete Idiot's Guide to Surfing the Internet with WebTV
D17	Learning how to Surf the Web
D18	Kids Online: Protecting your Children in Cyberspace

Table 3.1: Additional documents to be appended to those from Table 2.1.

3.1 Recomputing the PSVD

Given unlimited time and memory, the ideal way of incorporating new documents and terms into an LSI database is to recompute the PSVD; i.e., compute the PSVD of the matrix

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{D} \\ \mathbf{0} & \mathbf{T} \end{bmatrix}, \tag{3.1}$$

where $\mathbf{A} \in \Re^{t\times d}$ is the original term-document matrix, $\mathbf{D} \in \Re^{t\times p}$ is the term-document matrix containing p new documents, and $\mathbf{T} \in \Re^{q\times p}$ is the term-document matrix containing the q new terms. It is possible, and probable, that due to memory constraints, the original term-document matrix $\mathbf{A}$ has not been stored in memory and is no longer available for use in computing $\hat{\mathbf{A}}$. In this case, Equation (3.1) is modified such that the term-document matrix $\mathbf{A}$ is replaced by the reduced-rank matrix $\mathbf{A_k} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$, where k is the rank (number of dimensions) of $\mathbf{A}_k$:

$$\hat{\mathbf{A}} \approx \begin{bmatrix} \mathbf{A}_k & \mathbf{D} \\ \mathbf{0} & \mathbf{T} \end{bmatrix}. \tag{3.2}$$

Although recomputing the PSVD of the new term-document matrix each time changes are made to the document collection is the most accurate method of incorporating change in the term-document representation of the dataset, it is also the most expensive method. Alternatives to this approach are discussed in Sections 3.2–3.5.

3.1.1 Recomputing Example

Suppose that the documents from Tables 2.1 and 3.1 are combined in order to create a new term-document matrix $\hat{\mathbf{A}}$. Whereas the original term-document matrix from Section 2.6 contains 13 terms and 14 documents, this new term-document matrix $\hat{\mathbf{A}}$ contains 14 terms (the term *web* is being added) and 18 documents. As with the example in Section 2.6, the new term-document matrix is projected into two dimensional space, for ease of visualizing. Thus, let $k = 2$, where k is the number of dimensions. Recalling Equation (2.3), the projection is expressed as

$$\hat{\mathbf{A}} \approx \hat{\mathbf{A}}_2 = \hat{\mathbf{U}}_2 \hat{\mathbf{\Sigma}}_2 \hat{\mathbf{V}}_2^T,$$

where $\hat{\mathbf{A}}_2 \in \Re^{14\times18}$, $\hat{\mathbf{U}}_2 \in \Re^{14\times2}$, $\hat{\mathbf{\Sigma}}_2 \in \Re^{2\times2}$, and $\hat{\mathbf{V}}_2^T \in \Re^{2\times18}$. See Figure 3.1 for the two-dimensional plot of the documents and terms using this procedure. Compare Figure 3.1 to Figure 2.6, and note that the positions of the documents and terms that were in the original term-document matrix have shifted in response to the new documents and term. Despite this shift, note that as in Figure 2.6, the terms *water* and *story* still map to the same point, as do the terms *ride* and *world*, and the documents D10 and D11. The two distinct clusters of documents are also maintained. The new documents, which are Internet related, are clustered with documents D1–D7 about surfing the Internet, rather than with documents D8–D14, which are about surfing waves.

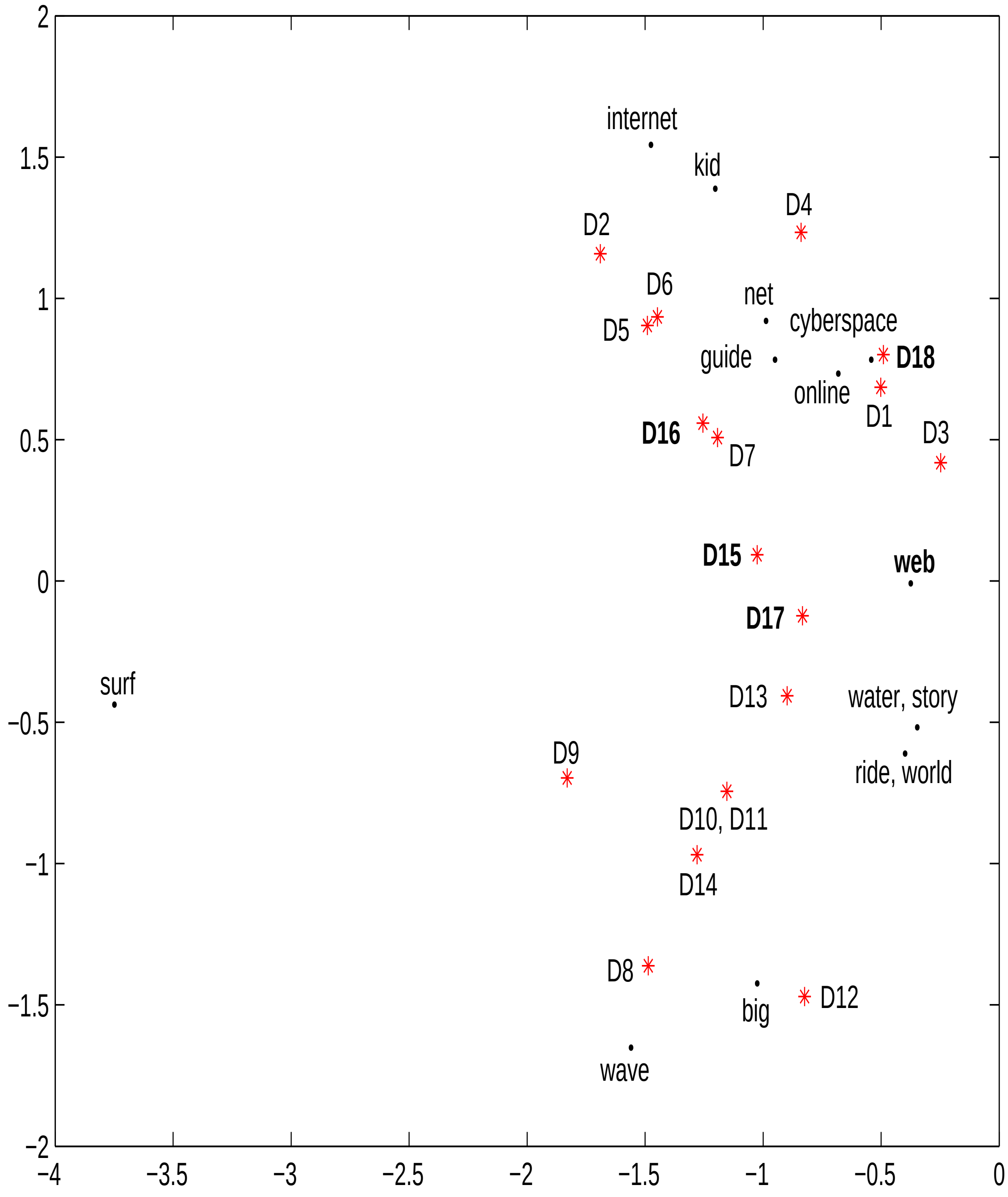

Figure 3.1: Two-dimensional plot of 14 terms and 18 documents using a PSVD that has been recomputed from scratch.

3.2 Folding-in

Because recomputing the PSVD when new documents and terms are added to an existing LSI database is very expensive, the method of folding-in new documents and terms is often used [11]. The folding-in method is computationally inexpensive compared to recomputing the PSVD. Unfortunately, because the folding-in method bases the new representation of the data on the existing latent semantic structure, its use can cause deterioration in the numerical representation of the dataset by the PSVD.

Recall from Equation (2.3) that the PSVD of the term-document matrix $\mathbf{A}$ is $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$, where k is the number of dimensions. Also recall, from Section 2.3 that the columns of matrix $\mathbf{U}_k$ are mutually orthogonal, as are the columns of matrix $\mathbf{V}_k$. Folding-in p documents appends p rows to the bottom of $\mathbf{V}_k$, giving $\hat{\mathbf{V}}_k$, and folding-in q terms appends q rows to the bottom of $\mathbf{U}_k$, giving $\hat{\mathbf{U}}_k$. This process corrupts the orthogonality of the columns of $\hat{\mathbf{U}}_k$ and $\hat{\mathbf{V}}_k$, potentially leading to a misrepresentation of the dataset.

3.2.1 Adding Documents

As before, let $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t\times k}, \mathbf{\Sigma}_k \in \Re^{k\times k}$, and $\mathbf{V}_k \in \Re^{d\times k}$. Let $\mathbf{D} \in \Re^{t\times p}$ be the term-document matrix containing the document vectors to be appended to $\mathbf{A}$, where p is the number of new documents.

Using the PSVD, $\mathbf{D}$ is projected into the k-dimensional space, giving $\mathbf{D}_k$:

$$\mathbf{D}_k = \mathbf{D}^T\mathbf{U}_k\mathbf{\Sigma}_k^{-1}.$$

The projection $\mathbf{D}_k \in \Re^{p\times k}$ is folded-in to the existing PSVD of $\mathbf{A}$ by appending it to the bottom of $\mathbf{V}_k$, giving the modified matrix $\hat{\mathbf{V}}_k \in \Re^{(d+p)\times k}$. $\mathbf{U}_k$ and $\mathbf{\Sigma}_k$ are not modified in any way with this method. Figure 3.2 gives a schematic representation of folding-in documents.

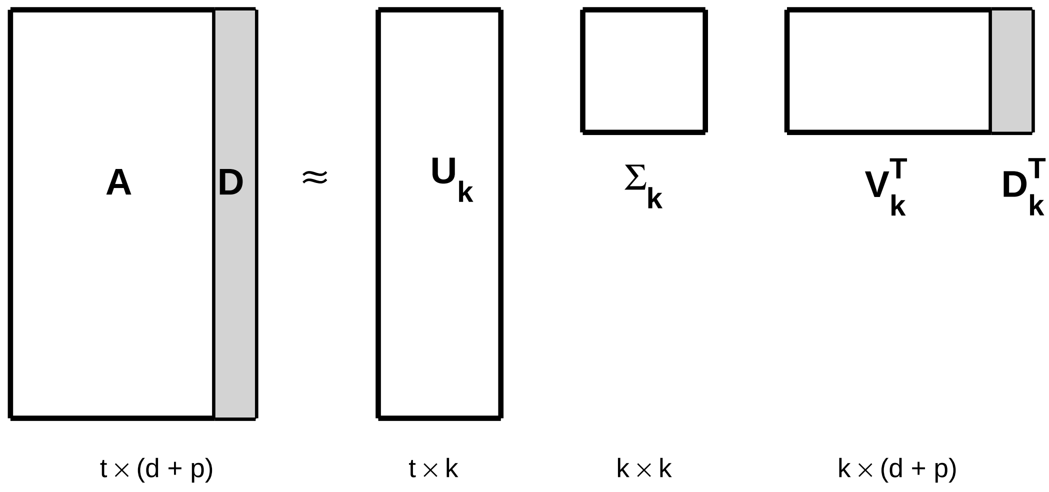

Figure 3.2: A schematic representation of folding-in p new documents.

3.2.2 Adding Terms

Folding-in terms follows a similar process. Let $\mathbf{T} \in \Re^{q \times d}$ be the term-document matrix containing the term vectors to be appended to $\mathbf{A}$, where q is the number of new terms.

$\mathbf{T}$ is projected into the k-dimensional space, giving $\mathbf{T}_k$:

$$\mathbf{T}_k = \mathbf{T}\mathbf{V}_k\mathbf{\Sigma}_k^{-1}.$$

The projection $\mathbf{T}_k \in \Re^{q \times k}$ is folded-in to the existing PSVD of $\mathbf{A}$ by appending it to the bottom of $\mathbf{U}_k$, giving the modified matrix $\hat{\mathbf{U}}_k \in \Re^{(t+p) \times k}$. $\mathbf{V}_k$ and $\mathbf{\Sigma}_k$ are not modified in any way with this method. Figure 3.3 gives a schematic representation of folding-in documents.

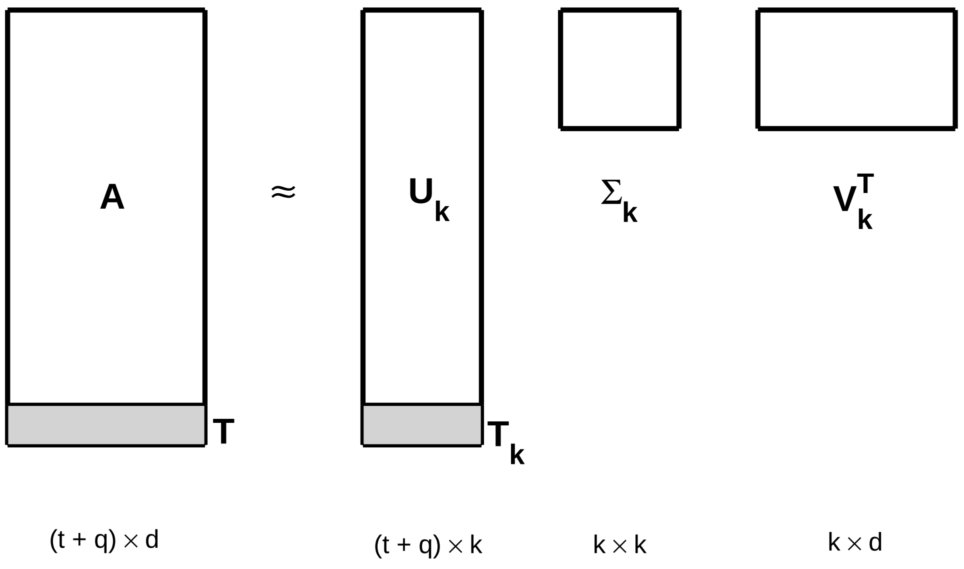

Figure 3.3: A schematic representation of folding-in q new terms.

3.2.3 Example

As with the example in Section 3.1.1, suppose that the documents from Tables 2.1 and 3.1 are combined in order to create a new term-document matrix $\hat{\mathbf{A}}$ containing 14 terms (the term *web* is being added) and 18 documents. As with the examples in Sections 2.6 and 3.1.1, the new term-document matrix is projected into two-dimensional space for ease of plotting.

See Figure 3.4 for the two-dimensional plot of the documents and terms using the folding-in method. Recall that the new documents that have been folded-in are D15–D18 and that *web* is the new term that has been folded-in. Compare Figure 3.4 to Figures 2.6 and 3.1, and note that the positions of the documents and terms that were in the original term-document matrix have not shifted in response to the new documents and term. Unlike recomputing the PSVD, in which the new data directly changes the existing latent semantic structure, folding-in merely adds to the existing latent semantic structure [26]. This means that new documents and terms do not affect the representation of documents and terms that are already in the dataset. As in Figures 2.6 and 3.1, the terms *water* and *story* still map to the same point, as do the terms *ride* and *world*, and the documents D10 and D11.

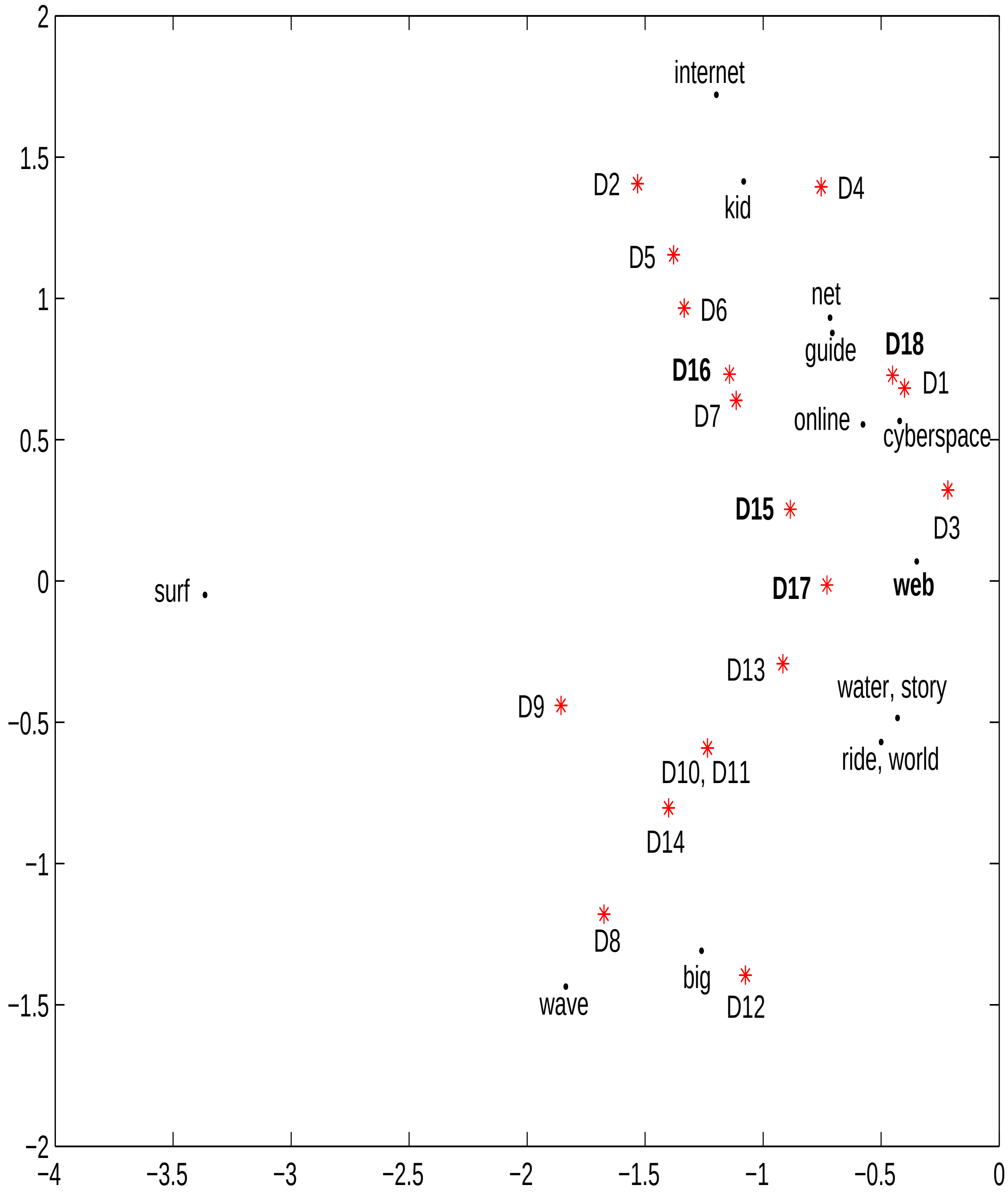

Figure 3.4: Two-dimensional plot of 14 terms and 18 documents using a PSVD that has been modified by folding-in 4 documents and 1 term.

3.3 PSVD-updating (O'Brien, and Berry, Dumais, and O'Brien)

Recomputing the PSVD each time changes are made to an LSI database can be prohibitively expensive, but the inexpensive alternative of folding-in new terms and documents may result in an inaccurate numerical representation of the dataset by the modified PSVD; this is illustrated by the experimental results in Chapter 5. Another option, aside from the recomputing or folding-in methods, is to use PSVD-updating methods to modify, or update, the existing PSVD of the current term-document matrix to incorporate the addition of new documents and terms. The first PSVD-updating techniques for LSI were introduced by O'Brien [26], and Berry, Dumais, and O'Brien [4]. In this thesis, these PSVD-updating methods are referred to as *O'Brien's PSVD-updating methods.* Section 3.3.1 describes the algorithm for PSVD-updating when new documents are added, Section 3.3.2 describes the algorithm for PSVD-updating when new terms are added, and Section 3.3.3 details the algorithm for adjusting term weights. Section 3.3 concludes with a brief example of these document and term PSVD-updating methods.

3.3.1 Adding Documents

As before, let $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t\times k}, \mathbf{\Sigma}_k \in \Re^{k\times k}$, and $\mathbf{V}_k \in \Re^{d\times k}$. Let $\mathbf{D} \in \Re^{t\times p}$ be the term-document matrix containing the document vectors to be appended to $\mathbf{A}$, where p is the number of new documents. Define

$$\hat{\mathbf{A}} = \left[\begin{array}{cc} \mathbf{A}_k & \mathbf{D} \end{array} \right].$$

Then

$$\begin{aligned} \hat{\mathbf{A}} &= \left[\begin{array}{cc} \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T & \mathbf{D} \end{array} \right], \\ \mathbf{U}_k^T\hat{\mathbf{A}} &= \left[\begin{array}{cc} \mathbf{\Sigma}_k\mathbf{V}_k^T & \mathbf{U}_k^T\mathbf{D} \end{array} \right], \quad \text{and} \end{aligned}$$

$$\mathbf{U}_k^T\hat{\mathbf{A}}\begin{bmatrix}\mathbf{V}_k & \mathbf{0}_{dp}\\ \mathbf{0}_{pk} & \mathbf{I}_p\end{bmatrix} = \begin{bmatrix}\boldsymbol{\Sigma}_k & \mathbf{U}_k^T\mathbf{D}\end{bmatrix} = \tilde{\mathbf{A}}\,,$$

where $\mathbf{I}_p \in \Re^{p\times p}$ is the identity matrix and $\mathbf{0}_{dp} \in \Re^{d\times p}$ and $\mathbf{0}_{pk} \in \Re^{p\times k}$ are zero matrices.

Let $\hat{\mathbf{A}} = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{V}}^T$ be the SVD of $\hat{\mathbf{A}}$, and let $\tilde{\mathbf{A}} = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^T$ be the SVD of $\tilde{\mathbf{A}}$. Then

$$\left(\mathbf{U}_k^T\hat{\mathbf{U}}\right)\hat{\boldsymbol{\Sigma}}\left(\hat{\mathbf{V}}^T\begin{bmatrix}\mathbf{V}_k & \mathbf{0}_{dp}\\ \mathbf{0}_{pk} & \mathbf{I}_p\end{bmatrix}\right) = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^T\,.$$

This gives

$$\begin{aligned}\mathbf{U}_k^T\hat{\mathbf{U}} &= \tilde{\mathbf{U}}\,,\\ \hat{\boldsymbol{\Sigma}} &= \tilde{\boldsymbol{\Sigma}}\,,\quad\text{and}\\ \hat{\mathbf{V}}^T\begin{bmatrix}\mathbf{V}_k & \mathbf{0}_{dp}\\ \mathbf{0}_{pk} & \mathbf{I}_p\end{bmatrix} &= \tilde{\mathbf{V}}^T\,.\end{aligned}$$

Therefore,

$$\begin{aligned}\hat{\mathbf{U}} &= \mathbf{U}_k\tilde{\mathbf{U}}\,,\quad\text{and}\\ \hat{\mathbf{V}} &= \begin{bmatrix}\mathbf{V}_k & \mathbf{0}_{dp}\\ \mathbf{0}_{pk} & \mathbf{I}_p\end{bmatrix}\tilde{\mathbf{V}}\,,\end{aligned}$$

where $\hat{\mathbf{U}} \in \Re^{t\times k}$, $\hat{\boldsymbol{\Sigma}} \in \Re^{k\times(k+p)}$, and $\hat{\mathbf{V}} \in \Re^{d\times(k+p)}$. $\hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{V}}^T$ is the updated PSVD, i.e., the PSVD of $\hat{\mathbf{A}}$.

3.3.2 Adding Terms

Adding terms follows a similar process. As before, let $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t\times k}, \mathbf{\Sigma}_k \in \Re^{k\times k}$, and $\mathbf{V}_k \in \Re^{d\times k}$. Let $\mathbf{T} \in \Re^{q\times d}$ be the term-document matrix containing the term vectors to be appended to $\mathbf{A}$, where q is the number of new terms. Define

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix}.$$

Then

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T \\ \mathbf{T} \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{U}_k^T & \mathbf{0}_{kq} \\ \mathbf{0}_{qt} & \mathbf{I}_q \end{bmatrix} \hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k\mathbf{V}_k^T \\ \mathbf{T} \end{bmatrix}, \quad \text{and}$$

$$\begin{bmatrix} \mathbf{U}_k^T & \mathbf{0}_{kq} \\ \mathbf{0}_{qt} & \mathbf{I}_q \end{bmatrix} \hat{\mathbf{A}}\mathbf{V}_k = \begin{bmatrix} \mathbf{\Sigma}_k \\ \mathbf{T}\mathbf{V}_k \end{bmatrix} = \tilde{\mathbf{A}},$$

where $\mathbf{I}_q$ is the identity matrix of size $q \times q$, $\mathbf{0}_{kq}$ is a zero matrix of size $k \times q$, and $\mathbf{0}_{qt}$ is a zero matrix of size $q \times t$.

Let $\hat{\mathbf{A}} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ be the SVD of $\hat{\mathbf{A}}$, and let $\tilde{\mathbf{A}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$ be the SVD of $\tilde{\mathbf{A}}$. Then

$$\left(\begin{bmatrix} \mathbf{U}_k^T & \mathbf{0}_{kq} \\ \mathbf{0}_{qt} & \mathbf{I}_q \end{bmatrix} \hat{\mathbf{U}}\right) \hat{\mathbf{\Sigma}} \left(\hat{\mathbf{V}}^T\mathbf{V}_k\right) = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T.$$

This gives

$$\begin{aligned}
\begin{bmatrix} \mathbf{U}_k^T & \mathbf{0}_{kq} \\ \mathbf{0}_{qt} & \mathbf{I}_q \end{bmatrix} \hat{\mathbf{U}} &= \tilde{\mathbf{U}}\,, \\
\hat{\mathbf{\Sigma}} &= \tilde{\mathbf{\Sigma}}\,, \quad \text{and} \\
\hat{\mathbf{V}}^T \mathbf{V}_k &= \tilde{\mathbf{V}}^T\,.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\hat{\mathbf{U}} &= \begin{bmatrix} \mathbf{U}_k & \mathbf{0}_{qt}^T \\ \mathbf{0}_{qk}^T & \mathbf{I}_q \end{bmatrix} \tilde{\mathbf{U}}\,, \quad \text{and} \\
\hat{\mathbf{V}} &= \mathbf{V}_k \tilde{\mathbf{V}}\,,
\end{aligned}$$

where $\hat{\mathbf{U}} \in \Re^{(t+q)\times(k+q)}$, $\hat{\mathbf{\Sigma}} \in \Re^{(k+q)\times k}$, and $\hat{\mathbf{V}} \in \Re^{d\times k}$. $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ is the updated PSVD, i.e., the PSVD of $\hat{\mathbf{A}}$.

3.3.3 Adjusting Term Weights

A typical scenario for using the updating methods in LSI is that first, new documents are added to the document collection, and the PSVD of the term-document matrix is updated to reflect the addition of these documents. If the addition of these documents increases the number of terms, the PSVD of the term-document matrix is then updated to reflect the addition of the new terms. If the addition of these new documents and terms affects the weights in the term-document matrix, then as a final step, the PSVD of the term-document matrix is updated to reflect the corresponding changes to the term-weights. For example, if the $tf \times idf$ weighting scheme discussed in Section 4.1.1 is being used, then the weights would need to be updated to reflect changes in the global *idf* factor.

Assume that there are s terms whose weights need to be adjusted in an existing term-document matrix $\mathbf{A} \in \Re^{t\times d}$. Let $\mathbf{S} \in \Re^{t\times s}$ be the *selection* matrix. A selection matrix has a column for each term whose weight must be modified: each column has one entry which is 1 (to select the term), and all other entries are 0. For example if term i is represented by column j, then the entry s_{ij} is 1, and all other entries in column j are 0. Let $\mathbf{W} \in \Re^{d \times s}$ be the matrix such that each column $\mathbf{w}_i$ contains the difference between the old term weights and the new term weights for the term i. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}} = \mathbf{A}_k + \mathbf{S}\mathbf{W}^T$ is the adjusted term-document matrix.

As before, let $\mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t \times d}$, and let $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ be the PSVD of the term-document matrix $\hat{\mathbf{A}} \in \Re^{t \times d}$. Then

$$\begin{aligned}
\hat{\mathbf{A}} &= \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T + \mathbf{S}\mathbf{W}^T\,, \\
\mathbf{U}_k^T \hat{\mathbf{A}} &= \mathbf{\Sigma}_k \mathbf{V}_k^T + \mathbf{U}_k^T \mathbf{S}\mathbf{W}^T\,, \quad \text{and} \\
\mathbf{U}_k^T \hat{\mathbf{A}} \mathbf{V}_k &= \mathbf{\Sigma}_k + \mathbf{U}_k^T \mathbf{S}\mathbf{W}^T \mathbf{V}_k = \tilde{\mathbf{A}}\,.
\end{aligned}$$

Let $\tilde{\mathbf{A}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\mathbf{V}^T$ be the PSVD of $\tilde{\mathbf{A}}$. Then

$$\begin{aligned}
\mathbf{U}_k^T \hat{\mathbf{A}} \mathbf{V}_k &= \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T\,, \quad \text{and} \\
\left(\mathbf{U}_k^T \hat{\mathbf{U}}\right) \hat{\mathbf{\Sigma}} \left(\hat{\mathbf{V}}^T \mathbf{V}_k\right) &= \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T\,.
\end{aligned}$$

This gives

$$\begin{aligned}
\mathbf{U}_k^T \hat{\mathbf{U}} &= \tilde{\mathbf{U}}\,, \\
\hat{\mathbf{\Sigma}} &= \tilde{\mathbf{\Sigma}}\,, \quad \text{and} \\
\hat{\mathbf{V}}^T \mathbf{V}_k &= \tilde{\mathbf{V}}^T\,.
\end{aligned}$$

Therefore

$$\begin{aligned}\hat{\mathbf{U}} &= \mathbf{U}_k\tilde{\mathbf{U}}\,, \quad \text{and} \\ \hat{\mathbf{V}} &= \mathbf{V}_k\tilde{\mathbf{V}}\,,\end{aligned}$$

where $\hat{\mathbf{U}} \in \Re^{t\times k}$, $\hat{\mathbf{\Sigma}} \in \Re^{k\times k}$, and $\hat{\mathbf{V}} \in \Re^{d\times k}$. $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ is the updated PSVD, i.e., the PSVD of $\hat{\mathbf{A}}$.

3.3.4 Example

As with the examples in Sections 3.1.1 and 3.2.3, suppose that the documents from Tables 2.1 and 3.1 are combined in order to create a new term-document matrix $\hat{\mathbf{A}}$ containing 14 terms (the term *web* is being added) and 18 documents. As with the examples in Sections 2.6, 3.1.1, and 3.2.3, the new term-document matrix is projected into two-dimensional space, for ease of plotting.

See Figure 3.5 for the two-dimensional plot of the documents and terms using the O'Brien's PSVD-updating methods. Recall that the new documents that have been added are D15–D18, and that *web* is the new term that has been added. Compare Figure 3.5 to Figures 2.6, 3.1, and 3.4. Note that unlike the example in Figure 3.1, in which the documents and term were folded-in, the positions of the documents and terms that were in the original term-document matrix have shifted in response to the new documents and term, similar to Figure 2.6 in which the PSVD was recomputed. This is a more accurate representation of the dataset than the result of the folding-in method. As in Figures 2.6, 3.1, and 3.1, the terms *water* and *story* still map to the same point, as do the terms *ride* and *world*, and the documents D10 and D11.

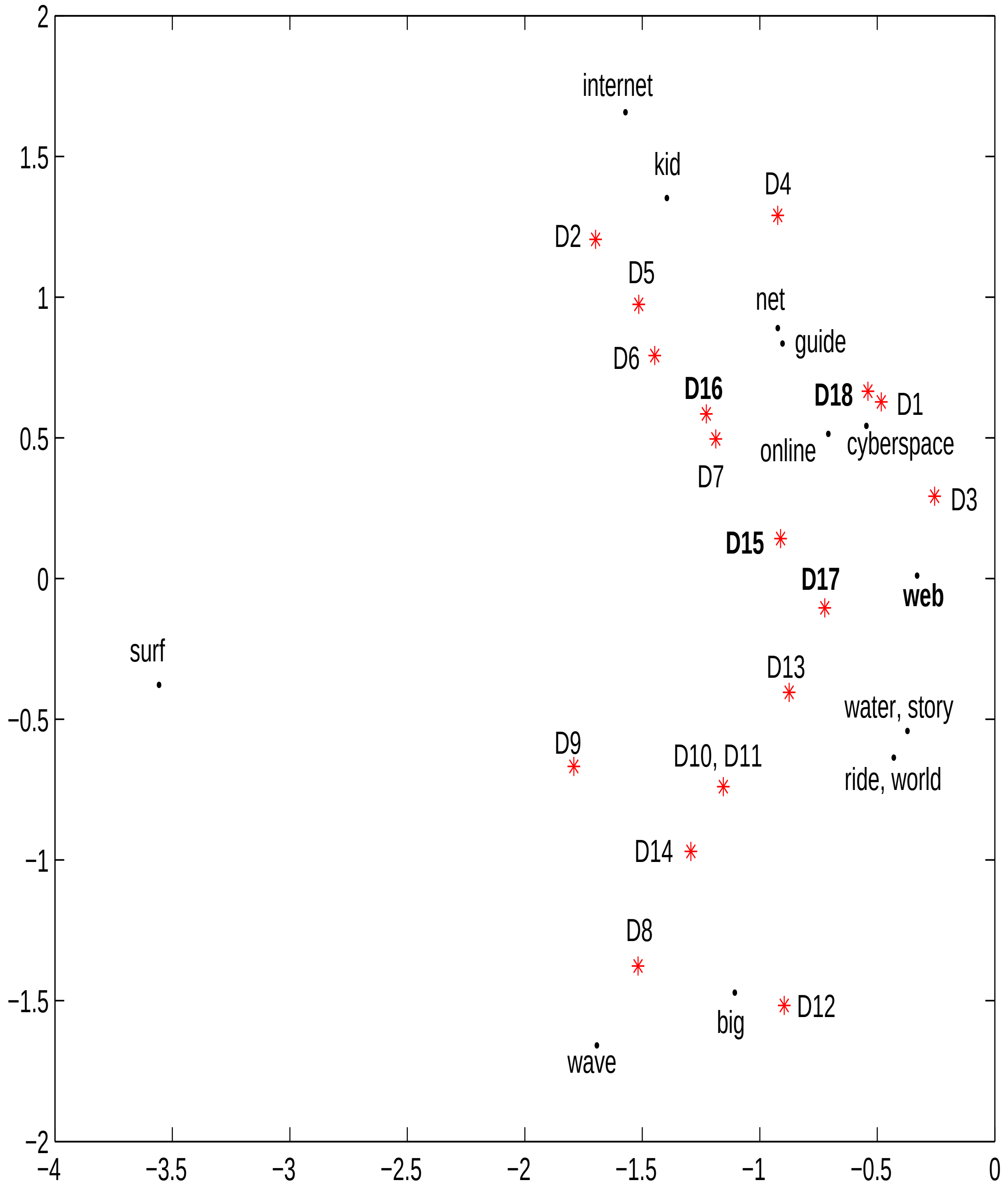

Figure 3.5: Two-dimensional plot of 14 terms and 18 documents using a PSVD that has been modified by O'Brien's PSVD-updating methods to incorporate 4 new documents and 1 new term.

3.4 PSVD-updating (Zha and Simon)

Although O'Brien's PSVD-updating methods, detailed in Section 3.3, result in a more accurate representation of the modified LSI database than do the folding-up methods described in Section 3.2, Zha and Simon have proposed alternative PSVD-updating methods for LSI [33]; their research indicates that these PSVD-updating methods give even better results than do O'Brien's methods. Section 3.4.1 describes Zha and Simon's algorithm for PSVD-updating when new documents are added, Section 3.4.2 describes their algorithm for PSVD-updating when new terms are added, and Section 3.4.3 details the algorithm for adjusting term weights. Section 3.4 concludes with a brief example of these document and term PSVD-updating methods in Section 3.4.4.

3.4.1 Adding Documents

As before, let $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t\times k}, \mathbf{\Sigma}_k \in \Re^{k\times k}$, and $\mathbf{V}_k \in \Re^{d\times k}$. Let $\mathbf{D} \in \Re^{t\times p}$ be the term-document matrix containing the document vectors to be appended to $\mathbf{A}$, where p is the number of new documents. Define

$$\hat{\mathbf{A}} = \left[\begin{array}{cc} \mathbf{A}_k & \mathbf{D} \end{array} \right].$$

The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\hat{\mathbf{A}}$. Let $\hat{\mathbf{D}} \in \Re^{t\times p}$ be defined by

$$\hat{\mathbf{D}} = \left[\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T\right]\mathbf{D}.$$

The matrix $\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T$ is an *orthogonal projection* mapping the columns of $\mathbf{D}$ into the subspace that is orthogonal to the space spanned by the columns of $\mathbf{U}_k$. Note that an orthogonal projection is a matrix $\mathbf{P}$ such that $\mathbf{P}^2 = \mathbf{P}$ and $\mathbf{P} = \mathbf{P}^T$. Having formed the projection $\hat{\mathbf{D}}$, now form the reduced QR decomposition, as described in Section 2.2, of $\hat{\mathbf{D}}$ such that

$$\mathbf{Q_D R_D} = \hat{\mathbf{D}}.$$

Recall that with such a decomposition, $\mathbf{Q_D} \in \Re^{t\times p}$ has orthogonal columns, and $\mathbf{R_D} \in \Re^{p\times p}$ is upper triangular. Then,

$$\hat{\mathbf{A}} = \left[\begin{array}{cc} \mathbf{A}_k & \mathbf{D} \end{array}\right] = \left[\begin{array}{cc} \mathbf{U}_k & \mathbf{Q_D} \end{array}\right] \left[\begin{array}{cc} \mathbf{\Sigma}_k & \mathbf{U}_k^T\mathbf{D} \\ \mathbf{0}_{pk} & \mathbf{R_D} \end{array}\right] \left[\begin{array}{cc} \mathbf{V}_k^T & \mathbf{0}_{kp} \\ \mathbf{0}_{pd} & \mathbf{I}_p \end{array}\right],$$

where $\mathbf{I}_p \in \Re^{p\times p}$ is the identity matrix and $\mathbf{0}_{pk} \in \Re^{p\times k}$, $\mathbf{0}_{kp} \in \Re^{k\times p}$, and $\mathbf{0}_{pd} \in \Re^{p\times d}$ are zero matrices.

$$\text{Let} \qquad \tilde{\mathbf{A}} \in \Re^{(k+p)\times(k+p)} \qquad \text{be defined by} \qquad \tilde{\mathbf{A}} = \left[\begin{array}{cc} \mathbf{\Sigma}_k & \mathbf{U}_k^T\mathbf{D} \\ \mathbf{0}_{pk} & \mathbf{R_D} \end{array}\right].$$

Let $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$ be the SVD of $\tilde{\mathbf{A}}$, and partition it to give

$$\tilde{\mathbf{A}} = \left[\begin{array}{cc} \tilde{\mathbf{U}}_k & \tilde{\mathbf{U}}_p \end{array}\right] \left[\begin{array}{cc} \tilde{\mathbf{\Sigma}}_k & \mathbf{0}_{kp} \\ \mathbf{0}_{pk} & \tilde{\mathbf{\Sigma}}_p \end{array}\right] \left[\begin{array}{cc} \tilde{\mathbf{V}}_k & \tilde{\mathbf{V}}_p \end{array}\right]^T,$$

where $\tilde{\mathbf{U}}_k \in \Re^{(k+p)\times k}$, $\tilde{\mathbf{U}}_p \in \Re^{(k+p)\times p}$, $\tilde{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, $\tilde{\mathbf{\Sigma}}_p \in \Re^{p\times p}$, $\tilde{\mathbf{V}}_k \in \Re^{(k+p)\times k}$, and $\tilde{\mathbf{V}}_p \in \Re^{(k+p)\times p}$.

Then the rank-k PSVD of the updated term-document matrix $\hat{\mathbf{A}} = [\mathbf{A}_k, \mathbf{D}]$ is

$$\hat{\mathbf{A}}_k = \left(\left[\begin{array}{cc} \mathbf{U}_k & \mathbf{Q_D} \end{array}\right] \tilde{\mathbf{U}}_k\right) \tilde{\mathbf{\Sigma}}_k \left(\left[\begin{array}{cc} \mathbf{V}_k & \mathbf{0}_{dp} \\ \mathbf{0}_{pk} & \mathbf{I}_p \end{array}\right] \tilde{\mathbf{V}}_k\right)^T.$$

3.4.2 Adding Terms

Adding terms follows a similar process. As before, let $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t\times k}$, $\mathbf{\Sigma}_k \in \Re^{k\times k}$, and $\mathbf{V}_k \in \Re^{d\times k}$.

Let $\mathbf{T} \in \Re^{q\times d}$ be the term-document matrix containing the term vectors to be appended to $\mathbf{A}$, where q is the number of new terms. Define

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix}.$$

The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\hat{\mathbf{A}}$. Let $\hat{\mathbf{T}} \in \Re^{d\times q}$ be defined by

$$\hat{\mathbf{T}} = \left(\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T\right)\mathbf{T}^T.$$

The matrix $\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T$ is an orthogonal projection mapping the rows of $\mathbf{T}$ into the subspace that is orthogonal to the space spanned by the columns of $\mathbf{V}_k$. Having formed the projection $\hat{\mathbf{T}}$, now form the reduced QR decomposition of $\hat{\mathbf{T}}$ such that

$$\mathbf{Q_T}\mathbf{R_T} = \hat{\mathbf{T}}.$$

Then $\mathbf{Q_T} \in \Re^{d\times q}$ has orthogonal columns, $\mathbf{R_T} \in \Re^{q\times q}$ is upper triangular, and

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_k & \mathbf{0}_{tq} \\ \mathbf{0}_{qk} & \mathbf{I}_q \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0}_{kq} \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix} \left[\mathbf{V}_k, \mathbf{Q_T}\right]^T,$$

where $\mathbf{I}_q \in \Re^{q\times q}$ is the identity matrix and $\mathbf{0}_{tq} \in \Re^{t\times q}$, $\mathbf{0}_{kq} \in \Re^{k\times q}$, and $\mathbf{0}_{qk} \in \Re^{q\times k}$ are zero matrices.

Let $\tilde{\mathbf{A}} \in \Re^{(k+q)\times(k+q)}$ be defined by $\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0}_{kq} \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix}.$

Let $\tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^T$ be the SVD of $\tilde{\mathbf{A}}$, and partition it to give

$$\tilde{\mathbf{A}} = \begin{bmatrix} \tilde{\mathbf{U}}_k & \tilde{\mathbf{U}}_q \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_k & \mathbf{0}_{kq} \\ \mathbf{0}_{qk} & \tilde{\boldsymbol{\Sigma}}_q \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{V}}_k & \tilde{\mathbf{V}}_q \end{bmatrix}^T,$$

where $\tilde{\mathbf{U}}_k \in \Re^{(k+q)\times k}$, $\tilde{\mathbf{U}}_q \in \Re^{(k+q)\times q}$, $\tilde{\boldsymbol{\Sigma}}_k \in \Re^{k\times k}$, $\tilde{\boldsymbol{\Sigma}}_q \in \Re^{q\times q}$, $\tilde{\mathbf{V}}_k \in \Re^{(k+q)\times k}$, and $\tilde{\mathbf{V}}_q \in \Re^{(k+q)\times q}$.

Then the rank-k PSVD of the updated term-document matrix

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix} \quad \text{is} \quad \hat{\mathbf{A}}_k = \left(\begin{bmatrix} \mathbf{U}_k & \mathbf{0}_{tq} \\ \mathbf{0}_{qk} & \mathbf{I}_q \end{bmatrix} \tilde{\mathbf{U}}_k \right) \tilde{\boldsymbol{\Sigma}}_k \left(\begin{bmatrix} \mathbf{V}_k & \mathbf{Q}_\mathbf{T} \end{bmatrix} \tilde{\mathbf{V}}_k \right)^T.$$

3.4.3 Adjusting Term Weights

Assume that there are s terms whose weights need to be adjusted in an existing term-document matrix $\mathbf{A} \in \Re^{t\times d}$. Let $\mathbf{S} \in \Re^{t\times s}$ be a matrix which has a column for each term whose weight must be modified: each column has one entry which is 1 (to select the term), and all other entries are 0. For example if term i is represented by column j, then the entry s_{ij} is 1, and all other entries in column j are 0. $\mathbf{S}$ is known as a *selection* matrix. Let $\mathbf{W} \in \Re^{d\times s}$ be the matrix such that each column $\mathbf{w}_i$ contains the difference between the old term weights and the new term weights for the term i. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}} = \mathbf{A}_k + \mathbf{S}\mathbf{W}^T$ is the adjusted term-document matrix.

As before, let $\mathbf{U}_k\boldsymbol{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t\times d}$, and let $\mathbf{U}_{\hat{A}}\boldsymbol{\Sigma}_{\hat{A}}\mathbf{V}_{\hat{A}}^T$ be the PSVD of the term-document matrix $\hat{\mathbf{A}} \in \Re^{t\times d}$.

Let $\hat{\mathbf{S}} \in \Re^{t\times s}$ be defined by

$$\hat{\mathbf{S}} = \left(\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T \right) \mathbf{S},$$

and let $\hat{\mathbf{W}} \in \Re^{d\times s}$ be defined by

$$\hat{\mathbf{W}} = \left(\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T \right) \mathbf{W}.$$

The matrix $\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T$ is an orthogonal projection mapping the columns of $\mathbf{S}$ into the subspace that is orthogonal to the space spanned by the columns of $\mathbf{U}_k$, and the matrix $\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T$ is an orthogonal projection mapping the columns of $\mathbf{W}$ into the subspace that is orthogonal to the space spanned by the columns of $\mathbf{V}_k$. Form the reduced QR decomposition of $\hat{\mathbf{S}}$ such that

$$\mathbf{Q_S}\mathbf{R_S} = \hat{\mathbf{S}}.$$

Then $\mathbf{Q_S} \in \Re^{t\times s}$ has orthogonal columns, and $\mathbf{R_S} \in \Re^{s\times s}$ is upper triangular. Also form the reduced QR decomposition of $\hat{\mathbf{W}}$ such that

$$\mathbf{Q_W}\mathbf{R_W} = \hat{\mathbf{W}}.$$

Then $\mathbf{Q_W} \in \Re^{d\times s}$ has orthogonal columns, and $\mathbf{R_W} \in \Re^{s\times s}$ is upper triangular. Using these decompositions,

$$\begin{aligned}
\hat{\mathbf{A}} &= \mathbf{A}_k + \mathbf{S}\mathbf{W}^T \\
&= \begin{bmatrix} \mathbf{U}_k & \mathbf{Q_S} \end{bmatrix} \left(\begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0}_{ks} \\ \mathbf{0}_{sk} & \mathbf{0}_{ss} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R_S} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R_W} \end{bmatrix}^T \right) \begin{bmatrix} \mathbf{V}_k & \mathbf{Q_W} \end{bmatrix}^T,
\end{aligned}$$

where $\mathbf{0}_{ks} \in \Re^{k\times s}$, $\mathbf{0}_{sk} \in \Re^{s\times k}$, and $\mathbf{0}_{ss} \in \Re^{s\times s}$ are zero matrices.

Let $\tilde{\mathbf{A}} \in \Re^{(k+s)\times(k+q)}$ be defined by

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0}_{ks} \\ \mathbf{0}_{sk} & \mathbf{0}_{ss} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R_S} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R_W} \end{bmatrix}^T.$$

Let $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$ be the SVD of $\tilde{\mathbf{A}}$, and partition it to give

$$\tilde{\mathbf{A}} = \begin{bmatrix} \tilde{\mathbf{U}}_k & \tilde{\mathbf{U}}_s \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{\Sigma}}_k & \mathbf{0}_{ks} \\ \mathbf{0}_{sk} & \tilde{\mathbf{\Sigma}}_s \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{V}}_k & \tilde{\mathbf{V}}_s \end{bmatrix}^T,$$

where $\tilde{\mathbf{U}}_k \in \Re^{(k+s)\times k}$, $\tilde{\mathbf{U}}_s \in \Re^{(k+s)\times s}$, $\tilde{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, $\tilde{\mathbf{\Sigma}}_s \in \Re^{s\times s}$, $\tilde{\mathbf{V}}_k \in \Re^{(k+s)\times k}$, and $\tilde{\mathbf{V}}_s \in \Re^{(k+s)\times s}$.

Then the rank-k PSVD of the updated term-document matrix $\hat{\mathbf{A}} = \mathbf{A}_k + \mathbf{S}\mathbf{W}^T$ is

$$\hat{\mathbf{A}}_k = \left(\begin{bmatrix} \mathbf{U}_k & \mathbf{Q}_\mathbf{S} \end{bmatrix} \tilde{\mathbf{U}}_k\right) \tilde{\mathbf{\Sigma}}_k \left(\begin{bmatrix} \mathbf{V}_k & \mathbf{Q}_\mathbf{W} \end{bmatrix} \tilde{\mathbf{V}}_k\right)^T.$$

3.4.4 Example

As with the examples in Sections 3.1.1, 3.2.3, and 3.3.4, suppose that the documents from Tables 2.1 and 3.1 are combined in order to create a new term-document matrix $\hat{\mathbf{A}}$ containing 14 terms (the term *web* is being added) and 18 documents. As with the previous examples, the new term-document matrix is projected into two-dimensional space for ease of plotting.

See Figure 3.6 for the two-dimensional plot of the documents and terms using the Zha and Simon's PSVD-updating methods. Recall that the new documents that have been added are D15–D18 and that *web* is the new term that has been added. Compare Figure 3.5 to Figures 2.6, 3.1, 3.4, and 3.5. Note that unlike the example in Figure 3.1 in which the documents and terms were folded in, the positions of the documents and terms that were in the original term-document matrix have shifted in response to the new documents and term, similar to Figure 2.6 in which the PSVD was recomputed. Also note that Figure 3.6 more closely resembles Figure 3.1 in which the PSVD is recomputed, than do either Figures 3.4 or 3.5. Zha and Simon's PSVD-updating methods give a more accurate representation of the dataset than either the folding-in methods or O'Brien's PSVD-updating methods [33]. As before, the terms *water* and *story* still map to the same point, as do the terms *ride* and *world*, and the documents D10 and D11.

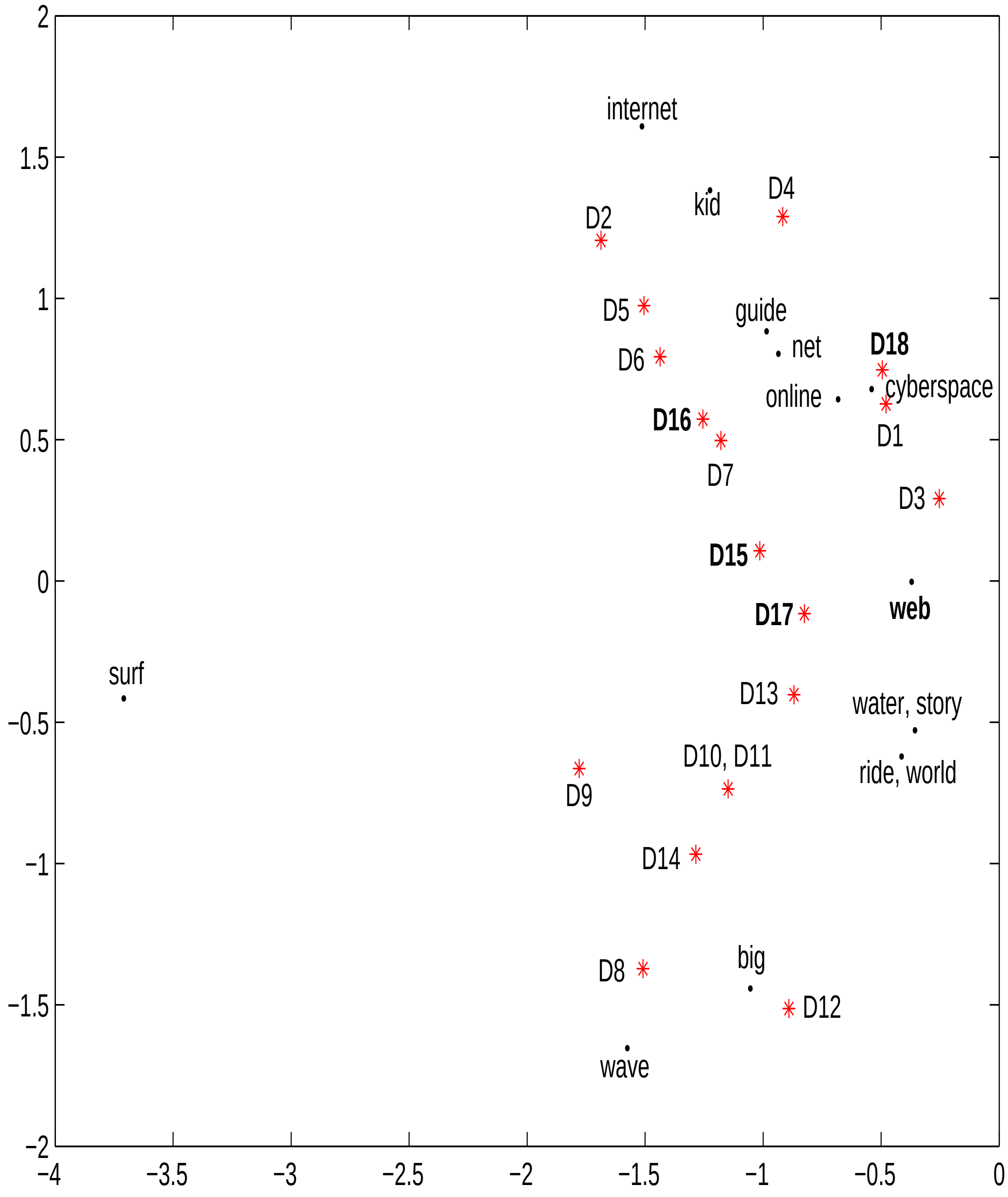

Figure 3.6: Two-dimensional plot of 14 terms and 18 documents using a PSVD that has been modified by Zha and Simon's PSVD-updating methods to incorporate 4 new documents and 1 new term.

3.5 Folding-up

It is well known that folding-in is a very inexpensive way to incorporate new information into an existing term-document matrix, compared to recomputing the PSVD [4]. However, because the folding-in method bases the new representation of the data on the existing latent semantic structure, its use can cause deterioration in the numerical representation of the dataset by the PSVD, even after only a small number of updates. On the other hand, PSVD-updating gives the same result (to within rounding errors) as recomputing the PSVD, with significantly less computational expense. However, it is still significantly more computationally expensive than folding-in.

We now describe a new hybrid method that uses a combination of folding-in and PSVD-updating in order to reduce the computational expense of PSVD-updating even further, without significantly degrading the results. This new method, called *folding-up*, will be shown to be computationally faster than either recomputing the PSVD or using PSVD-updating methods, with no significant difference in retrieval performance. The idea is that when there are new documents or terms to be added to the LSI database, they are first folded-in, although the original vectors are stored for later use. After a certain amount of folding-in has been done (as described below), the changes that have been made by the folding-in process are discarded, and all the document or term vectors that have been folded-in are instead added to the LSI database using a PSVD-updating method, such as the method introduced by Zha and Simon and discussed in Section 3.4. The original document or term vectors that have been added to the LSI database by the PSVD-updating process can then be discarded. The goal of the folding-up method is to switch from the process of folding-in to that of using a PSVD-updating method before the accuracy of the numerical representation of the database degrades significantly from the folding-in process. After the PSVD-update has been performed, the cycle begins again; documents and/or terms are again folded-in until it is deemed necessary to switch to a PSVD-updating method. In folding-up, the PSVD-updating process can be thought of as a correction step, but it is important to decide at what point this step becomes necessary.

Recall from Equation (2.3) that the PSVD of the term-document matrix $\mathbf{A}$ is $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$, where k is the number of dimensions. Also recall, from Section 2.3, that the columns of matrix $\mathbf{U}_k$ are mutually orthogonal, as are the columns of matrix

$\mathbf{V}_k$. Recall from Section 3.2 that folding-in p documents appends p rows to the bottom of $\mathbf{V}_k$, to get $\hat{\mathbf{V}}_k$, and folding-in q terms appends q rows to the bottom of $\mathbf{U}_k$, to get $\hat{\mathbf{U}}_k$. This process corrupts the orthogonality of the columns of $\hat{\mathbf{U}}_k$ and $\hat{\mathbf{V}}_k$, potentially leading to a misrepresentation of the dataset. With the folding-up method, the deterioration of the orthogonality of $\hat{\mathbf{V}}_k$ (when documents are being added) or $\hat{\mathbf{U}}_k$ (when terms are being added) is monitored. This distortion of orthogonality is an indication of how much inaccuracy has been introduced to the representation of the dataset with the addition of new documents and terms. When the loss of orthogonality reaches a limit that is deemed unacceptable, the changes that have been made by the folding-in method are discarded; i.e., the vectors that have been appended to $\mathbf{V}_k$ and $\mathbf{U}_k$ are discarded. PSVD-updating methods are then applied such that the modified PSVD reflects the addition of all of the document and term vectors that have been folded-in since the last update. The process then continues with folding-in until the next update.

The loss of orthogonality in $\hat{\mathbf{U}}_k$ and $\hat{\mathbf{V}}_k$ can be measured by

$$\|\hat{\mathbf{U}}_k^T\hat{\mathbf{U}}_k - \mathbf{I}_k\|_\infty \quad \text{and} \quad \|\hat{\mathbf{V}}_k^T\hat{\mathbf{V}}_k - \mathbf{I}_k\|_\infty, \tag{3.3}$$

respectively, where $\|\cdot\|_\infty$ is the matrix infinity norm, and $\mathbf{I}_k \in \Re^{k\times k}$ is the identity matrix. However, when the term-document matrix is very large, this measurement is computationally expensive. Recall from Section 3.2 that

$$\hat{\mathbf{U}}_k = \begin{bmatrix} \mathbf{U}_k \\ \mathbf{T}_k \end{bmatrix}, \quad \text{and}$$

$$\hat{\mathbf{V}}_k = \begin{bmatrix} \mathbf{V}_k \\ \mathbf{D}_k \end{bmatrix}.$$

Then

$$\hat{\mathbf{U}}_k^T\hat{\mathbf{U}}_k = \begin{bmatrix} \mathbf{U}_k^T & \mathbf{T}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{U}_k \\ \\ \mathbf{T}_k \end{bmatrix} = \left[\mathbf{U}_k^T\mathbf{U}_k + \mathbf{T}_k^T\mathbf{T}_k\right], \quad \text{and} \tag{3.4}$$

$$\hat{\mathbf{V}}_k^T\hat{\mathbf{V}}_k = \begin{bmatrix} \mathbf{V}_k^T & \mathbf{D}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{V}_k \\ \mathbf{D}_k \end{bmatrix} = \left[\mathbf{V}_k^T\mathbf{V}_k + \mathbf{D}_k^T\mathbf{D}_k\right]. \tag{3.5}$$

Substituting Equations (3.4) and (3.5) into Equation (3.3) gives

$$\|\mathbf{U}_k^T\mathbf{U}_k + \mathbf{T}_k^T\mathbf{T}_k - \mathbf{I}_k\|_\infty = \|\mathbf{I}_k + \mathbf{T}_k^T\mathbf{T}_k - \mathbf{I}_k\|_\infty = \|\mathbf{T}_k^T\mathbf{T}_k\|_\infty \quad \text{and} \tag{3.6}$$

$$\|\mathbf{V}_k^T\mathbf{V}_k + \mathbf{D}_k^T\mathbf{D}_k - \mathbf{I}_k\|_\infty = \|\mathbf{I}_k + \mathbf{D}_k^T\mathbf{D}_k - \mathbf{I}_k\|_\infty = \|\mathbf{D}_k^T\mathbf{D}_k\|_\infty. \tag{3.7}$$

Thus, the deterioration in the orthogonality of $\hat{\mathbf{U}}_k$ and $\hat{\mathbf{V}}_k$ can be measured by $\|\mathbf{T}_k^T\mathbf{T}_k\|_\infty$ and $\|\mathbf{D}_k^T\mathbf{D}_k\|_\infty$, respectively.

Using the results from Equations (3.6) and (3.7) rather than Equations (3.3) to monitor the deterioration in orthogonality offers significant computational savings. This is because $\hat{\mathbf{U}}_k$ is of size $(t+q)\times k$, whereas $\mathbf{T}_k$ is of size $q \times k$, with typically $t \gg q$, and so the multiplication $\mathbf{T}_k^T\mathbf{T}_k$ is much less expensive than the multiplication $\hat{\mathbf{U}}_k^T\hat{\mathbf{U}}_k$. Likewise, $\hat{\mathbf{V}}_k$ is of size $(d+p)\times k$ whereas $\mathbf{D}_k$ is of size $p \times k$, with typically $d \gg p$, thus the multiplication $\mathbf{D}_k^T\mathbf{D}_k$ is much less expensive than the multiplication $\hat{\mathbf{V}}_k^T\hat{\mathbf{V}}_k$.

The process of folding-up also has the overhead of saving the document vectors that are being folded-in between updates; however, it repays this cost with a saving in computation time, coupled with the precision advantages of updating. Experiments in Chapter 5 demonstrate that folding-up produces results that are not statistically different from those produced by recomputing the PSVD.

Chapter 4

Term Weighting and Evaluation Metrics

Term weighting, an approach commonly used to improve *retrieval performance*, is discussed in Section 4.1. Retrieval performance is the ability of a retrieval system to return relevant documents, while minimizing the return of nonrelevant documents. The retrieval performance metrics *precision* and *recall* are discussed in Section 4.2.1, and *average precision* is discussed in Section 4.2.2. An example of forming a *precision versus recall curve* is given in Section 4.2.3.

4.1 Term Weights

As discussed in Section 1.1, a term-document matrix $\mathbf{A}$ has t rows and d columns, where t is the number of semantically significant terms, or *index* terms, and d is the number of documents in the text collection. Not all of the index terms for a document collection are equally useful in distinguishing the contents of the documents. Terms that are in all the documents are not useful as index terms, and it is for this reason that lists of stopwords are used to remove common words from the list of index terms, as discussed in Section 1.1. In contrast, a word that appears in only a few documents may be very useful as an index term, because it narrows the number of documents that are likely to be of relevance to the user [1]. Each term-document entry a_{ij} indicates the importance of term i relative to document j, where $1 \leq i \leq t$, and $1 \leq j \leq d$. The entries may be binary, raw term frequencies, or weighted term frequencies. As a simplification, the term-weights are generally assumed to be mutually independent [1, 28].

As discussed in Section 1.1, local and global weighting are the two forms of weighting commonly used in practice. Local weighting indicates the importance of a term in a document, whereas global weighting indicates the importance of the term in the document collection as a whole.

Four commonly used local weighting schemes are *binary* [2, 28], *logarithmic* [2, 17], *term frequency* [1, 2, 28], and *augmented normalized term frequency* [2, 17, 28]. Denote the frequency of a term i in document j by f_{ij}. The binary term weight function χ is defined as

$$\chi\left(f_{ij}\right) = \begin{cases} 1 & \text{if} \quad f_{ij} > 0 \\ 0 & \text{if} \quad f_{ij} = 0. \end{cases} \tag{4.1}$$

The logarithmic term weight is defined as $\log(1 + f_{ij})$, and the term frequency term weight is simply f_{ij}, the number of times term i appears in document j. The augmented normalized term frequency weighting scheme is defined as follows:

$$\frac{\chi\left(f_{ij}\right) + \left(\frac{f_{ij}}{\max_k f_{kj}}\right)}{2}, \tag{4.2}$$

where $\chi(f_{ij})$ is the binary weight function from Equation (4.1), and $\max_k f_{kj}$ is the maximum frequency of any term in document j. According to Berry and Browne [2], the choice of the local weighting scheme depends on the type of vocabulary prevalent in the document collection. They note that with general or varied vocabularies, the use of raw term frequencies f_{ij} is often sufficient, whereas for scientific vocabularies, normalized term frequency schemes such as Equation (4.2) are recommended.

Although global weights are typically not used in dynamic environments, thus avoiding the need to update the global weight factor as the number of documents in the document collection changes, global weights are often used in static environments in which the number of documents in the document collection is stable [2]. Global weighting schemes have the effect of reducing the weight of terms that are found in many documents [26].

Four commonly used global weighting schemes are *inverse document frequency* (*idf*) [1, 2, 12, 28], *global frequency* $\times$ *inverse document frequency* ($gf \times$ *idf*) [2, 12], *normal* [2, 12], and *probabilistic inverse* [2, 17, 28]. As before, denote the frequency of a term i in document j by f_{ij}, and let $\chi(f_{ij})$ be the binary weight function from Equation (4.1). Let N be the total number of documents in the document collection, let gf_i be the global frequency of term i, and let df_i be the number of documents containing term i. The *idf* weighting scheme, which provides a measure of the inverse of the frequency of a term among all the documents in the document collection, is defined as

$$\log\left(\frac{N}{df_i}\right) + 1,$$

and the $gf \times idf$ global weighting scheme, which is the ratio of the global frequency of a term to the number of documents in which it occurs, is defined as

$$\frac{gf_i}{df_i}.$$

The normal global weighting scheme normalizes the local weights for each term i, and is defined as

$$\sqrt{\frac{1}{\sum_j (f_{ij})^2}},$$

and the probabilistic inverse global weighting scheme is defined as

$$\log\left(\frac{N - df_i}{df_i}\right).$$

A common combination of local and global weighting is to use the $tf \times idf$ weighting scheme [1, 2, 28], as described in Section 4.1.1.

4.1.1 Tf×Idf Weighting Scheme

Although there are many term-weighting schemes, the most popular of these are some form of the $tf \times idf$ weighting scheme. The tf factor in the $tf \times idf$ weighting scheme stands for the term frequency (the number of times a term appears in a document); this is a measure of the how well a particular term represents a particular document [1]. This is a local weight. The *idf* factor stands for *inverse document frequency*, which is a measure of the inverse of the frequency of a term among all the documents in the document collection; this is a global weight. The idea behind using the *idf* factor is that words that are found in only a small proportion of the document collection are more likely to be useful in differentiating between documents [1]. The normalized tf factor for term i in document j is given by

$$tf_{ij} = \frac{f_{ij}}{\max_k f_{kj}},$$

where f_{ij} is the raw frequency of term i in document j, and $\max_k f_{kj}$ is the raw frequency of the most common index term, k, in document j. The $tf \times idf$ weight a_{ij} for term i in document j is then

$$a_{ij} = tf_{ij} \times \left(\frac{gf_i}{df_i}\right), \tag{4.3}$$

where $\frac{gf_i}{df_i}$ is the *idf* factor for term i. Normalizing Equation (4.3) gives

$$a_{ij} = \frac{tf_{ij} \times idf_i}{\sqrt{\sum\limits_{k=1}^{t} (tf_{ij})^2 (idf_i)^2}}, \tag{4.4}$$

where t is the number of index terms.

For query weights, a slightly modified formula is suggested by Salton and Buckley [28]. The $tf \times idf$ weight w_{iq} for term i in query q is given by

$$\begin{aligned} w_{iq} &= \left(0.5 + \frac{0.5\ f_{iq}}{\max_k f_{kq}}\right) \times \log \frac{N}{df_i} \\ &= (0.5 + 0.5\ tf_{iq}) \times idf_i. \end{aligned} \tag{4.5}$$

Normalizing Equation (4.5) gives

$$w_{iq} = \frac{(0.5 + 0.5\ tf_{iq}) \times idf_i}{\sqrt{(0.5 + 0.5\ tf_{iq})^2\,(idf_i)^2}}. \tag{4.6}$$

4.2 Metrics

In response to a user's query, an IR system returns a ranked list of documents. In order to measure the quality of this list, and thus the quality of the retrieval performance, it is necessary to define performance metrics. Retrieval performance is the ability of an IR system to retrieve relevant information and eschew irrelevant information [2]. Two standard metrics used to evaluate IR systems are *precision* and *recall.* These metrics are discussed in Section 4.2.1, and *average* precision is discussed in Section 4.2.2.

4.2.1 Precision and Recall

As an IR performance metric, *precision*, P, is defined as the proportion of retrieved documents that are relevant out of all the documents that are retrieved; this is the *purity* of the retrieval, and it is expressed as

$$P = \frac{D_{rr}}{D_{tr}}, \tag{4.7}$$

where D_{rr} is the number of retrieved documents that are relevant, and D_{tr} is the total number of retrieved documents.

The IR performance metric *recall*, R, is defined as the proportion of relevant documents that are retrieved, out of all the relevant documents; this is the *completeness* of the retrieval, and is expressed as

$$R = \frac{D_{rr}}{N_r}, \tag{4.8}$$

where D_{rr} is again the number of retrieved documents that are relevant, and N_r is the total number of relevant documents.

Ideally, an IR system will have both high precision and high recall; however there is a known trade-off between precision and recall. Research indicates that as precision increases, recall decreases [6].

The definitions of precision and recall, in Equations (4.7) and (4.8) respectively, assume that all the documents in the dataset have been examined by the user to determine their relevance. It is typically not the case, however, that a ranked list of the entire document collection is returned to the user of the IR system (although this *is* the case for the experiments in Chapter 5). Generally, either a list of the top ranked n documents is returned, or a ranked list of all those documents whose similarity score (using, for example, the cosine similarity measure) is above some threshold is returned. In either case, the user examines this list starting with the top document; the precision and recall measures change as the user proceeds through the list [1]. Thus, to properly evaluate the retrieval performance it is necessary to plot a *precision versus recall curve.* The precision versus recall curve is generally based on 11 standard recall levels. These recall levels are $0\%, 10\%, 20\%, \ldots, 100\%$. If the recall levels for a query are distinct from the 11 standard recall levels, then the precision at each level is calculated using the following interpolation procedure (this procedure is also used to determine the precision at the standard recall level 0%) [1]. Let r_i denote the standard recall level i, where $i \in \{0, 1, 2 \ldots, 10\}$; for example, let r_3 denote the standard recall level of 30%. Then the interpolated precision, P, at the ith standard recall level is the maximum known precision at any recall level between recall level i and recall level $i + 1$ [1]; it is expressed as

$$P(r_i) = \max_{r_i \leq r \leq r_{i+1}} P(r), \tag{4.9}$$

where $P(r_i)$ is the precision at the ith standard recall level.

4.2.2 Average Precision

In order to compare the performance of IR algorithms, it is necessary to compare the precision versus recall curves for a collection of test queries; the set of queries is run using each algorithm. To effectively evaluate the performance of an algorithm over all the queries, it is necessary to average the precision values at each of the standard recall levels [1]. This is expressed as

$$\bar{P}(r) = \sum_{j=1}^{N_q} \frac{P_j(r)}{N_q}, \tag{4.10}$$

where $\bar{P}(r)$ is the average precision at standard recall level r, N_q is the number of test queries, and $P_j(r)$ is the precision at recall level r for the jth test query.

4.2.3 Precision versus Recall Curve Example

Suppose that a document collection contains 100 documents, denoted $D_1, D_2, \ldots, D_{100}$, and suppose that for a particular test query, there are 10 relevant documents, which make up the set $\{D_1, D_{23}, D_{41}, D_{49}, D_{53}, D_{59}, D_{72}, D_{77}, D_{86}, D_{97}\}$. Table 4.1 gives a ranked listing of the top 10 documents retrieved by a fictional IR system, where $\bullet$ denotes a relevant document. Table 4.1 also gives the recall and precision values, calculated as described in Section 4.2.1. Figure 4.1 depicts the precision versus recall curve for this example. Note that because only 6 out of 10 relevant documents are retrieved, the precision for recall levels 7 to 10 (70%, ..., 100%) is 0.

Ranking	Document ID	Relevant	Recall	Precision
1	D_{44}	●	$\frac{1}{10} = 0.10$	$\frac{1}{1} = 1.00$
2	D_{16}		$\frac{1}{10} = 0.10$	$\frac{1}{2} = 0.50$
3	D_{97}	●	$\frac{2}{10} = 0.20$	$\frac{2}{3} = 0.67$
4	D_{59}	●	$\frac{3}{10} = 0.30$	$\frac{3}{4} = 0.75$
5	D_{62}		$\frac{3}{10} = 0.30$	$\frac{3}{5} = 0.60$
6	D_{23}	●	$\frac{4}{10} = 0.40$	$\frac{4}{6} = 0.67$
7	D_{100}		$\frac{4}{10} = 0.40$	$\frac{4}{7} = 0.57$
8	D_{77}	●	$\frac{5}{10} = 0.50$	$\frac{5}{8} = 0.62$
9	D_{1}	●	$\frac{6}{10} = 0.60$	$\frac{6}{9} = 0.67$
10	D_{85}		$\frac{6}{10} = 0.60$	$\frac{6}{10} = 0.60$

Table 4.1: Recall and Precision Example.

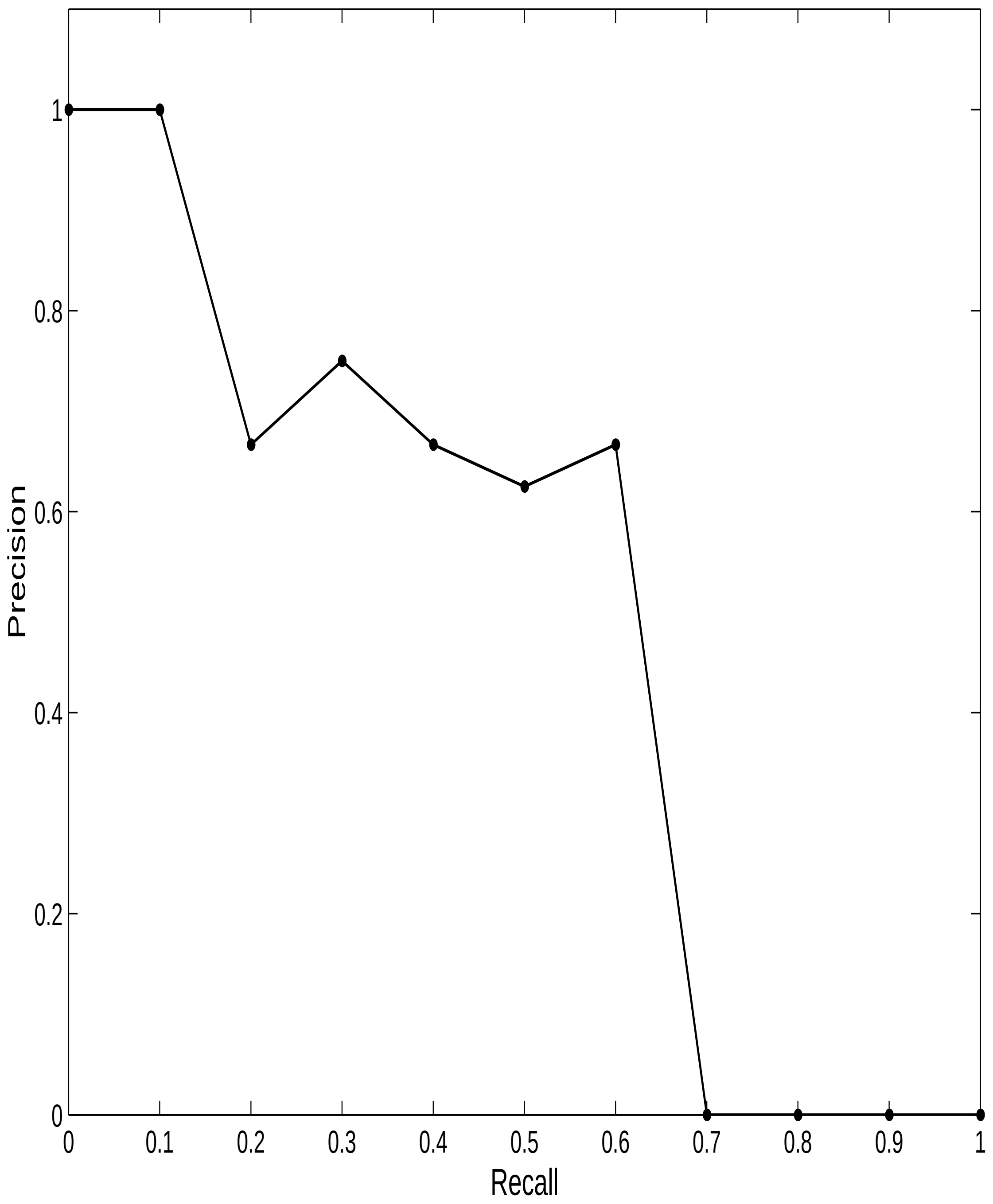

Figure 4.1: Precision versus recall curve at 11 standard recall levels.

Chapter 5

Experiments and Results

The experiments in this thesis are run using *Matlab* Release 13 on an Ultra3 SunFire V880 (Solaris 8 operating system). Three document collections are used: Medline, Cranfield, and CISI [9]. Sections 5.1–5.3 describe the experiments and discuss the results for each collection respectively. Each experiment compares the average precision for the four methods discussed in Chapter 3, i.e., recomputing, folding-in documents, PSVD-updating, and the new hybrid folding-up method. Note that in these experiments, Zha and Simon's PSVD-updating method is used; Zha and Simon have clearly shown [33] that their updating method outperforms O'Brien's PSVD updating method, in terms of retrieval performance. For each experiment, two versions of the folding-up method are tested; the first measures orthogonal deterioration in order to determine when to switch from folding-in to updating, whereas the second method switches from folding-in to updating based on how many documents have been folded-in, relative to the number of documents in the most recently updated term-document matrix. These two folding-up methods will henceforth be referred to as folding-up method 1 and folding-up method 2, respectively.

Each experiment includes a comparison of the average precisions for three trials of folding-up method 1, in which the error threshold is varied. The error threshold is set at 0.75, 1.00, and 1.50 respectively. Each experiment also includes a comparison of the average precisions for three trials of folding-up method 2. Here the percentage of documents folded-in before an update is performed, relative to the size of the term-document matrix, is varied. The percentage is set at $8\%, 10\%$, and 12% respectively. Although in each case these are nominal ranges, they give some indication of the sensitivity of the folding-up methods to these parameters.

A Jarque-Bera test for goodness-of-fit to a uniform distribution [19] is run for the data in each experiment. These tests indicate that although the data for the Medline document collection are uniformly distributed, the data for the Cranfield and

CISI collections are not. Thus, the statistical comparisons are made pairwise using a distribution-free Kruskal-Wallis test. The Kruskal-Wallis test is a non-parametric equivalent to an ANOVA test [14, 18]. For each experiment, the final average precision values for each method are compared using the Kruskal-Wallis test at significance level 0.05.

In each experiment, the term-document matrix for the whole text collection is partitioned into an initial matrix and a number of smaller submatrices. The initial matrix is incrementally enlarged by iteratively appending the submatrices, and the average precision for each method is plotted at each increment. Note that this precision is averaged not only over the number of queries, but also over the 11 standard recall levels. In each case, the PSVD of the initial matrix is computed using the `svds` function for sparse matrices in *Matlab*. The measure of similarity for each experiment is the cosine similarity measure. A local normalized term frequency weighting scheme is used in each experiment; no global weighting is used.

5.1 Medline Text Collection

The Medline text collection contains 1033 medical abstracts and 30 queries. After stopword removal and stemming, there are 5735 terms, giving a 5735×1033 term-document matrix. This matrix is partitioned into an initial matrix with 433 documents and a number of submatrices to be appended incrementally. For each of the experiments with the Medline collection, $k = 75$, where k is the number of dimensions used in computing the PSVD. Table 5.1 gives a comparison of CPU times for each of the experiments.

5.1.1 Medline: Experiment 1

The Medline term-document matrix is partitioned into an initial matrix of 433 documents and 300 submatrices of 2 documents each. Thus, there are 300 additions of 2 documents to the term-document matrix. Figure 5.1, shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. Although the difference between the results is slight, it is interesting to note that in this case the smallest error threshold does not return the highest average precision.

Figure 5.2 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at 8%, 10%, and 12%. The difference between the results is minimal, which is an indication of the robustness of the method.

Figure 5.3 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 8%. As expected, Figure 5.3 shows that the average precision for folding-in deteriorates rapidly relative to recomputing the PSVD; the final average precision is significantly different from that of recomputing the PSVD ($p = 0.0093$). The average precision for updating and folding-up, methods 1 and 2, is much higher than that of folding-in, and the final average precision in each case is not significantly different than that of recomputing ($p = 0.2488$ for updating, $p = 0.5154$ for folding-up method 1, and $p = 0.6898$ for folding-up method 2). See Table 5.1 for a comparison of the computation times.

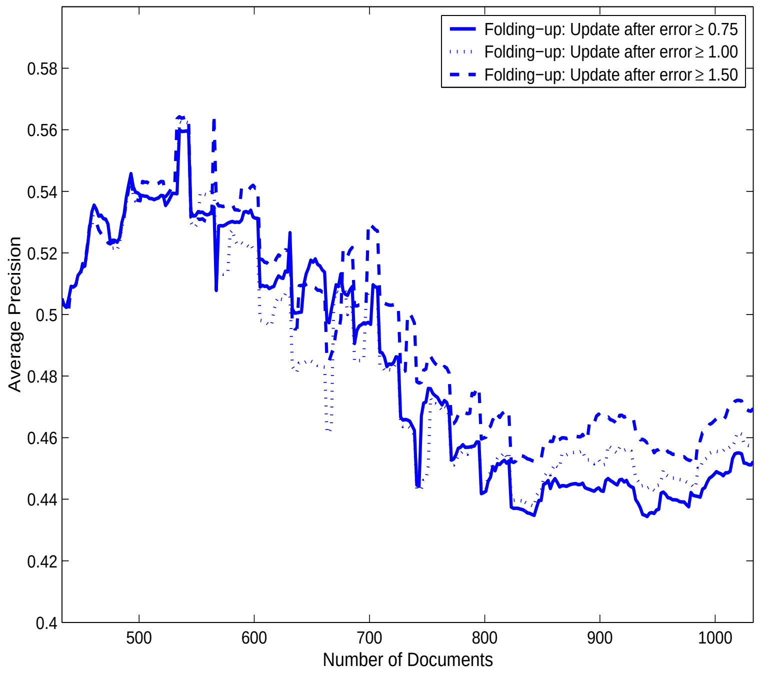

Figure 5.1: Comparison of average precisions of folding-up method 1 for the Medline collection with 600 documents added in 300 groups of 2.

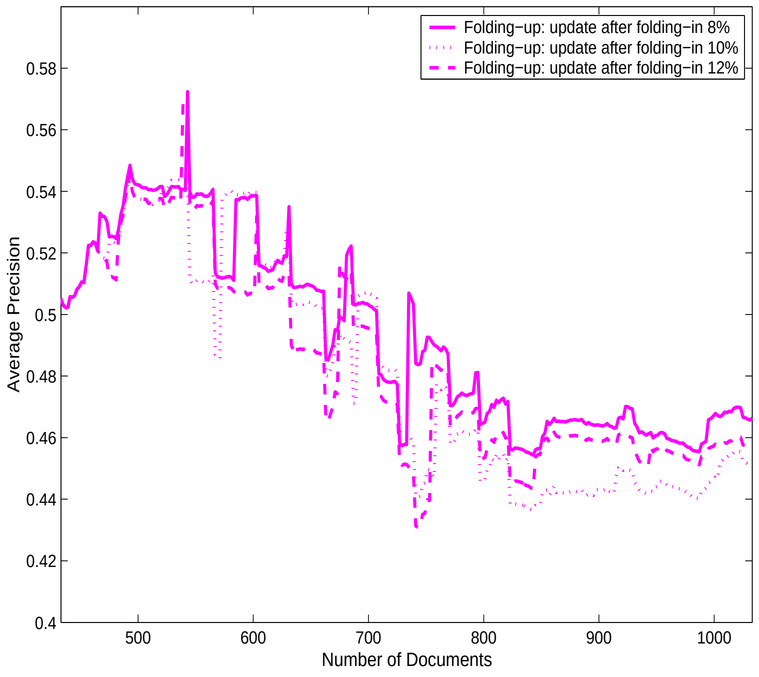

Figure 5.2: Comparison of average precisions of folding-up method 2 for the Medline collection with 600 documents added in 300 groups of 2.

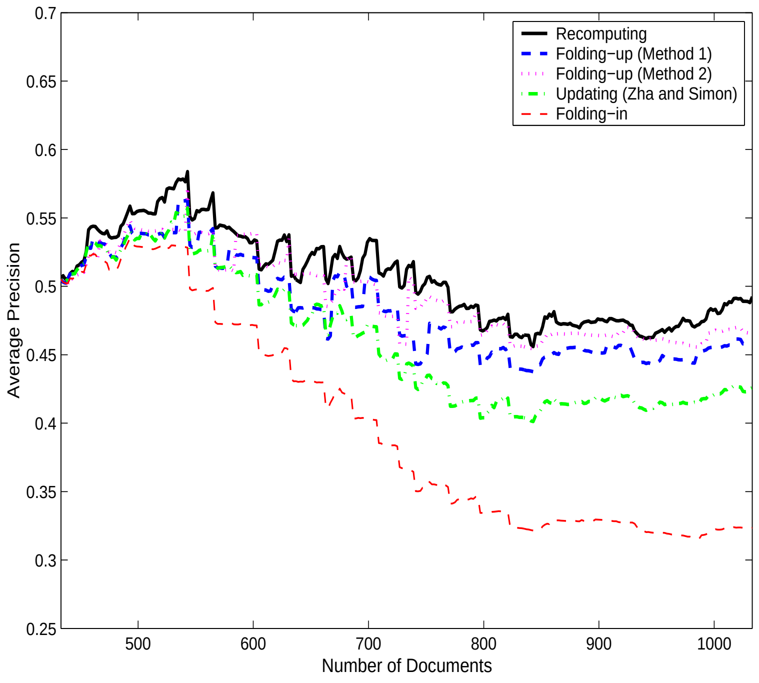

Figure 5.3: Comparison of average precisions for the Medline collection with 600 documents added in 300 groups of 2.

5.1.2 Medline: Experiment 2

The Medline term-document matrix is partitioned into an initial matrix of 433 documents and 150 submatrices of 4 documents each. Figure 5.4, shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50; Figure 5.2 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. The difference between the results in each case is minimal.

Figure 5.6 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 8%. The final average precisions for updating and the folding-up methods are not significantly different than that of recomputing ($p = 0.2488$ for updating, $p = 0.6048$ for folding-up method 1, and $p = 0.6467$ for folding-up method 2). See Table 5.1 for a comparison of the computation times.

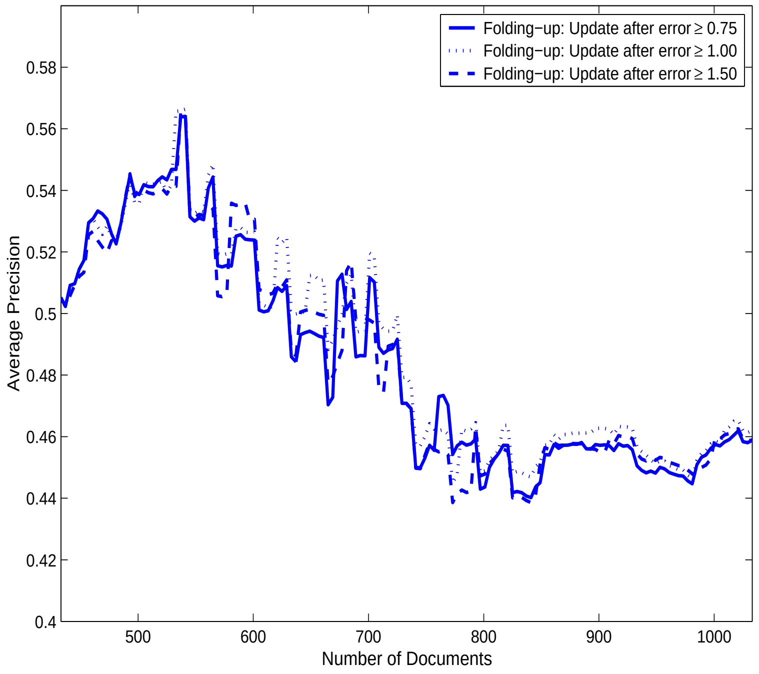

Figure 5.4: Comparison of average precisions of folding-up method 1 for the Medline collection with 600 documents added in 150 groups of 4.

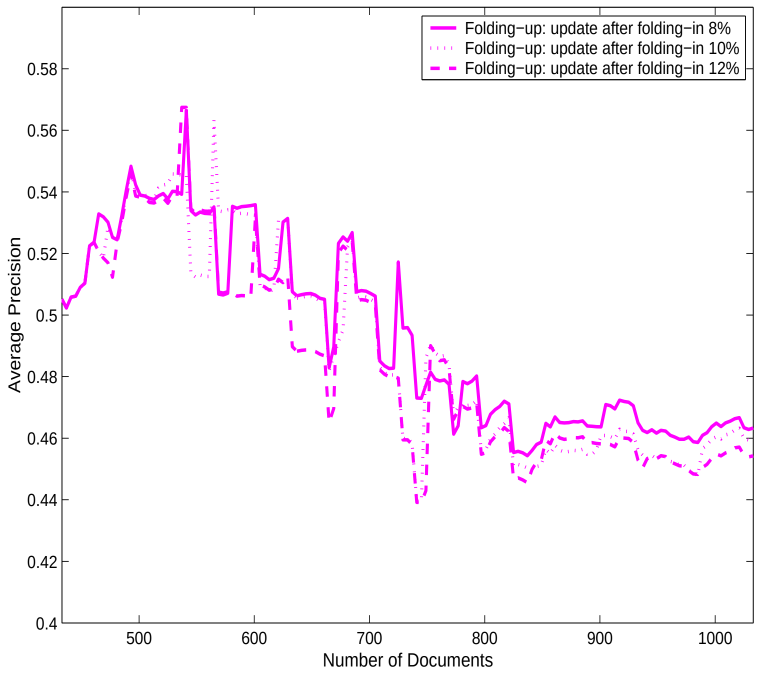

Figure 5.5: Comparison of average precisions of folding-up method 2 for the Medline collection with 600 documents added in 150 groups of 4.

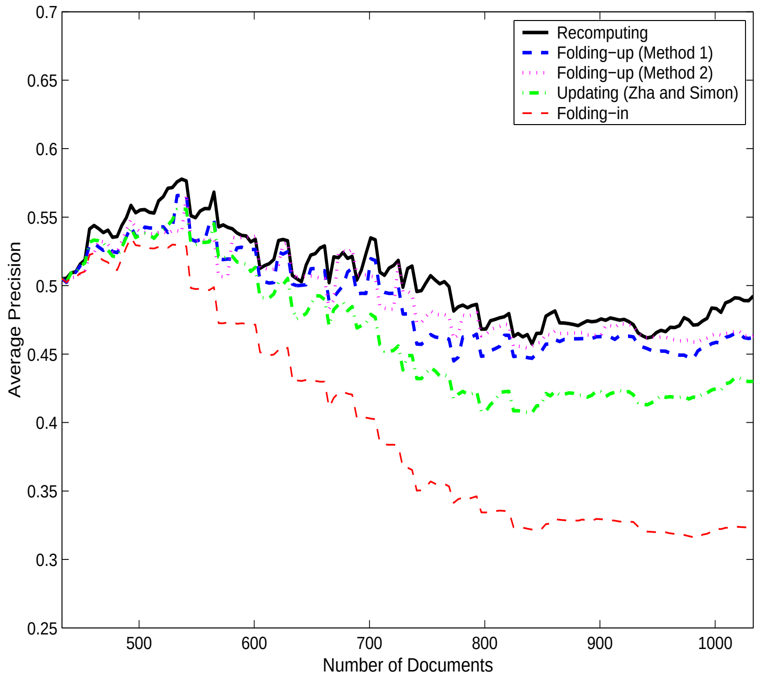

Figure 5.6: Comparison of average precisions for the Medline collection with 600 documents added in 150 groups of 4.

5.1.3 Medline: Experiment 3

In this experiment, the Medline term-document matrix is partitioned into an initial matrix of 433 documents and 100 submatrices of 6 documents each. Thus, there are 100 additions of 6 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.1.1 and 5.1.2, the number of columns in the initial term-document matrix more than doubles as the submatrices are added incrementally. Figure 5.7 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.1 and 5.4, this figure illustrates that this range of error thresholds produces similar results. This is an indication that the error threshold does not have to be finely tuned. Figure 5.2 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Again we see that the given range of percentages produces similar results.

Figure 5.9 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 again uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 8%. The methods of updating and folding-up once again out perform folding-in, with the folding-up methods giving higher average precisions than updating. The final average precision for folding-in is again significantly different from that of recomputing the PSVD ($p = 0.0093$). Updating and folding up (methods 1 and 2) again have final average precisions not significantly different than that of recomputing ($p = 0.2612$ for updating, $p = 0.6681$ for folding-up method 1, and $p = 0.7007$ for folding-up method 2). In this experiment, as with those in Sections 5.1.1 and 5.1.2, the two folding up methods produce very similar results, although here folding-up method 2 gives slightly higher average precisions than does folding-up method 1. As expected, folding-up method 2 has a slightly faster computation time than folding-up method 1. It is interesting to note that in this case, the folding-up methods are approximately 170 times faster than recomputing and more than twice as fast as updating. A comparison of all the computation times is found in Table 5.1.

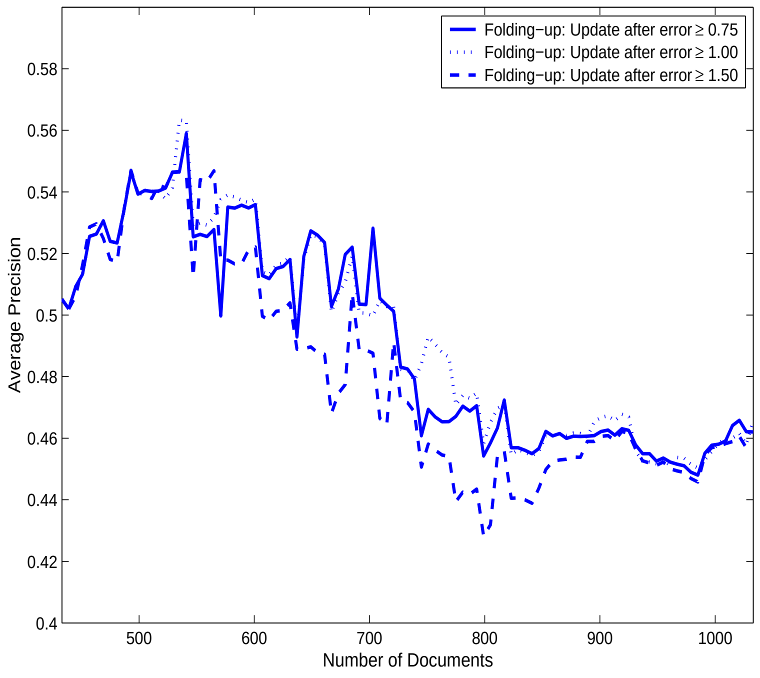

Figure 5.7: Comparison of average precisions of folding-up method 1 for the Medline collection with 600 documents added in 100 groups of 6.

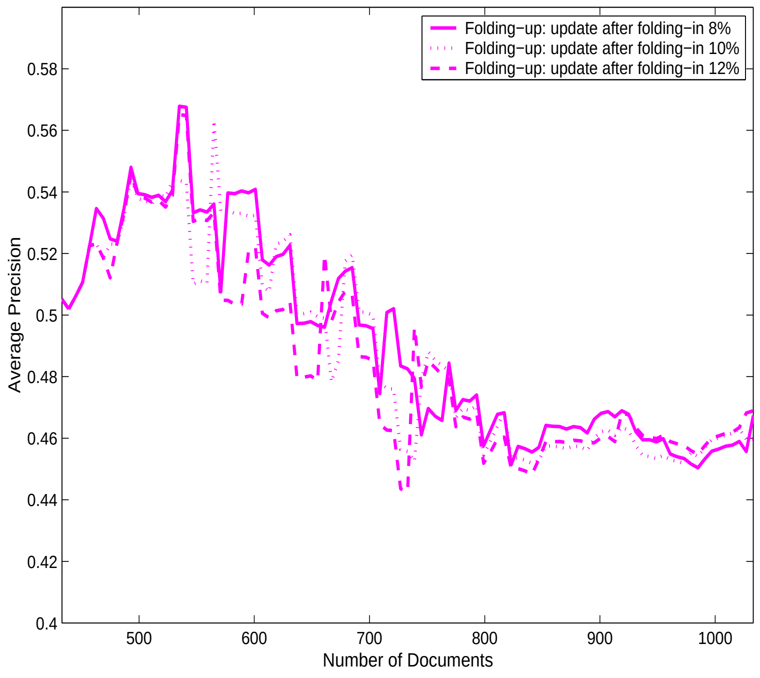

Figure 5.8: Comparison of average precisions of folding-up method 2 for the Medline collection with 600 documents added in 100 groups of 6.

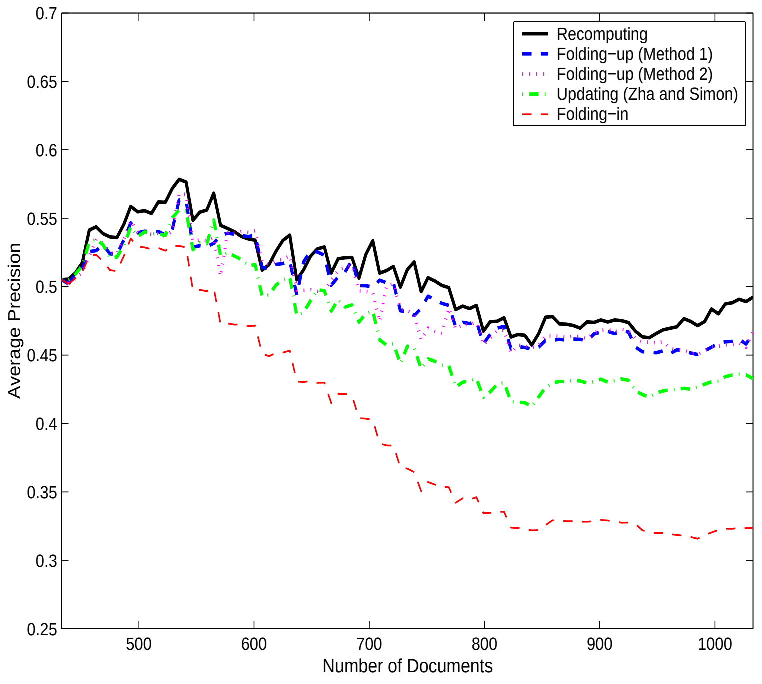

Figure 5.9: Comparison of average precisions for the Medline collection with 600 documents added in 100 groups of 6.

5.1.4 Medline: Experiment 4

The Medline term-document matrix is partitioned into an initial matrix of 433 documents and 60 submatrices of 10 documents each. Figure 5.11 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. Figure 5.10 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at 8%, 10%, and 12%. Figure 5.12 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 8%. The average precision for updating and folding-up, methods 1 and 2, is again much higher than that of folding-in, with final average precisions not significantly different than that of recomputing; $p = 0.3750$ for updating, $p = 0.6048$ for folding-up method 1, and $p = 0.4508$ for folding-up method 2. See Table 5.1 for a comparison of the computation times.

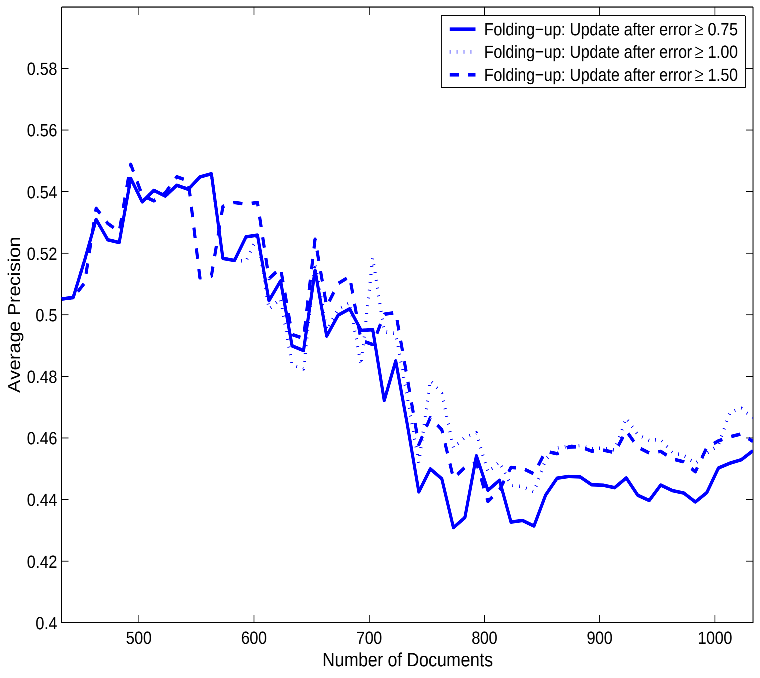

Figure 5.10: Comparison of average precisions of folding-up method 1 for the Medline collection with 600 documents added in 60 groups of 10.

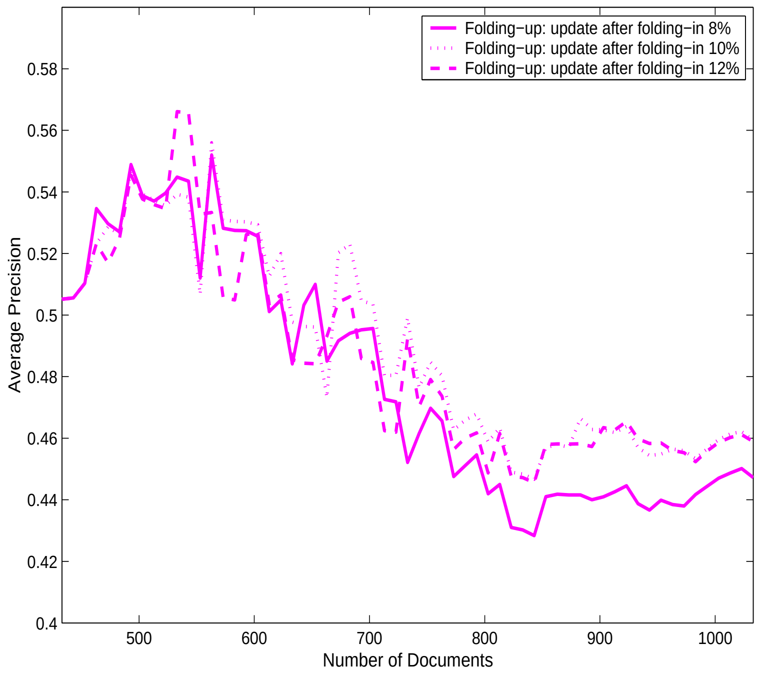

Figure 5.11: Comparison of average precisions of folding-up method 2 for the Medline collection with 600 documents added in 60 groups of 10.

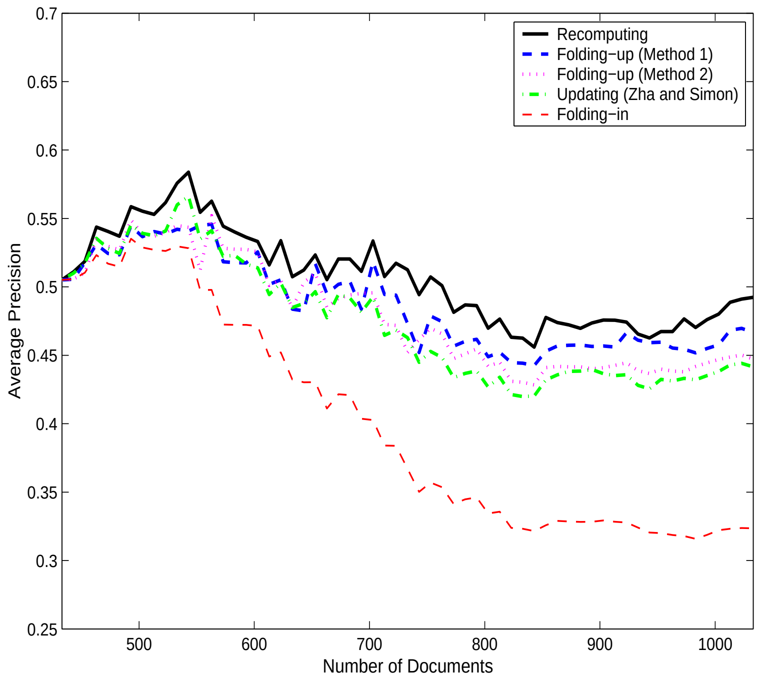

Figure 5.12: Comparison of average precisions for the Medline collection with 600 documents added in 60 groups of 10.

5.1.5 Medline: Experiment 5

In this experiment, the Medline term-document matrix is partitioned into an initial matrix of 433 documents and 30 submatrices of 20 documents each. This allows 30 additions of 20 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.1.1 and 5.1.4, the number of columns in the initial term-document matrix more than doubles as the submatrices are added incrementally. Figure 5.13 shows a comparison for three tests of folding-up method 1, in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.1 – 5.10, this figure illustrates that this range of error thresholds produces similar results, despite the fact that the number of documents being added at each increment has increase tenfold over these experiments. Figure 5.14 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Again we see that the given range of percentages produces similar results. This is an indication the percentage of documents (relative to the term-document matrix) that are folded-in before an update can fall within a range of values.

Figure 5.15 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 8%. Figure 5.15 shows that the average precision for folding-in deteriorates rapidly relative to recomputing the PSVD. The average precision for updating and folding-up, methods 1 and 2, is again much higher than that of folding-in, with final average precisions not significantly different than that of recomputing; $p = 0.4688$ for updating, $p = 0.5642$ for folding-up method 1, and $p = 0.5444$ for folding-up method 2. In this experiment, as with those in Sections 5.1.1 – 5.1.4, the two folding up methods produce very similar results. See Table 5.1 for a comparison of the computation times.

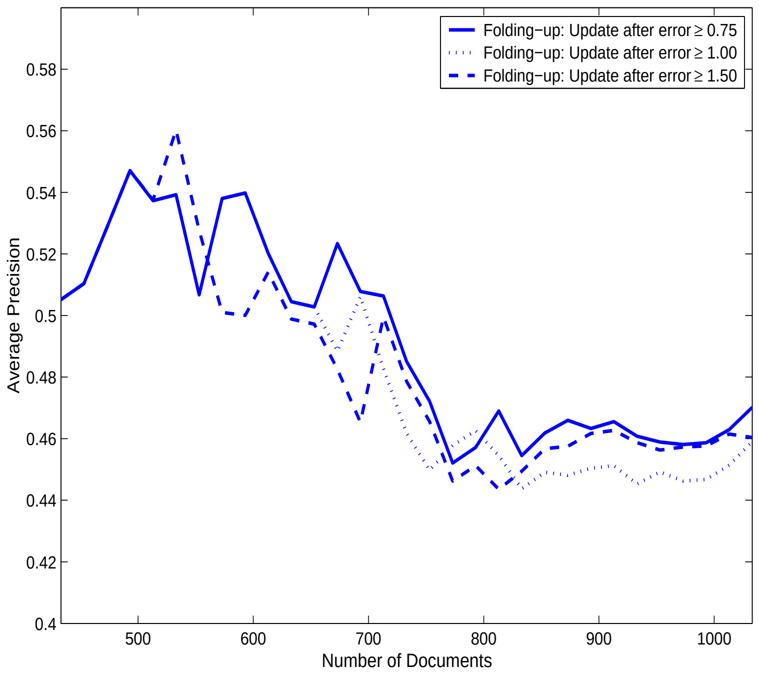

Figure 5.13: Comparison of average precisions of folding-up method 1 for the Medline collection with 600 documents added in 30 groups of 20.

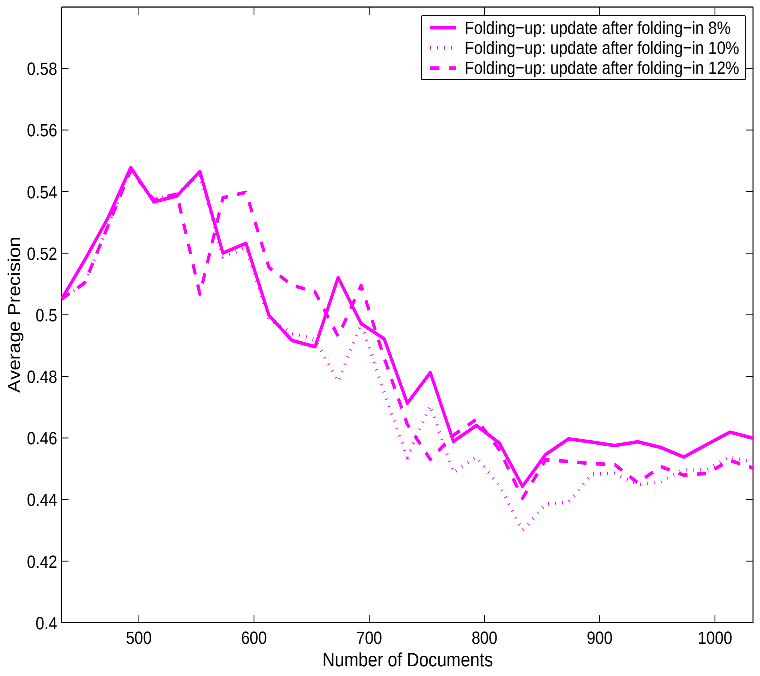

Figure 5.14: Comparison of average precisions of folding-up method 2 for the Medline collection with 600 documents added in 30 groups of 20.

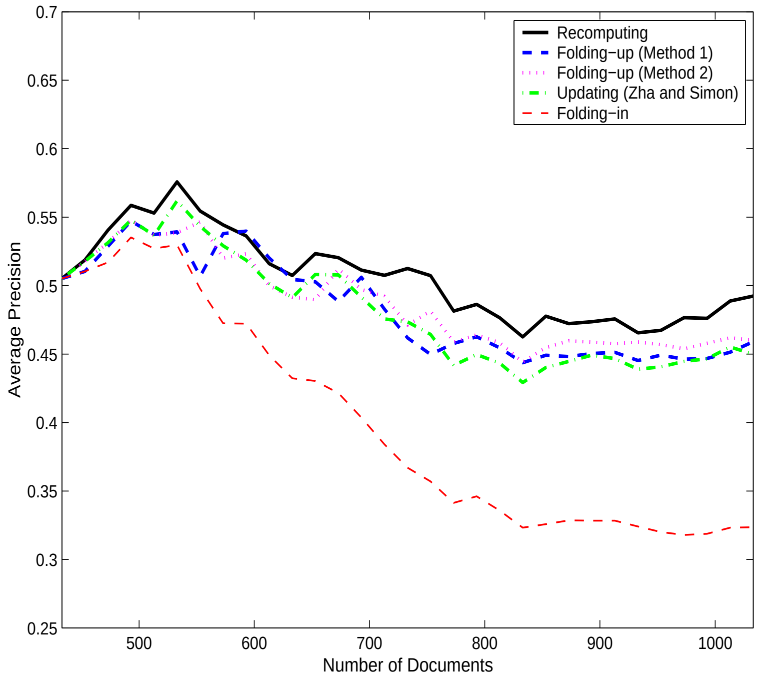

Figure 5.15: Comparison of average precisions for the Medline collection with 600 documents added in 30 groups of 20.

Method	CPU time	CPU time	CPU time	CPU time	CPU time
	Groups of 2	Groups of 4	Groups of 6	Groups of 10	Groups of 20
Recomputing	6551.0	3371.6	2260.8	1344.8	699.0
Updating	87.8	44.7	31.9	20.9	13.8
Folding-up 1	16.3	13.3	13.3	12.0	12.3
Folding-up 2	13.2	11.5	12.7	11.1	11.1
Folding-in	3.3	1.8	1.3	0.9	0.6

Table 5.1: Comparison of total CPU times (seconds) for the Medline collection with 600 documents added in groups of 2, 4, 6, 10, and 20.

For the case in which documents are added in groups of 2 in Table 5.1, the folding-up methods are more than 400 times faster than recomputing, and more than five times faster than updating. For the case in which documents are added in groups of 20, the folding-up methods are more than 50 times faster than recomputing.

5.2 Cranfield Text Collection

The Cranfield text collection contains 1398 aerospace systems abstracts and 225 queries. After stopword removal and stemming, there are 4563 terms, giving a 4563×1398 term-document matrix. This matrix is partitioned into an initial matrix with 598 documents and a number of submatrices to be appended incrementally. For each of the experiments with the Cranfield collection, $k = 300$, where k is the number of dimensions used in computing the PSVD. Table 5.2 gives a comparison of CPU times for each of the experiments.

5.2.1 Cranfield: Experiment 1

In this experiment, the Cranfield term-document matrix is partitioned into an initial matrix of 598 documents and 400 submatrices of 2 documents each. Thus, there are 400 additions of 2 documents to the term-document matrix, more than doubling the size of the initial matrix. Figure 5.16 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. The difference between the results is so slight that it is difficult to distinguish between the individual lines in the graph. Figure 5.17 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. The difference between the results is slight, which again is an indication of the robustness of the method.

Figure 5.18 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The average precision for updating and folding-up, methods 1 and 2, is higher than that of folding-in, and the final average precision in each case is not significantly different than that of recomputing ($p = 0.9754$ for updating, $p = 0.9529$ for folding-up method 1, and $p = 0.9520$ for folding-up method 2). It is interesting to note that in this experiment, folding-up method 1 is more than 500 times faster than recomputing and more than two and one half times faster than updating. Folding-up method 2 is more than 3800 times faster than recomputing in this case, and more than 20 times faster than updating.

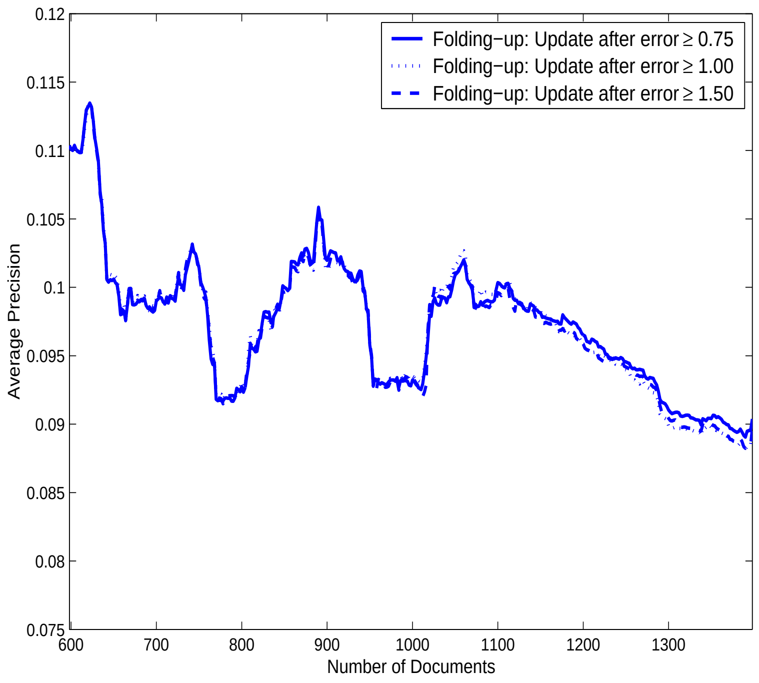

Figure 5.16: Comparison of average precisions of folding-up method 1 for the Cranfield collection with 800 documents added in 400 groups of 2.

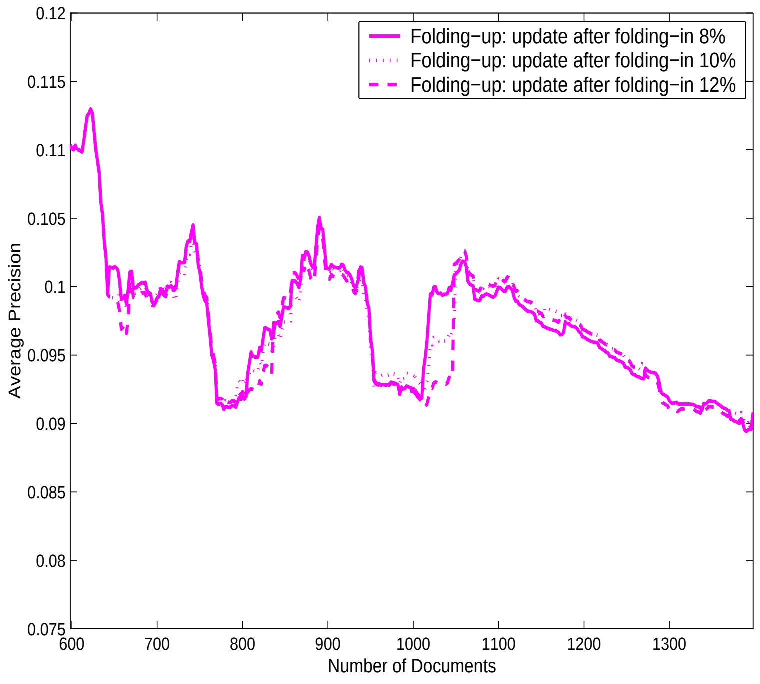

Figure 5.17: Comparison of average precisions of folding-up method 2 for the Cranfield collection with 800 documents added in 400 groups of 2.

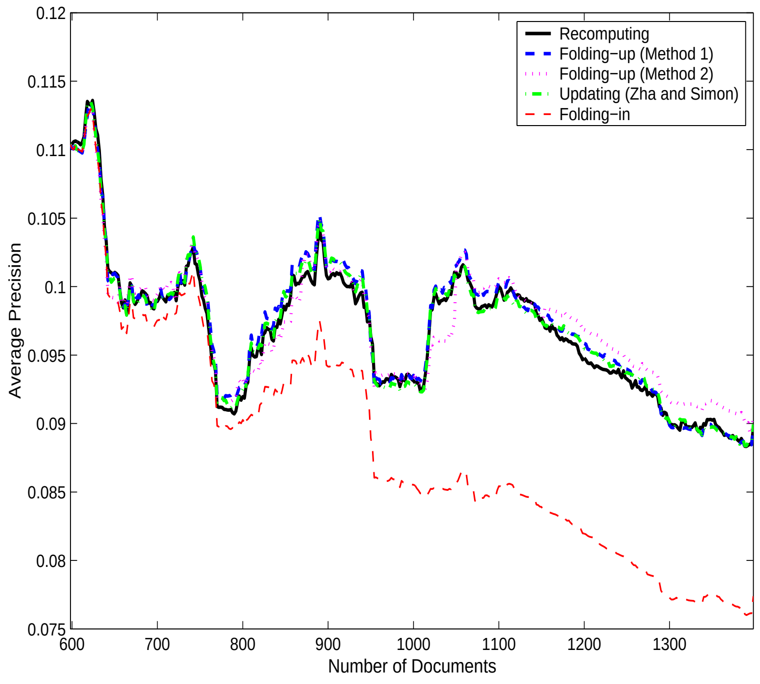

Figure 5.18: Comparison of average precisions for the Cranfield collection with 800 documents added in 400 groups of 2.

5.2.2 Cranfield: Experiment 2

The Cranfield term-document matrix is partitioned into an initial matrix of 598 documents and 200 submatrices of 4 documents each. Figure 5.19, shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50; Figure 5.17 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at 8%, 10%, and 12%. The difference between the results in each case is minimal. Figure 5.21 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The final average precisions for updating and folding-up, methods 1 and 2, are not significantly different than that of recomputing ($p = 0.9783$ for updating, $p = 0.9769$ for folding-up method 1, and $p = 0.9942$ for folding-up method 2). See Table 5.2 for a comparison of the computation times.

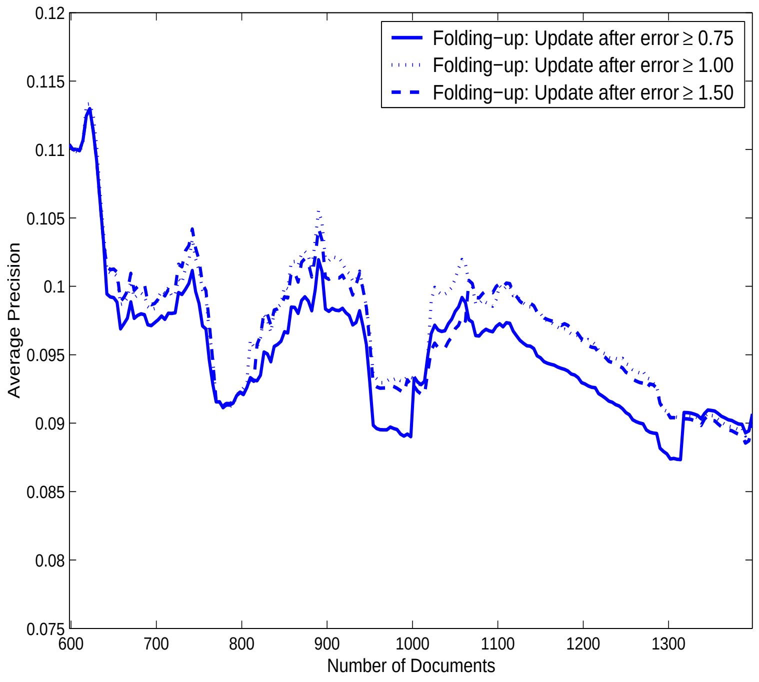

Figure 5.19: Comparison of average precisions of folding-up method 1 for the Cranfield collection with 800 documents added in 200 groups of 4.

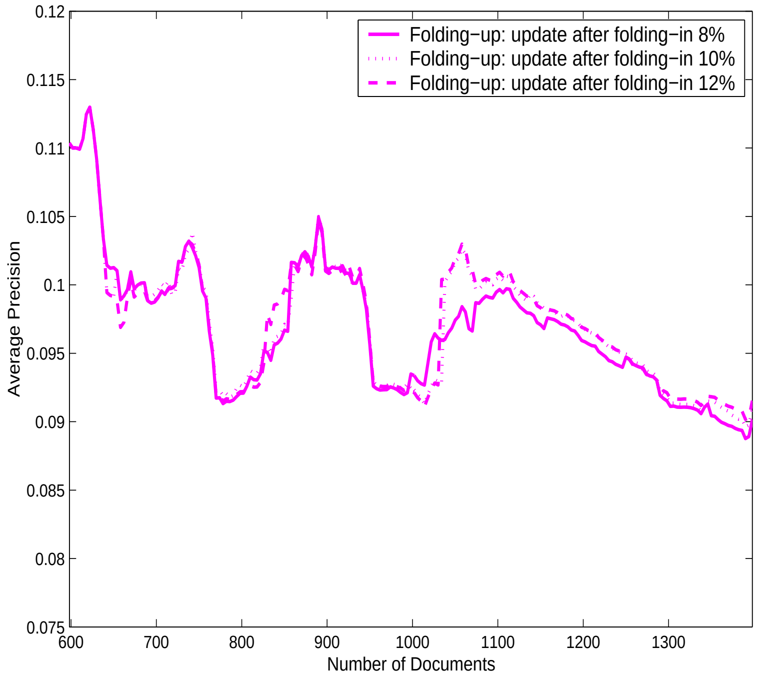

Figure 5.20: Comparison of average precisions of folding-up method 2 for the Cranfield collection with 800 documents added in 200 groups of 4.

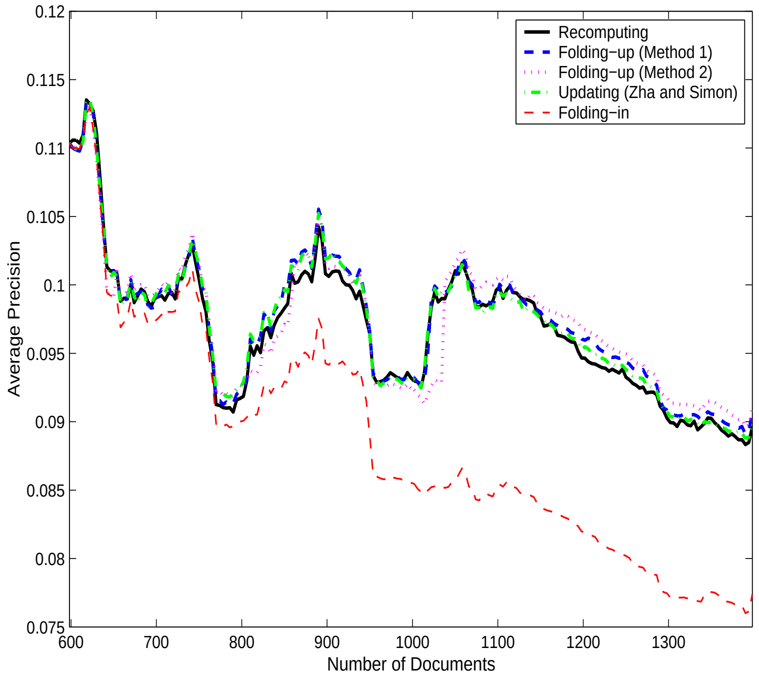

Figure 5.21: Comparison of average precisions for the CISI collection with 800 documents added in 200 groups of 4.

5.2.3 Cranfield: Experiment 3

In this experiment, the Cranfield term-document matrix is partitioned into an initial matrix of 598 documents and 100 submatrices of 8 documents each. Thus, there are 100 additions of 8 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.2.1 and 5.2.2, the number of columns in the initial term-document matrix more than doubles as the submatrices are added incrementally. Figure 5.22 shows a comparison for three tests of folding-up method 1, in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.16 and 5.19, this figure illustrates that this range of error thresholds produces very similar results. Again this is an indication that the error threshold does not have to be finely tuned. Figure 5.23 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. As with Figures 5.17 and 5.20, this figure shows that the range of percentages produces similar retrieval performance. This is an indication of the robustness of folding-up method 2.

Figure 5.24 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The methods of updating and folding-up once again out perform folding-in. The average precisions for recomputing, updating, and the folding-up methods are all very similar. Updating and folding up (methods 1 and 2) again have final average precisions not significantly different than that of recomputing ($p = 0.9633$ for updating, $p = 0.9615$ for folding-up method 1, and $p = 0.9997$ for folding-up method 2). As expected, the folding-up methods are faster than either recomputing or updating, and folding-up method 2 has a slightly faster computation time than folding-up method 1. In this case, folding-up method 1 is more than 380 times faster than recomputing and more than twice as fast as updating; folding-up method 2 is almost 1150 times faster than recomputing and more than six times faster than updating. A comparison of all the computation times is found in Table 5.3.

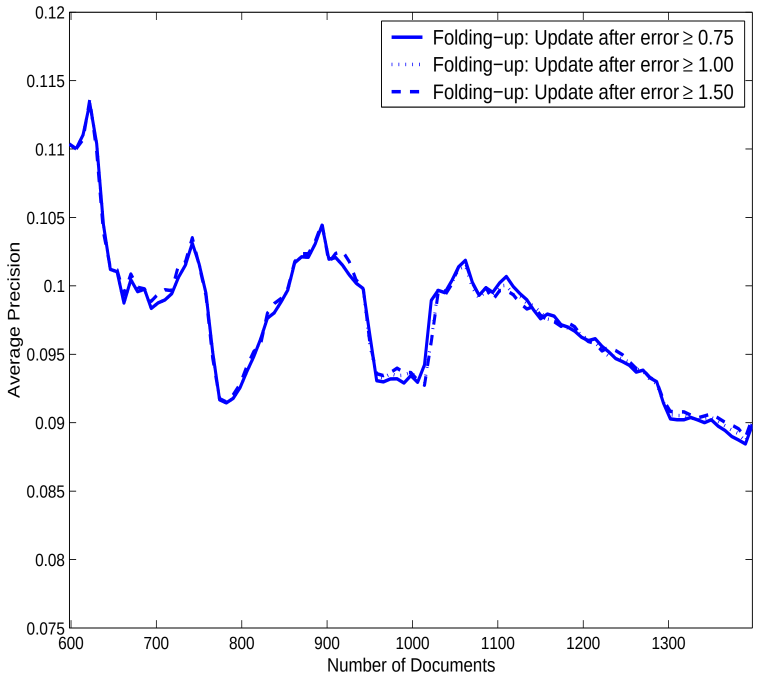

Figure 5.22: Comparison of average precisions of folding-up method 1 for the Cranfield collection with 800 documents added in 100 groups of 8.

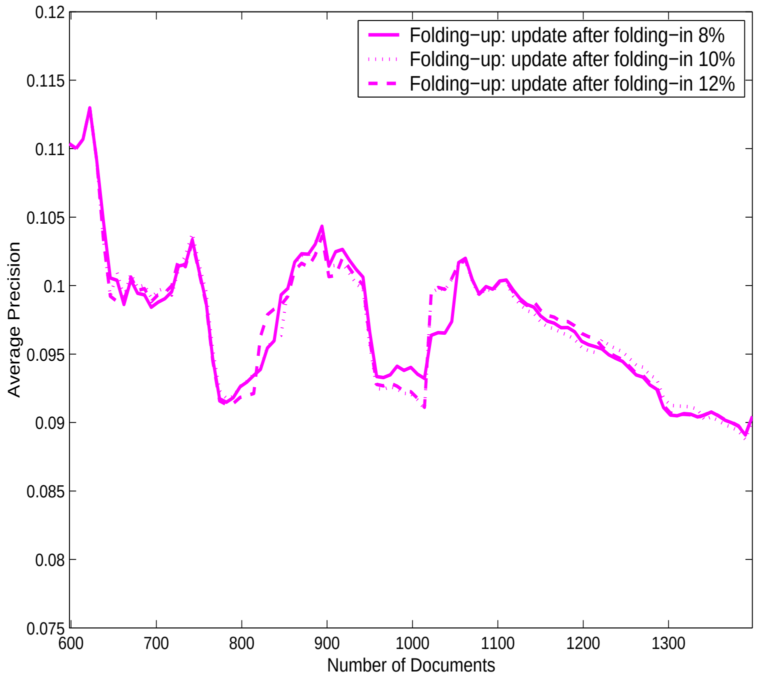

Figure 5.23: Comparison of average precisions of folding-up method 2 for the Cranfield collection with 800 documents added in 100 groups of 8.

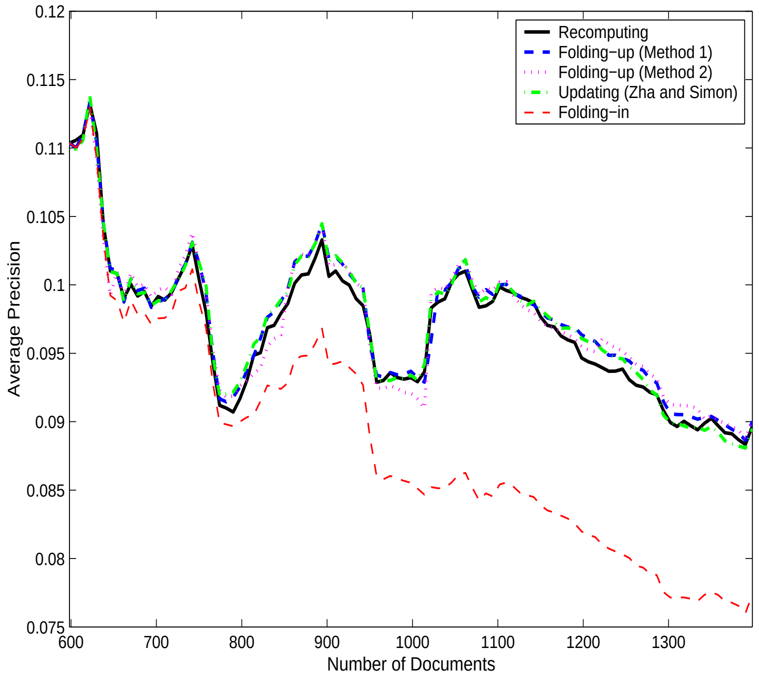

Figure 5.24: Comparison of average precisions for the Cranfield collection with 800 documents added in 100 groups of 8.

5.2.4 Cranfield: Experiment 4

Here the Cranfield term-document matrix is partitioned into an initial matrix of 598 documents and 80 submatrices of 10 documents each. Figure 5.26 shows a comparison for three tests of folding-up method 1, in which the error threshold is set at 0.75, 1.00, and 1.50. Figure 5.25 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Figure 5.27 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 again uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The final average precisions for updating and folding-up, methods 1 and 2, are not significantly different than that of recomputing; $p = 0.9584$ for updating, $p = 0.9905$ for folding-up method 1, and $p = 0.9853$ for folding-up method 2. See Table 5.2 for a comparison of the computation times.

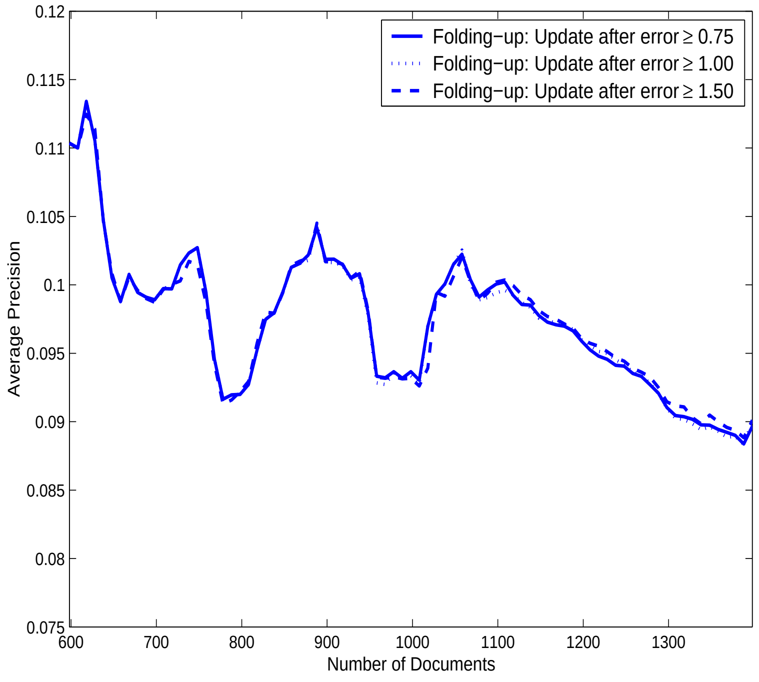

Figure 5.25: Comparison of average precisions of folding-up method 1 for the Cranfield collection with 800 documents added in 80 groups of 10.

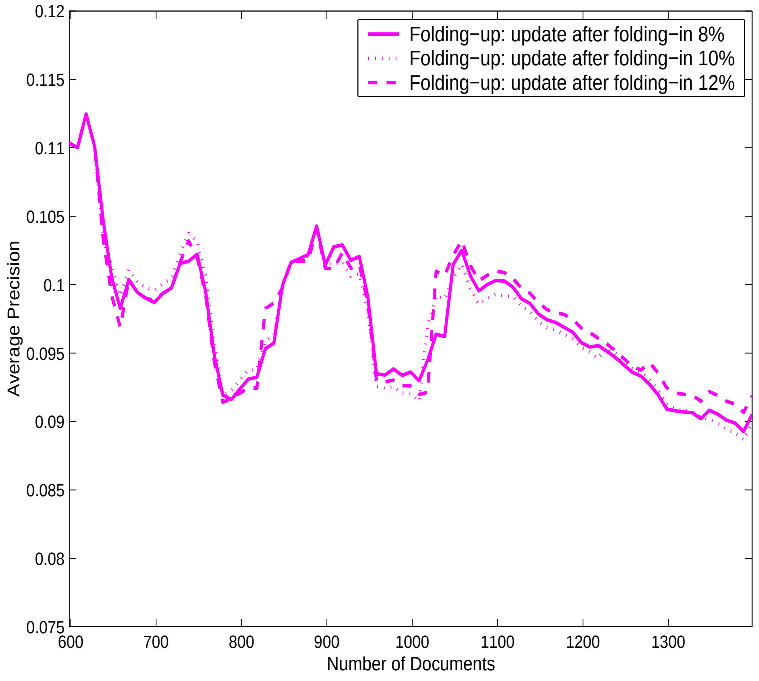

Figure 5.26: Comparison of average precisions of folding-up method 2 for the Cranfield collection with 800 documents added in 80 groups of 10.

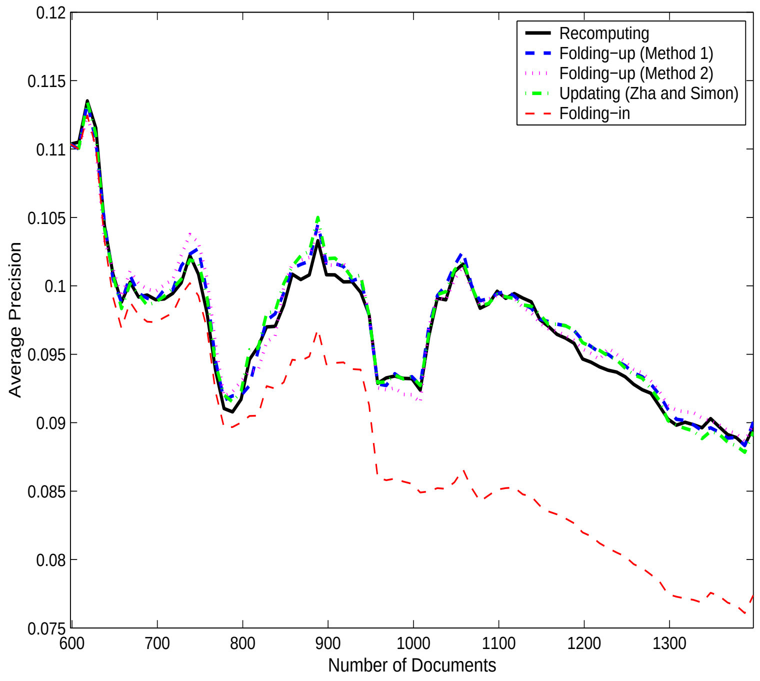

Figure 5.27: Comparison of average precisions for the Cranfield collection with 800 documents added in 80 groups of 10.

5.2.5 Cranfield: Experiment 5

In this experiment, the Cranfield term-document matrix is partitioned into an initial matrix of 598 documents and 40 submatrices of 20 documents each. This allows 40 additions of 20 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.2.1 and 5.2.4, the number of columns in the initial term-document matrix more than doubles as the submatrices are added incrementally. Figure 5.28 shows a comparison for three tests of folding-up method 1, in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.16 – 5.25, this figure illustrates that this range of error thresholds produces similar results, despite the fact that the number of documents being added at each increment has increase tenfold over these experiments. Figure 5.29 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Again we see that the given range of percentages produces similar results. This is an indication the percentage of documents (relative to the term-document matrix) that are folded-in before an update can fall within a range of values.

Figure 5.30 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. Figure 5.30 shows that the average precision for folding-in deteriorates relative to recomputing the PSVD. The final average precisions for updating and folding-up, methods 1 and 2, are again not significantly different than that of recomputing ($p = 0.9864$ for updating, $p = 0.9899$ for folding-up method 1, and $p = 0.9824$ for folding-up method 2). In this experiment, folding-up method 1 is 290 times faster than recomputing and more than twice as fast as updating. Folding-up method 2 is more than 450 times faster than recomputing, and more than three times faster than updating. See Table 5.2 for a comparison of all the computation times.

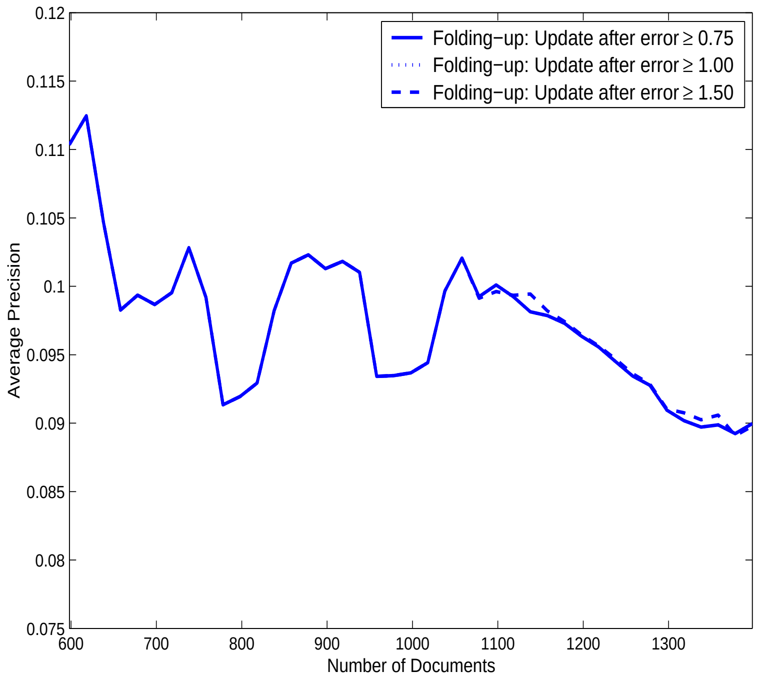

Figure 5.28: Comparison of average precisions of folding-up method 1 for the Cranfield collection with 800 documents added in 40 groups of 20.

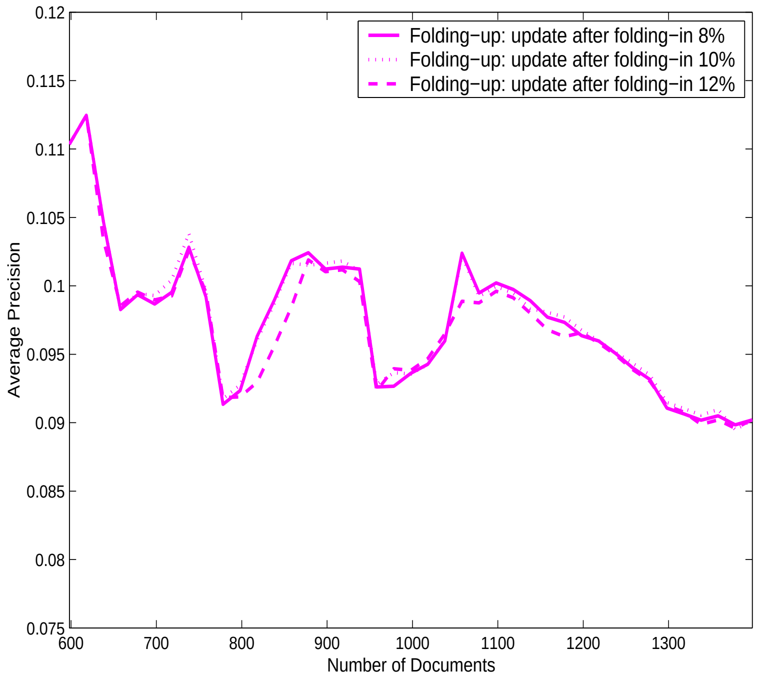

Figure 5.29: Comparison of average precisions of folding-up method 2 for the Cranfield collection with 800 documents added in 40 groups of 20.

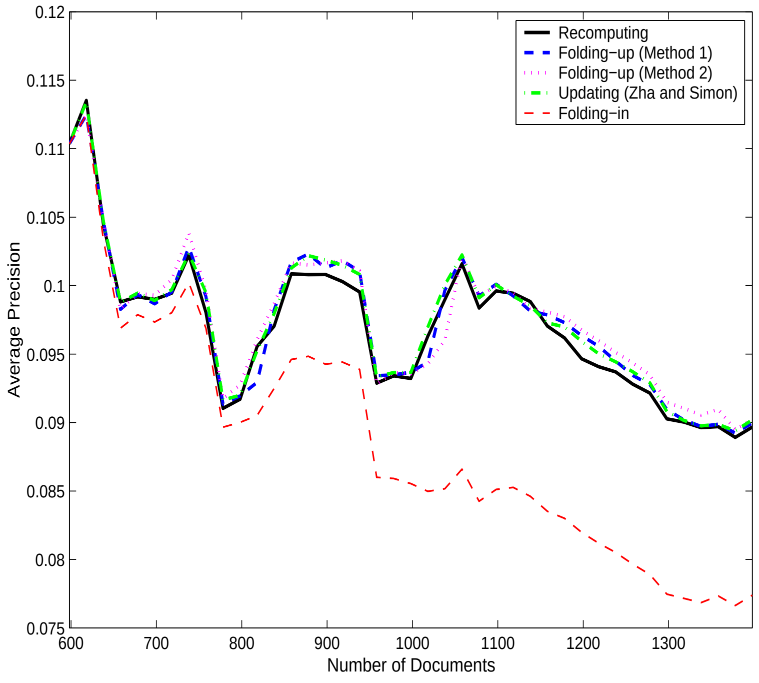

Figure 5.30: Comparison of average precisions for the Cranfield collection with 800 documents added in 40 groups of 20.

Method	CPU time	CPU time	CPU time	CPU time	CPU time
	Groups of 2	Groups of 4	Groups of 8	Groups of 10	Groups of 20
Recomputing	354556.3	179631.0	90863.4	73532.0	35828.3
Updating	1873.0	969.8	502.3	409.0	265.5
Folding-up 1	685.5	407.9	238.2	238.5	123.2
Folding-up 2	91.7	85.9	79.1	76.6	79.2
Folding-in	46.3	24.1	12.8	10.5	5.9

Table 5.2: Comparison of total CPU times (seconds) for the Cranfield collection with 800 documents added in groups of 2, 4, 8, 10, and 20.

In each case in Table 5.2, the folding-up methods are hundreds of times faster than recomputing. Both folding-up methods are also significantly faster than updating.

5.3 CISI Text Collection

The CISI text collection contains 1460 information retrieval abstracts and 35 queries. After stopword removal and stemming, there are 5544 terms, giving a 5544×1460 term-document matrix. This matrix is partitioned into an initial matrix with 460 documents and a number of submatrices to be appended incrementally. For each of the experiments with the CISI collection, $k = 150$, where k is the number of dimensions used in computing the PSVD. Table 5.3 gives a comparison of CPU times for each of the experiments.

5.3.1 CISI: Experiment 1

In this experiment, the CISI term-document matrix is partitioned into an initial matrix of 460 documents and 500 submatrices of 2 documents each. Thus, there are 500 additions of 2 documents to the term-document matrix, more than tripling the number of columns in the initial term-document matrix as the submatrices are add incrementally. Figure 5.31 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. Figure 5.32 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%.

Figure 5.33 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The final average precisions for updating and folding-up, methods 1 and 2 are not significantly different than that of recomputing ($p = 0.9018$ for updating, $p = 0.8925$ for folding-up method 1, and $p = 0.8740$ for folding-up method 2). In this case, folding-up method 1 is more than 650 times faster than recomputing and more than four and one half times faster than recomputing. Folding-up method 2 is more than 1700 times faster than recomputing and more than twelve times faster than updating. See Table 5.3 for a comparison of all the computation times.

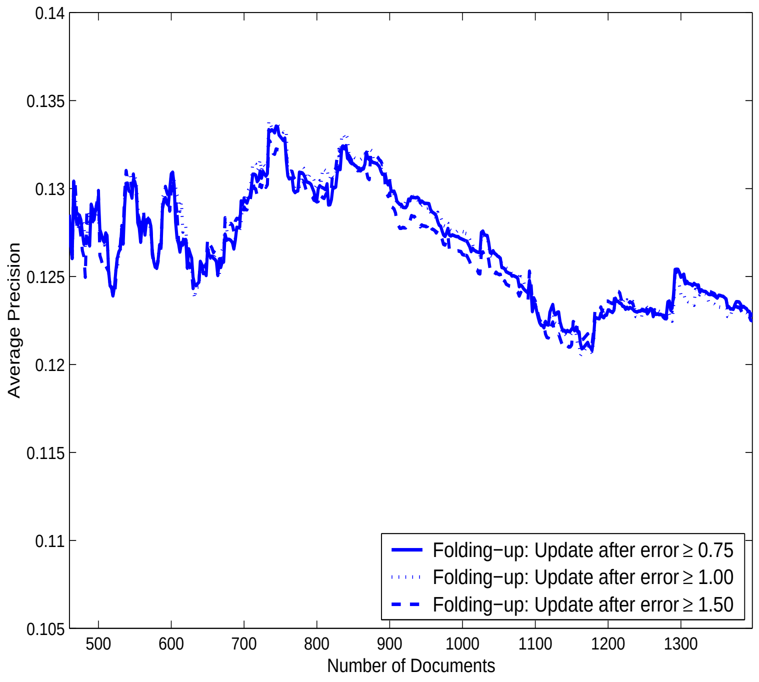

Figure 5.31: Comparison of average precisions of folding-up method 1 for the CISI collection with 1000 documents added in 500 groups of 2.

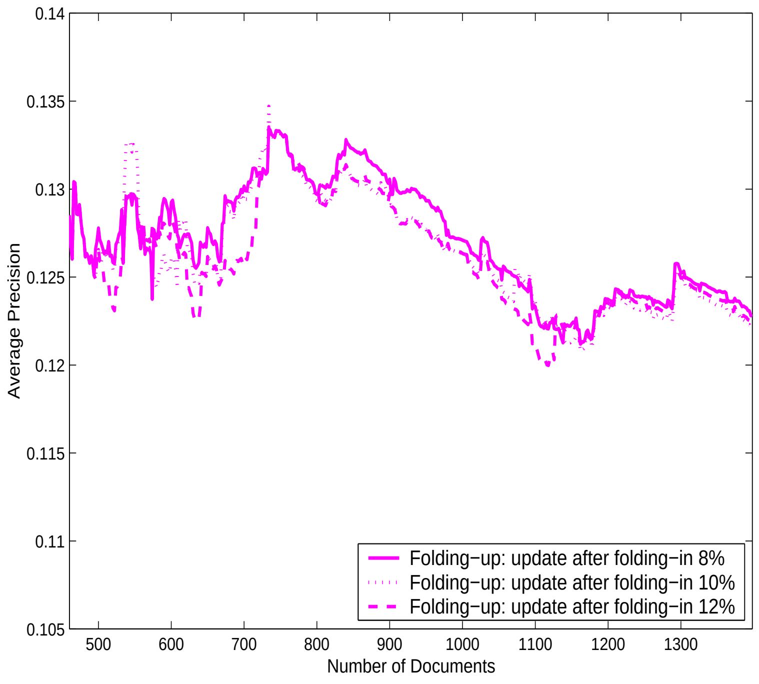

Figure 5.32: Comparison of average precisions of folding-up method 2 for the CISI collection with 1000 documents added in 500 groups of 2.

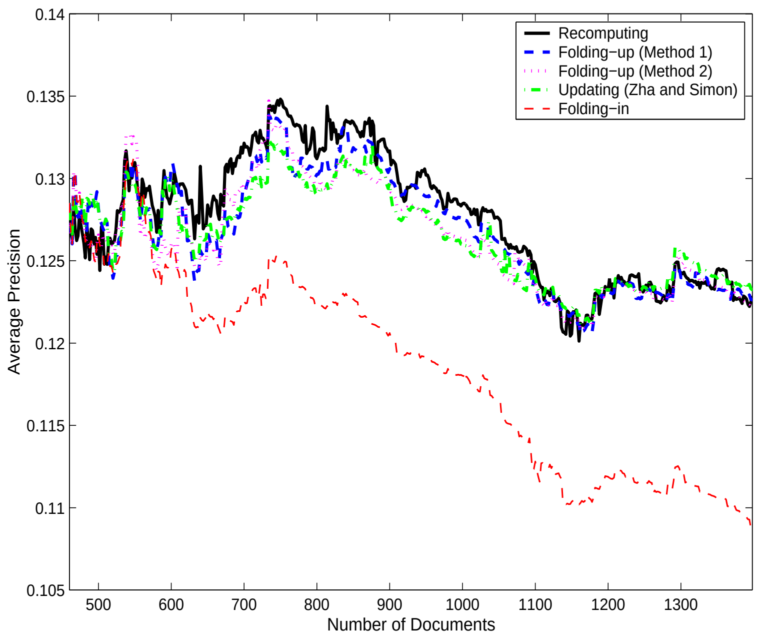

Figure 5.33: Comparison of average precisions for the CISI collection with 1000 documents added in 500 groups of 2.

5.3.2 CISI: Experiment 2

Here the CISI term-document matrix is partitioned into an initial matrix of 460 documents and 250 submatrices of 4 documents each. Figure 5.34, shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50; Figure 5.32 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. The difference between the results in each case is minimal. Figure 5.36 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The final average precisions for updating and folding-up, methods 1 and 2, are not significantly different than that of recomputing ($p = 0.9018$ for updating, $p = 0.9018$ for folding-up method 1, and $p = 0.8555$ for folding-up method 2). See Table 5.3 for a comparison of the computation times.

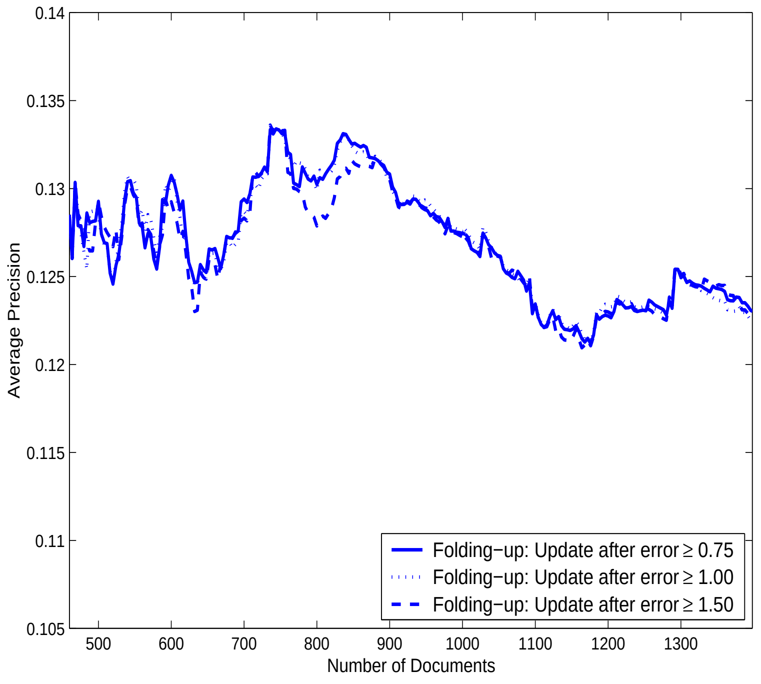

Figure 5.34: Comparison of average precisions of folding-up method 1 for the CISI collection with 1000 documents added in 250 groups of 4.

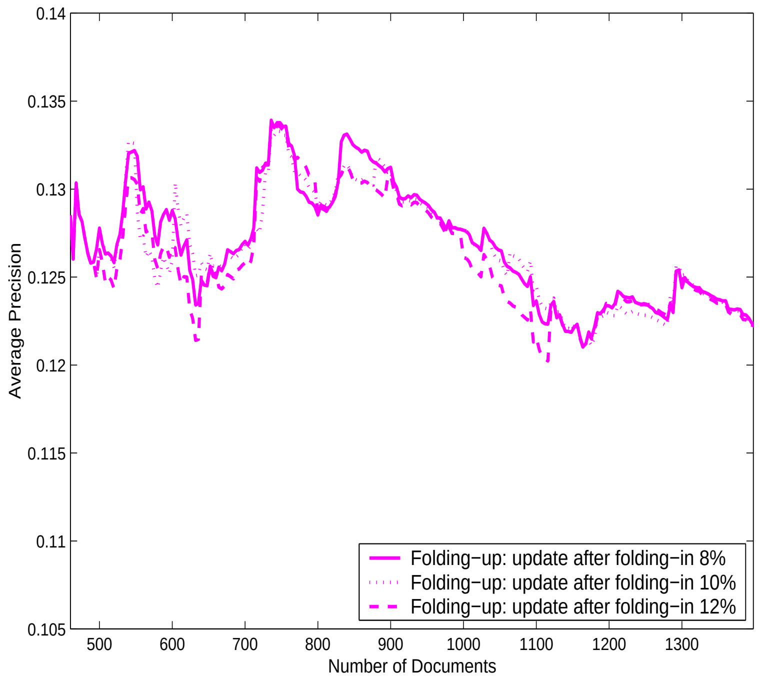

Figure 5.35: Comparison of average precisions of folding-up method 2 for the CISI collection with 1000 documents added in 250 groups of 4.

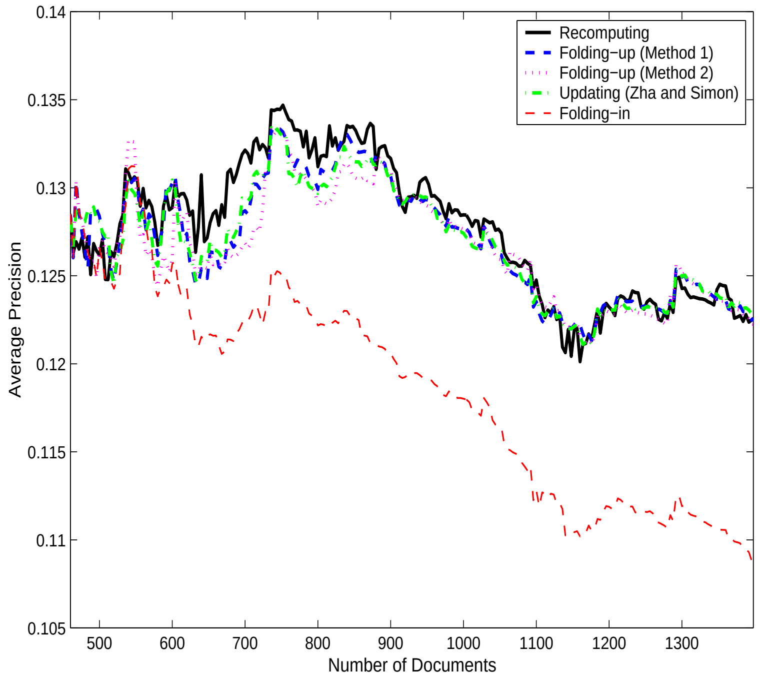

Figure 5.36: Comparison of average precisions for the CISI collection with 1000 documents added in 250 groups of 4.

5.3.3 CISI: Experiment 3

In this experiment, the CISI term-document matrix is partitioned into an initial matrix of 460 documents and 125 submatrices of 8 documents each. Thus, there are 125 additions of 8 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.3.1 and 5.3.2, the number of columns in the initial term-document matrix more than triples as the submatrices are added incrementally. Figure 5.37 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.31 and 5.34, this figure illustrates that this range of error thresholds produces very similar results. This is an indication that the error threshold does not have to be finely tuned. Figure 5.38 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at 8%, 10%, and 12%. As with Figures 5.32 and 5.35, this figure shows that the range of percentages produces similar retrieval performance. This is an indication of the robustness of folding-up method 2.

Figure 5.39 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The methods of updating and folding-up once again out perform folding-in. The average precisions for recomputing, updating, and the folding-up methods are all very similar. Updating and folding up (methods 1 and 2) again have final average precisions that are not significantly different than that of recomputing ($p = 0.8925$ for updating, $p = 0.8463$ for folding-up method 1, and $p = 0.8833$ for folding-up method 2). As expected, the folding-up methods are faster than either recomputing or updating, and folding-up method 2 has a slightly faster computation time than folding-up method 1. In this case, folding-up method 1 is more than 385 times faster than recomputing and more than two and one half times faster than updating. Folding-up method 2 is more than 580 times faster than recomputing and more than four times faster than updating. See Table 5.3 for a comparison of all the computation times.

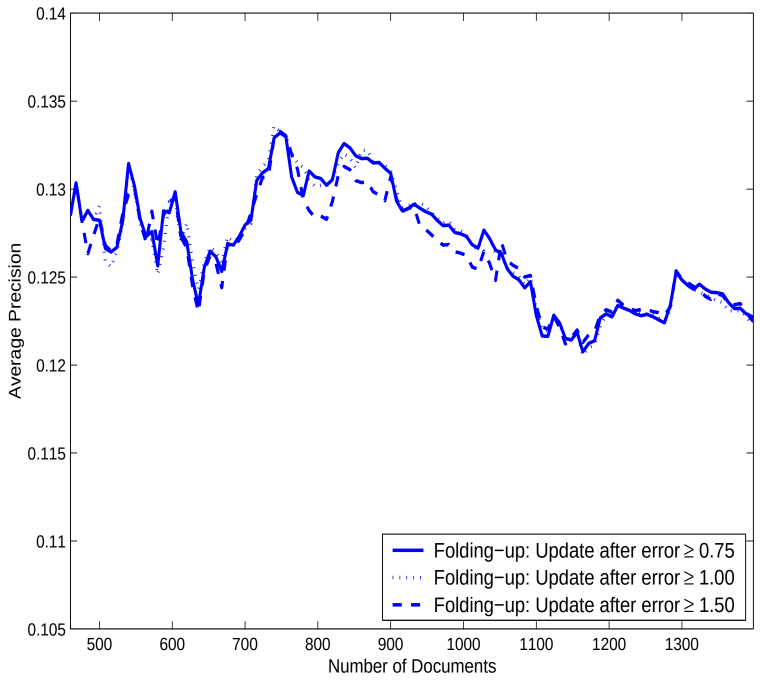

Figure 5.37: Comparison of average precisions of folding-up method 1 for the CISI collection with 1000 documents added in 125 groups of 8.

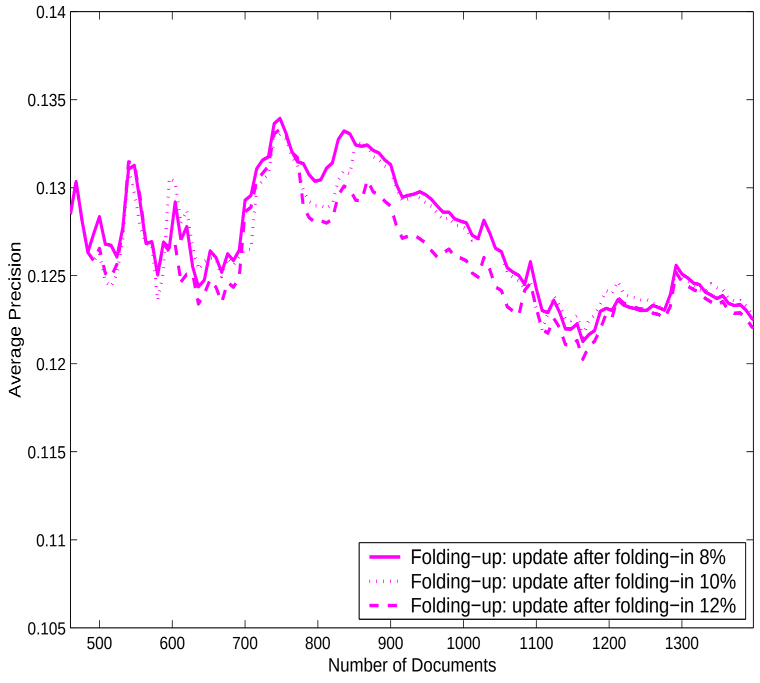

Figure 5.38: Comparison of average precisions of folding-up method 2 for the CISI collection with 1000 documents added in 125 groups of 8.

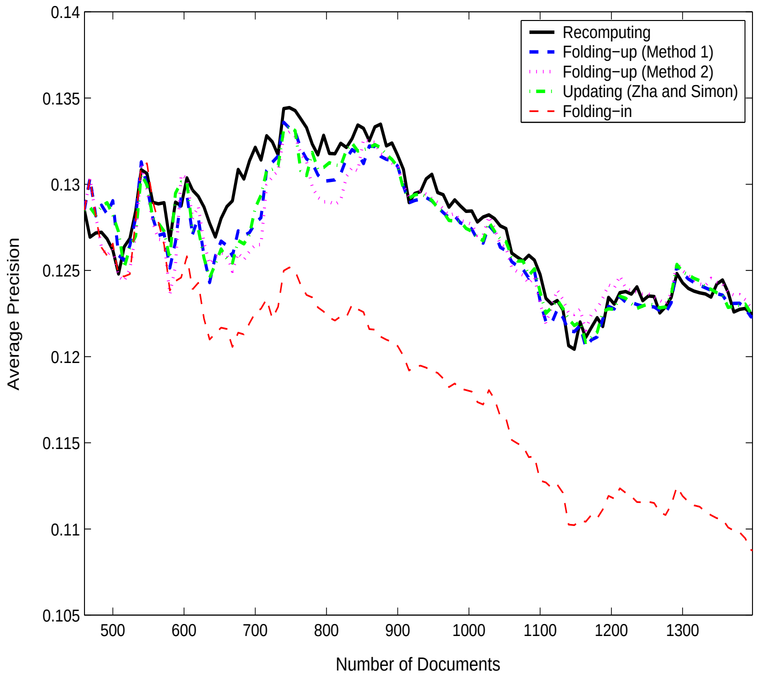

Figure 5.39: Comparison of average precisions for the CISI collection with 1000 documents added in 125 groups of 8.

5.3.4 CISI: Experiment 4

The CISI term-document matrix is partitioned into an initial matrix of 460 documents and 100 submatrices of 10 documents each. Figure 5.41 shows a comparison for three tests of folding-up method 1, in which the error threshold is set at 0.75, 1.00, and 1.50. Figure 5.40 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Figure 5.42 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. Note that in this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. The final average precisions for updating and folding-up, methods 1 and 2, are not significantly different than that of recomputing ($p = 0.9018$ for updating, $p = 0.8833$ for folding-up method 1, and $p = 0.8463$ for folding-up method 2). See Table 5.3 for a comparison of the computation times.

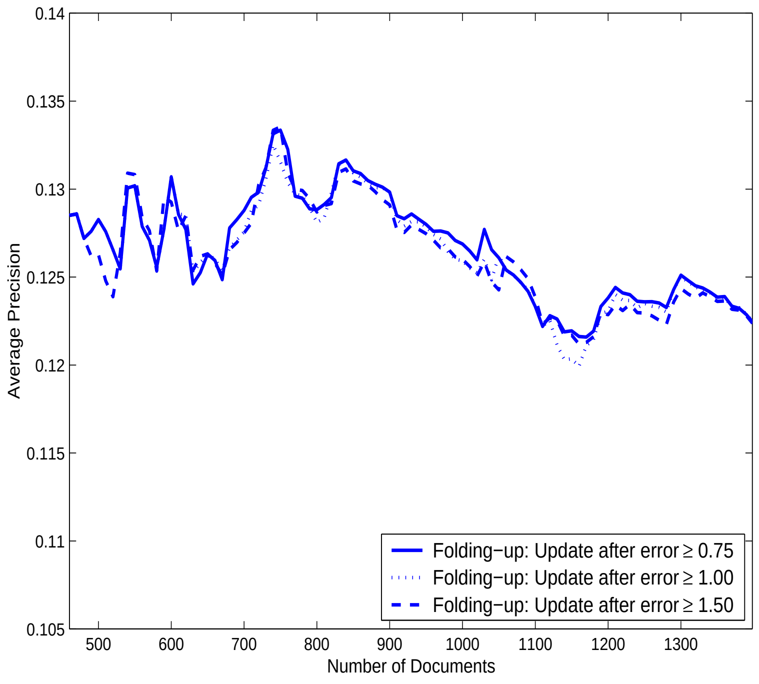

Figure 5.40: Comparison of average precisions of folding-up method 1 for the CISI collection with 1000 documents added in 100 groups of 10.

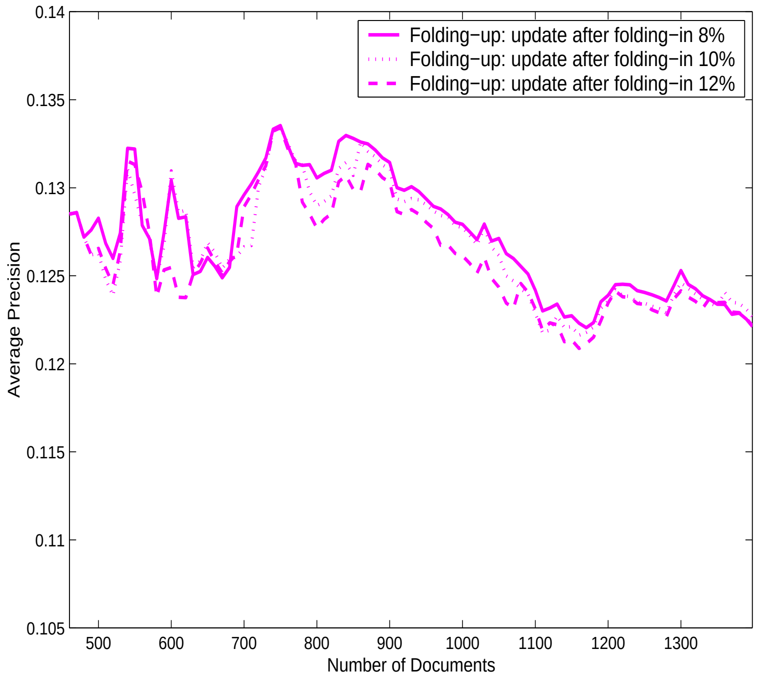

Figure 5.41: Comparison of average precisions of folding-up method 2 for the CISI collection with 1000 documents added in 100 groups of 10.

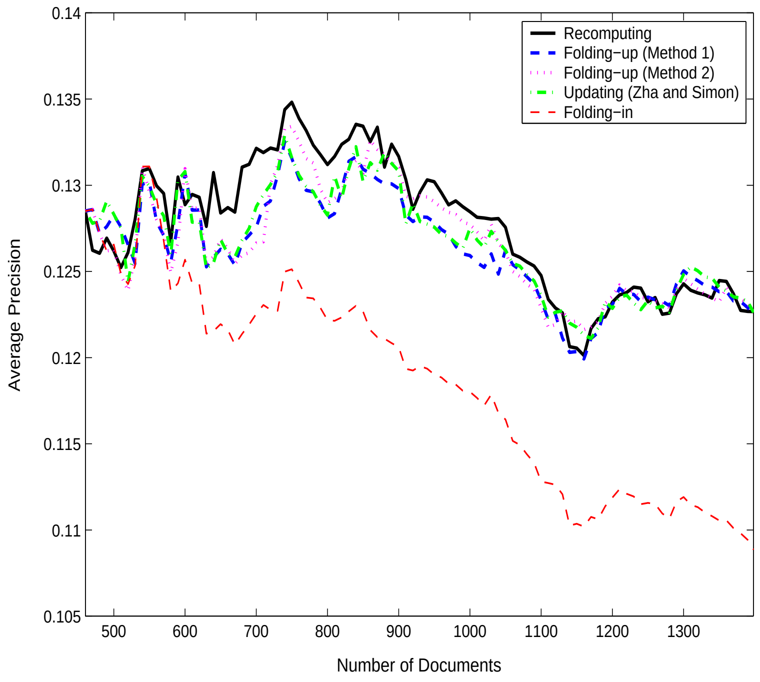

Figure 5.42: Comparison of average precisions for the CISI collection with 1000 documents added in 100 groups of 10.

5.3.5 CISI: Experiment 5

In this experiment, the CISI term-document matrix is partitioned into an initial matrix of 460 documents and 50 submatrices of 20 documents each. This allows 50 additions of 20 documents to the term-document matrix. Note that in this experiment, as those in Sections 5.3.1 and 5.3.4, the number of columns in the initial term-document matrix more than triples as the submatrices are add incrementally. Figure 5.43 shows a comparison for three tests of folding-up method 1 in which the error threshold is set at 0.75, 1.00, and 1.50. As with Figures 5.31 – 5.40, this figure illustrates that this range of error thresholds produces similar results, despite the fact that the number of documents being added at each increment has increase tenfold over these experiments. Figure 5.44 shows a comparison for three tests of folding-up method 2, in which the percentage of documents that can be folded-in before an update is performed is set at $8\%, 10\%$, and 12%. Again we see that the given range of percentages produces similar results. This is an indication the percentage of documents (relative to the term-document matrix) that are folded-in before an update can fall within a range of values.

Figure 5.45 compares the average precision for the methods of recomputing, folding-up (methods 1 and 2), PSVD-updating, and folding-in. In this figure, folding-up method 1 uses an error threshold of 1.00, and folding-up method 2 uses a percentage threshold of 10%. Figure 5.45 shows that the average precision for folding-in deteriorates rapidly relative to recomputing the PSVD. The final average precisions for updating and folding-up, methods 1 and 2, are not significantly different than that of recomputing ($p = 0.8925$ for updating, $p = 0.8833$ for folding-up method 1, and $p = 0.8555$ for folding-up method 2). In this experiment, the folding-up methods are more than 200 times faster than recomputing and more than one and a half times faster than updating. See Table 5.3 for a comparison of all the computation times.

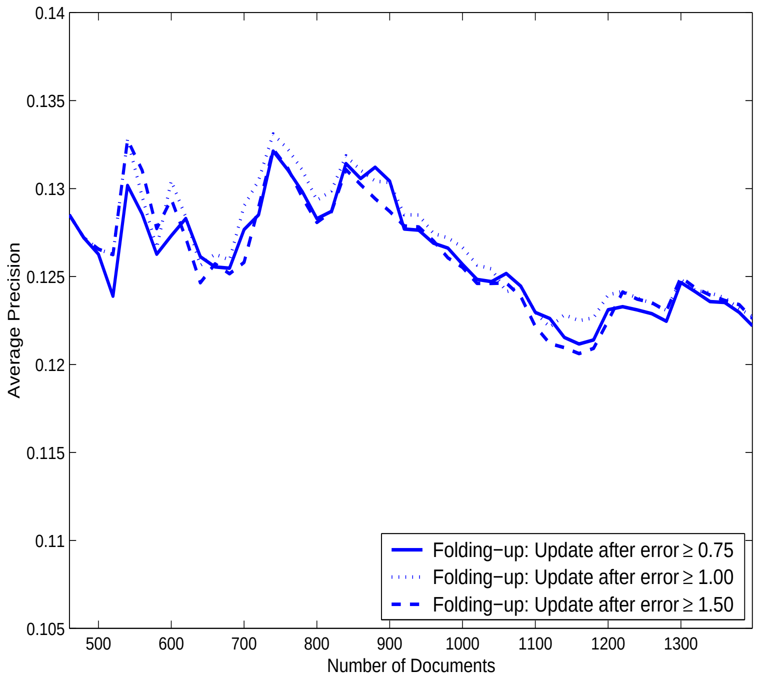

Figure 5.43: Comparison of average precisions of folding-up method 1 for the CISI collection with 1000 documents added in 50 groups of 20.

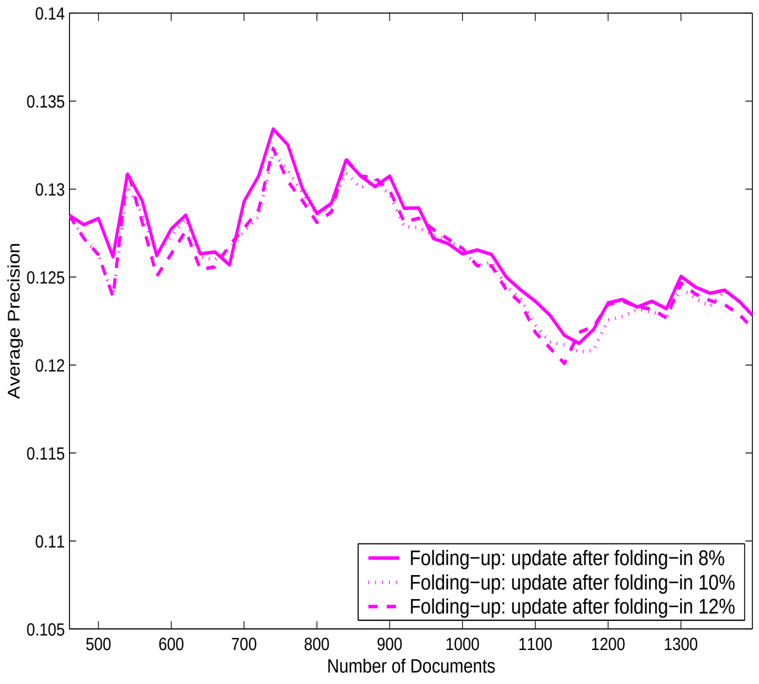

Figure 5.44: Comparison of average precisions of folding-up method 2 for the CISI collection with 1000 documents added in 50 groups of 20.

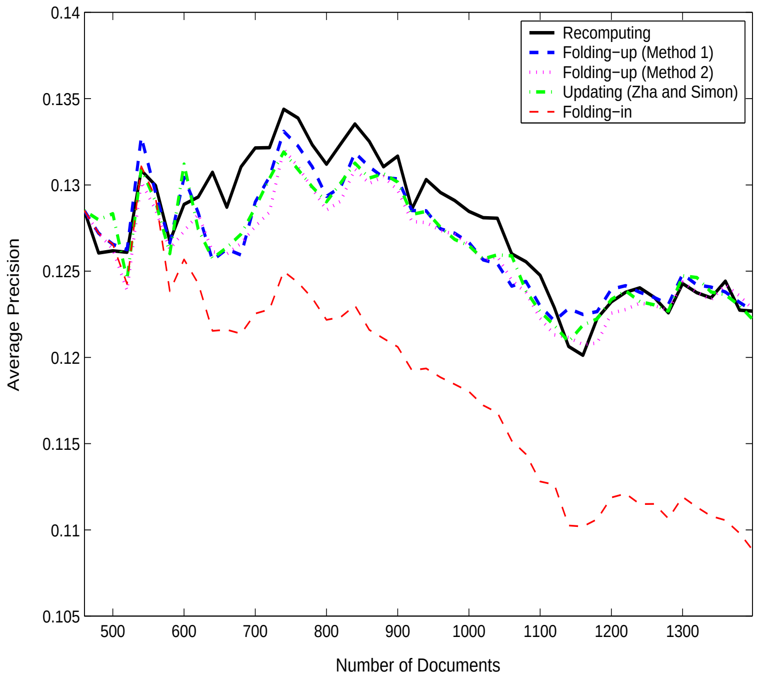

Figure 5.45: Comparison of average precisions for the CISI collection with 1000 documents added in 50 groups of 20.

Method	CPU time	CPU time	CPU time	CPU time	CPU time
	Groups of 2	Groups of 4	Groups of 8	Groups of 10	Groups of 20
Recomputing	87112.9	43705.9	21909.2	17546.1	8754.2
Updating	607.5	309.9	161.0	131.1	72.5
Folding-up 1	130.8	83.9	56.8	52.1	41.7
Folding-up 2	48.9	43.1	37.6	37.8	38.3
Folding-in	16.2	8.7	4.8	4.0	2.3

Table 5.3: Comparison of total CPU times (seconds) for the CISI collection with 1000 documents added in groups of 2, 4, 8, 10, and 20.

In each case in Table 5.3, the folding-up methods are hundreds of times faster than recomputing. Both folding-up methods are also significantly faster than updating.

Chapter 6

Conclusions and Future Work

LSI relies heavily on the PSVD of the term-document matrix representation of a document collection. Calculating the PSVD of large term-document matrices is computationally expensive. In a rapidly expanding environment, a term-document matrix is altered often as new documents and terms are added. Recomputing the PSVD of the term-document matrix each time these slight alterations occur can be prohibitively expensive; therefore when documents (or terms) are added to an existing dataset, it is beneficial to update the existing PSVD to reflect these changes.

As discussed in Section 3.2, folding-in is one method of adding new documents (or terms) to a term-document matrix. Updating the PSVD of the existing term-document matrix, as discussed in Sections 3.3 and 3.4, is another method. The folding-in method is computationally inexpensive, but as demonstrated by the experiments in Chapter 5, it may cause a deterioration in the accuracy of the PSVD that results in decreased retrieval performance when compared with recomputing the PSVD. The PSVD-updating method is computationally more expensive than the folding-in method, but it better maintains the accuracy of the PSVD. This thesis has introduced folding-up, a new hybrid method which combines folding-in and PSVD-updating. The experiments in Sections 5.1.1–5.3.5 demonstrate that folding-up is computationally faster than either recomputing the PSVD or using PSVD-updating methods, with no significant difference in retrieval performance.

Two versions of the folding-up method have been introduced; the first measures deterioration in matrix orthogonality in order to determine when to switch from folding-in to updating, whereas the second method switches from folding-in to updating based on how many documents have been folded-in relative to the number of documents in the most recently updated term-document matrix. These two folding-up methods are referred to as folding-up method 1 and folding-up method 2, respectively. Although folding-up methods 1 and 2 give similar retrieval performance for the three document collections examined in this thesis (Medline, Cranfield, and CISI [9]), folding-up method 1 is computationally more expensive than folding-up method 2. The higher computational cost in folding-up method 1 is due to the matrix multiplication $\mathbf{D}_k^T\mathbf{D}_k$, where $\mathbf{D}_k \in \Re^{t\times p}$ is the term-document matrix containing new documents to be appended to the term-document matrix $\mathbf{A} \in \Re^{t\times d}$. Moreover, the computation time of folding-up method 1 scales more dramatically with the size and number of updates; the computation time for folding-up method 2 is much less sensitive to the number and size of updates. For these reasons, folding-up method 2 would be the better choice in situations where p is small, or in which the speed of the computation is critical.

The goal of the folding-up method is to switch from the process of folding-in to that of using a PSVD-updating method before the accuracy of the numerical representation of the database degrades significantly from the folding-in process. Interesting future work could include establishing an optimal range for the error threshold that is used when measuring the loss of orthogonality during the folding-in stage of folding-up method 1. Tests would include expanding the error threshold range used in the experiments in Chapter 5 (0.75 to 1.50) by testing error thresholds in a wider range, for example 0.10 to 10.00. As well, a more meaningful error threshold measure could be developed by measuring the loss of orthogonality relative to the size of the matrix $\mathbf{D}_k$, for example

$$\frac{\|\mathbf{D}_k^T\mathbf{D}_k\|_\infty}{\|\mathbf{D}_k\|_\infty}.$$

An optimal range for the percentage of documents (relative to the term-document matrix) that can be folded-in to a term-document matrix without a significant loss in retrieval performance could also be investigated. Although these ranges are likely to be somewhat database dependent, the success in using similar ranges for the three diverse collections of abstracts (Medline, Cranfield, and CISI [9]) in Chapter 5 indicates that establishing such ranges is possible. Of course, testing with a variety of larger document collections would be necessary.

Although the folding-up experiments performed for this thesis use the PSVD-updating algorithms of Zha and Simon [33], the folding-up method can easily be adapted to use other PSVD-updating algorithms. As noted in Section 1.3, Okša, Bečka, and Vajteršic [27] have introduced an algorithm for the parallel computation of the SVD of matrices with a particular structure, specifically for use in the PSVD-updating algorithms proposed by Zha and Simon [33]. It would be interesting to compare an implementation of folding-up that is based on Zha and Simon's original PSVD-updating algorithms [33] with an implementation of folding-up that incorporates the parallel SVD computation developed by Okša, Bečka, and Vajteršic [27].

Bibliography

[1] R. A. Baeza-Yates, R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] M. W. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval.* Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, first edition, 1999.

[3] M. W. Berry, S. T. Dumais, and T. A. Letsche. Computational methods for intelligent information access, 1995. Presented at the Proceedings of Supercomputing.

[4] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[5] M. Brand. Fast online SVD revisions for lightweight recommender systems, 1993. In Proceedings of SIAM International Conference on Data Mining, 2003.

[6] M. Buckland and F. Gey. The trade-off between recall and precision. *Journal of the American Society for Information Science*, 45(1):12–19, 1994.

[7] J. Bunch and C. Neilson. Updating the singular value decomposition. *Numerische Mathematik*, 31(2):111–129, 1978.

[8] P. Businger. Updating a singular value decomposition. *BIT*, 10:376 – 385, 1970.

[9] Cornell SMART System ftp://ftp.cs.cornell.edu/pub/smart.

[10] Cornell SMART System ftp://ftp.cs.cornell.edu/pub/smart/english.stop.

[11] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[12] S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers*, 23:229–236, 1991.

[13] W. Ferzali and J.G. Proakis. Adaptive SVD algorithm with application to narrowband signal tracking. *SVD and Signal Processing*, II:149 – 159, 1991.

[14] J. D. Gibbons and S. Chakraborty. *Nonparametric Statistical Inference.* Statistics, A Series of Textbooks and Monographs. Marcel Dekker, New York, fourth edition, 2003.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[16] M. Gu and S. Eisenstat. A fast and stable algorithm for updating the singular value decomposition. *Technical Report, Department of Computer Science, Yale University*, YALEU/DCS/RR-966, 1994.

[17] D. Harman. Ranking algorithms. In *Information retrieval: data structures and algorithms*, pages 363–392. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[18] M. Hollander and D.A. Wolfe. *Nonparametric Statistical Methods.* John Wiley & Sons, New York, second edition, 1999.

[19] G.G. Judge, R.C. Hill, W.E. Griffiths, H. Lütkepohl, and T.C. Lee. *Introduction to the Theory and Practice of Econometrics.* John Wiley & Sons, New York, second edition, 1988.

[20] R. Lehoucq, D. Sorensen, and C. Yang. Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted arnoldi methods, 1997.

[21] T. A. Letsche and M. W. Berry. Large-scale information retrieval with latent semantic indexing. *Information Sciences*, 100(1-4):105–137, 1997.

[22] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford Ser. (2)*, 11:50–59, 1960.

[23] M. Moonen, P. Van Dooren, and J. Vandewalle. An SVD updating algorithm for subspace tracking. *SIAM J. Matrix Anal. Appl.*, 13:1015 – 1038, 1992.

[24] M. Moonen, P. Van Dooren, and J. Vandewalle. A systolic array for SVD updating. *SIAM J. Matrix Anal. Appl.*, 14:353 – 371, 1993.

[25] E.R. Jessup M.W. Berry, Z. Drmac. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[26] G. W. O'Brien. Information tools for updating an SVD-encoded indexing scheme, 1994. Master's Thesis, The University of Knoxville, Tennessee.

[27] G. Okša, M. Bečka, and M. Vajteršic. Parallel SVD computation in updating problems of latent semantic indexing. *Proceedings of ALGORITMY 2002, Conference on Scientific Computing*, pages 113–120, 2002.

[28] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[29] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Computer Science Series. McGraw-Hill, New York, NY, first edition, 1983.

[30] C. Sengupta, J.R Cavallaro, and B. Aazhang. Solving the SVD updating problem for subspace tracking on a fixed size linear array of processors. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 5:4137–4140, 1997.

[31] L. N. Trefethen and D. Bau, III. *Numerical linear algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[32] J. H. Wilkinson and C. Reinsch, editors. *Handbook for automatic computation. Vol. II.* Springer-Verlag, New York, 1971. Linear algebra, Compiled by J. H. Wilkinson and C. Reinsch, Die Grundlehren der Mathematischen Wissenschaften, Band 186.

[33] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21(2):782–791, 1999.

[34] H. Zha and Z. Zhang. Matrices with low-rank-plus-shift structure: partial SVD and latent semantic indexing. *SIAM J. Matrix Anal. Appl.*, 21(2):522–536, 1999.