The University of Saskatchewan
Department of Computer Science

Technical Report #2005-03

Information Retrieval manuscript No.
(will be inserted by the editor)

Updating the Partial Singular Value Decomposition in Latent Semantic Indexing

Jane E. Tougas[1] ⋆, **Henry Stern**[2], **Raymond J. Spiteri**[3] ⋆⋆

[1] Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada (tougas@cs.dal.ca).

[2] Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada (stern@cs.dal.ca).

[3] Department of Computer Science, University of Saskatchewan, Saskatoon, SK, S7N 5C9, Canada (spiteri@cs.usask.ca).

Received: date / Revised version: date

Abstract Latent semantic indexing (LSI) is a method of information retrieval that relies heavily on the partial singular value decomposition (PSVD) of the term-document matrix representation of a dataset. Calculating the PSVD of large term-document matrices is computationally expensive; hence in the case where terms or documents are merely added to an existing dataset, it is extremely beneficial to *update* the previously calculated PSVD to reflect the changes. In this article we show how updating

Send offprint requests to: Jane E. Tougas

⋆ *Research partially supported by NSERC Canada and a Killam scholarship.*

⋆⋆ *Research partially supported by a grant from NSERC Canada.*

can be used in LSI to significantly reduce the computational cost of finding the PSVD without significantly impacting performance. Moreover, we show how the computational cost can be reduced further, again without impacting performance, through a combination of updating and folding-in.

1 Introduction

The seemingly disparate fields of information retrieval (IR) and numerical linear algebra (NLA) are closely linked via latent semantic indexing (LSI) [5]. LSI is an IR method based on the vector-space model where a dataset is represented as a *term-document matrix*. LSI uses a matrix factorization method known as the partial singular value decomposition (PSVD) in an attempt to reduce the problems of *precision* and *recall failure* caused by *polysemy* and *synonymy*. Many terms have more than one meaning (they are *polysemous*). When a polysemous term is used in a search query, irrelevant documents about the term's other meaning(s) may be retrieved, degrading the precision level of the results. Moreover, many terms have similar meanings (they are *synonymous*). When a term that has a synonym is used in a query, relevant documents containing the synonym, but not the query term, may be overlooked, degrading the recall level of the results. Research indicates that LSI is more successful in dealing with the problems caused by synonymy than those caused by polysemy [5], but this does not detract from the importance of LSI in IR. As a vector-space model, LSI examines the document collection as a whole and determines which docu-

ments contain many of the same terms. The more terms that documents have in common, the more closely related the documents are considered to be. This process involves creating a term-document matrix $\mathbf{A} \in \Re^{t \times d}$, in which there is a column vector for each document, with as many entries as there are semantically significant terms in the documents. Each entry is the weighted frequency of a particular term in a particular document. The term-document matrix represents a t-dimensional space with t-dimensional document vectors, where t is the number of semantically significant terms. Each vector contains the coordinates of that document's location in the t-dimensional space. Queries are also represented as t-dimensional vectors. The vectors of documents and queries with many terms in common will be close together, whereas those with relatively few terms in common will be far apart. The query vectors are projected into the term-document matrix using the PSVD.

Even using the most advanced NLA methods, computing the PSVD of a matrix is an extremely expensive process. Because of the tremendous size of modern databases, a term-document matrix can potentially be very large, with hundreds of thousands or even millions of entries. In LSI, this means that most of the processing time is spent in performing the PSVD calculation [2], [3]. In a rapidly expanding environment, such as the Internet, the term-document matrix is altered often as new documents and terms are added. Recalculating the PSVD of the matrix each time these slight alterations occur is prohibitively expensive. Traditionally, LSI uses a process

known as *folding-in* to modify the PSVD. Although this method is very efficient, its accuracy may degrade, especially the more it is performed. An efficient and much more accurate approach is to *update* the PSVD; e.g., [6]. In this approach the existing PSVD is modified to reflect the changes to the term-document matrix; i.e., the PSVD of the modified term-document matrix is obtained by modifying the PSVD of the original term-document matrix.

The purpose of this paper is not only to show that updating the PSVD is more accurate than folding-in, but also to show that a combination of folding-in and updating the PSVD (which we call *folding-up*) is an even more attractive option than either folding-in or updating the PSVD on their own. Folding-up offers a significant improvement in computation time when compared to either recomputing the PSVD or just updating the PSVD. At the same time, folding-up provides a level of precision that is not statistically different from that given by recomputing the PSVD each time changes are made to the term-document matrix.

The remainder of the paper proceeds as follows. Section 2 covers background information on the PSVD and on the folding-in process, Section 3 gives a description of the algorithms used for updating the PSVD [6], and Section 5 gives experimental results using the document updating algorithm and the MEDLINE and CRANFIELD data collections [4]. Finally, Section 6 presents our conclusions.

2 Background

2.1 SVD

The SVD is a matrix factorization that can be used to capture the salient features of a matrix by determining important vectors (directions) and quantifying their importance via weighting factors. Given a matrix $\mathbf{A} \in \Re^{t \times d}$, its SVD is written as $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U} \in \Re^{t \times t}$, $\mathbf{V} \in \Re^{d \times d}$, and $\mathbf{\Sigma} \in \Re^{t \times d}$. $\mathbf{U}$ and$\mathbf{V}$ are *orthogonal matrices* containing the left and right *singular vectors* of $\mathbf{A}$ respectively. When $\mathbf{A}$ is a term-document matrix, $\mathbf{U}$ represents the term vectors and $\mathbf{V}$ represents the document vectors. The matrix $\mathbf{\Sigma}$ potentially has non-zero entries only on the diagonal. These diagonal entries, denoted σ_j for $j = 1, 2, ..., \min(m, n)$ and arranged in non-increasing order, are known as the *singular values* of matrix $\mathbf{A}$. The number of non-zero singular values of a matrix is known as its *rank*, r.

The SVD can be interpreted as the weighted sum of r rank-one matrices, $\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$, where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the jth columns of matrices $\mathbf{U}$ and $\mathbf{V}$, respectively. This interpretation of the SVD facilitates the formation of lower-rank approximations of $\mathbf{A}$. Replacing r in this sum by any k with $0 \leq k < r$ gives the *partial* SVD of $\mathbf{A}$, $\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \approx \mathbf{A}$. In matrix form, this is equivalent to taking $\mathbf{U}_k$ and $\mathbf{V}_k$ to be the first k columns of $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{\Sigma}_k$ to be the leading $k \times k$ submatrix of $\mathbf{\Sigma}$, yielding $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$. 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 huge dimensional reduction on the data is a muting

of the noise caused by synonymy 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. We note, however, that $\mathbf{A}_k$ is never explicitly formed; we use the matrices $\mathbf{U}_k, \mathbf{\Sigma}_k$, and $\mathbf{V}_k$ instead. The number of dimensions k to keep in the reduced term-document matrix when d is very large is still open to study and debate, but experiments indicate that values of k between 100 and 300 typically give the best results [3].

2.2 Folding-In

In LSI, when new documents and terms are added to a dataset, it is necessary to modify the PSVD of the term-document matrix to reflect these changes. Because recomputing the PSVD is very expensive, the method of folding-in new documents and terms is often used.

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.

Because we are using the PSVD, $\mathbf{D}$ must be 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.

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.

$\mathbf{T}$ must be 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.

3 Updating Methods

Updating the PSVD when the term-document matrix changes is a more complicated process than folding-in. However, the end result (in the absence of roundoff errors) is the exact PSVD of the modified term-document matrix without the expense of recomputing it from scratch. Typically, the PSVD is updated to reflect the new documents that have been added to the document collection. As with folding-in, adding these new documents

will often mean that new terms also need to be added, so the PSVD is then updated to reflect these changes. Finally, another updating method allows the PSVD to be updated again to reflect the changes to the term weights in the term-document matrix caused by the additional documents and terms. Subsection 3.1 describes document updating, 3.2 describes term updating, and 3.3 describes term weight updating. Each of the methods described is based on the updating method introduced by Zha and Simon [6]. This method does require one QR decomposition and one SVD per update; however, these potentially expensive computations are only performed on small intermediate matrices, where the computational complexity scales on the order of the size of the update and/or the reduced dimension k, not dimensions of the original matrix (see below).

As before, let $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ 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. It is assumed that the PSVD of $\mathbf{A}$ has been computed by some means prior to updating.

In the following, we let $\mathbf{I}_n$ denote the identity matrix of size n.

3.1 Updating documents

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, and let

$\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}]$ be the updated term-document matrix. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{D}} \in \Re^{t \times p} = \left(\mathbf{I}_t - \mathbf{U}_k \mathbf{U}_k^T\right) \mathbf{D}$.

Form the QR decomposition of $\hat{\mathbf{D}}$ such that $\mathbf{Q_D R_D} = \hat{\mathbf{D}}$, where $\mathbf{Q_D} \in \Re^{t \times p}$ is orthonormal, and $\mathbf{R_D} \in \Re^{p \times p}$ is upper triangular. Then

$$\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}] \approx [\mathbf{A}_k, \mathbf{D}] = [\mathbf{U}_k, \mathbf{Q_D}] \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T \mathbf{D} \\ \mathbf{0} & \mathbf{R_D} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{bmatrix}.$$

Now let $\hat{\mathbf{A}} \in \Re^{(k+p) \times (k+p)}$ be the matrix defined by

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T \mathbf{D} \\ \mathbf{0} & \mathbf{R_D} \end{bmatrix}.$$

Form the SVD of $\hat{\mathbf{A}}$ such that

$$\hat{\mathbf{A}} = \left[\hat{\mathbf{U}}_k, \hat{\mathbf{U}}_p\right] \begin{bmatrix} \hat{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{\Sigma}}_p \end{bmatrix} \left[\hat{\mathbf{V}}_k, \hat{\mathbf{V}}_p\right]^T,$$

where $\hat{\mathbf{U}}_k \in \Re^{(k+p) \times k}$, $\hat{\mathbf{\Sigma}}_k \in \Re^{k \times k}$, and $\hat{\mathbf{V}}_k \in \Re^{(k+p) \times k}$. Then the PSVD of $\tilde{\mathbf{A}}$ in k dimensions (the updated PSVD) is

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

3.2 Updating terms

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 documents, and let $\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{T}]$ be the updated term-document matrix. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{T}} \in \Re^{d \times q} = \left(\mathbf{I_d} - \mathbf{V}_k \mathbf{V}_k^T\right) \mathbf{T}^T$.

Form the QR decomposition of $\hat{\mathbf{T}}$ such that $\mathbf{Q_T R_T} = \hat{\mathbf{T}}$, where $\mathbf{Q_T} \in \Re^{d \times q}$ is orthonormal, and $\mathbf{R_T} \in \Re^{q \times q}$ is upper triangular. Then

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

Now let $\hat{\mathbf{A}} \in \Re^{(k+q) \times (k+q)}$ be the matrix defined by

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix} .$$

Form the SVD of $\hat{\mathbf{A}}$ such that

$$\hat{\mathbf{A}} = \left[\bar{\mathbf{U}}_k, \bar{\mathbf{U}}_q\right] \begin{bmatrix} \bar{\mathbf{\Sigma}}_k & \mathbf{0} \\ \\ \mathbf{0} & \bar{\mathbf{\Sigma}}_q \end{bmatrix} \left[\bar{\mathbf{V}}_k, \bar{\mathbf{V}}_q\right]^T ,$$

where $\bar{\mathbf{U}}_k \in \Re^{(k+q) \times k}$, $\bar{\mathbf{\Sigma}}_k \in \Re^{k \times k}$, and $\bar{\mathbf{V}}_k \in \Re^{(k+q) \times k}$. Then the PSVD of $\tilde{\mathbf{A}}$ in k dimensions (the updated PSVD) is

$$\tilde{\mathbf{A}}_k = \left(\begin{bmatrix} \mathbf{U}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q \end{bmatrix} \bar{\mathbf{U}}_k \right) \bar{\mathbf{\Sigma}}_k \left([\mathbf{V}_k, \mathbf{Q_T}] \, \bar{\mathbf{V}}_k \right)^T .$$

3.3 Updating term weights

Let $\mathbf{S} \in \Re^{t \times s}$, where s is the number of terms whose term weights need adjusting, be a selection matrix in which each column contains one 1, and all other entries are zero. Let $\mathbf{W} \in \Re^{d \times s}$ be the matrix in which each column $\mathbf{W}_i$ contains the difference between the old term weights and the new term weights for the term i. Let $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{S}\mathbf{W}^T$ be the adjusted term-document matrix. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{S}} \in \Re^{t \times s} = \left(\mathbf{I}_t - \mathbf{U}_k \mathbf{U}_k^T \right) \mathbf{S}$; let $\hat{\mathbf{W}} \in \Re^{d \times s} = \left(\mathbf{I}_d - \mathbf{V}_k \mathbf{V}_k^T \right) \mathbf{W}$.

Form the QR decomposition of $\hat{\mathbf{S}}$ such that $\mathbf{Q_M R_M} = \hat{\mathbf{S}}$, where $\mathbf{Q_M} \in \Re^{t \times s}$ is orthonormal, and $\mathbf{R_M} \in \Re^{s \times s}$ is upper triangular.

Form the QR decomposition of $\hat{\mathbf{W}}$ such that $\mathbf{Q_N R_N} = \hat{\mathbf{W}}$, where $\mathbf{Q_N} \in \Re^{d \times s}$ is orthonormal, and $\mathbf{R_N} \in \Re^{s \times s}$ is upper triangular. Then

$$\begin{aligned} \tilde{\mathbf{A}} &= \mathbf{A} + \mathbf{S}\mathbf{W}^T \\ &\approx \mathbf{A}_k + \mathbf{S}\mathbf{W}^T = [\mathbf{U}_k, \mathbf{Q_M}] \left(\begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T \mathbf{S} \\ \mathbf{R_M} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T \mathbf{W} \\ \mathbf{R_N} \end{bmatrix}^T \right) [\mathbf{V}_k, \mathbf{Q_N}]^T . \end{aligned}$$

Now let $\hat{\mathbf{A}} \in \Re^{(k+s) \times (k+s)}$ be the matrix defined by

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R_M} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R_N} \end{bmatrix}^T .$$

Form the SVD of $\hat{\mathbf{A}}$ such that

$$\hat{\mathbf{A}} = \left[\tilde{\mathbf{U}}_k, \tilde{\mathbf{U}}_s\right] \begin{bmatrix} \tilde{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{\Sigma}}_s \end{bmatrix} \left[\tilde{\mathbf{V}}_k, \tilde{\mathbf{V}}_s\right]^T ,$$

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

$\tilde{\mathbf{A}}$ in k dimensions (the updated PSVD) is

$$\tilde{\mathbf{A}}_k = \left([\mathbf{U}_k, \mathbf{Q_M}]\, \tilde{\mathbf{U}}_k\right) \tilde{\mathbf{\Sigma}}_k \left([\mathbf{V}_k, \mathbf{Q_N}]\, \tilde{\mathbf{V}}_k\right)^T .$$

4 Folding-up

It is well known that folding-in is a very inexpensive way compared to recomputing the PSVD to incorporate new information []. However, because the matrices $\mathbf{V}_k$ and $\mathbf{\Sigma}_k$ are never changed, the quality of the results produced by folding-in can be expected to deteriorate (perhaps even rapidly) after even only a small number of updates. On the other hand, updating the PSVD gives exactly 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 method which we call *folding-up* that uses a combination of

folding-in and updating at each increment in order to reduce the computational expense of updating even further without significantly degrading the results.

The idea behind folding-up is to fold-in documents until the number of documents folded-in reaches a pre-selected percentage of the current term-document matrix. If no updates have previously been done, the current term-document matrix is the initial matrix; otherwise it is the last updated term-document matrix. Once the number of documents that have been folded-in reaches the pre-selected percentage of the original matrix, the vectors that have been appended to $\mathbf{V}_k$ during folding-in are discarded. The PSVD is then updated to reflect the addition of all of the document vectors that have been folded-in since the last update. These document vectors are then discarded. The process then continues with folding-in until the next update.

The process of folding-up 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. We demonstrate by means of examples below that folding-in produces results that are not statistically different from those produced by recomputing the PSVD.

5 Experiments

The experiments in this section are run using *Matlab* Release 13 on an Ultra3 SunFire V880 (Solaris 8 operating system). Examples 5.1–5.2 use the MEDLINE text collection [4], containing 1033 documents and 30 queries. Removing semantically insignificant terms and stemming the remaining terms gives a term-document matrix $\mathbf{A}_{\text{MED}} \in \Re^{5735\times 1033}$. Examples 5.3–5.4 use the CRANFIELD text collection [4], containing 1400 documents and 225 queries. For this collection, no stemming is done, but semantically insignificant words are removed, giving a term-document matrix $\mathbf{A}_{\text{CRAN}} \in \Re^{5321\times 1400}$. For each text collection, we use a *term frequency inverse document frequency* (tfidf) weighting scheme [1]. The measure of similarity is the cosine of the angle between query and document vectors.

For each example, we start with a term-document matrix and incrementally update it with document vectors until the size of its column space has approximately doubled. Because the results from recomputing the PSVD represent what the other methods are attempting to reproduce, we then compare the final average precision obtained for folding-in, updating, and folding-up in each example with recomputing the PSVD. The statistical comparisons are made pairwise using a non-parametric Kruskal-Wallis test at significance level 0.05. In each case, the average precision for each of the queries at 11 standard recall levels $(0\%, 10\%, \cdots, 100\%)$ is averaged to produce the overall average precision at each increment of each experiment. For each method used, we plot the average precision at each of these increments,

starting with the initial term-document matrix. All PSVDs are computed using the *Matlab* function `svds`, with $k = 125$ for the MEDLINE collection and $k = 300$ for the CRANFIELD collection, where k is the number of singular values and corresponding left and right singular vectors computed. For the sake of brevity, the experiments described use only document updating. We note that similar results are produced using term updating.

5.1 Example 1

We partition $\mathbf{A}_{\text{MED}} \in \Re^{5735 \times 1033}$ so that the first 533 columns are used as the initial term-document matrix, and the remaining columns are added incrementally in groups of size 10. We compare the average precision (as described above) for four methods: recomputing the PSVD at each increment, folding-in at each increment, updating at each increment, and folding-up with folding-in at each increment and updates occurring when the number of documents folded-in reaches approximately 14% of the size of the initial matrix for the first update, and of the updated matrix thereafter.

As expected, Figure 1 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 ($p = 0.02$).

The average precision for updating does not begin to deteriorate until the initial matrix is more than one and a half times its original size, and the increments are less than 1.25% of the size of the matrix; the final average

precision is not significantly different from that of recomputing the PSVD ($p = 0.89$). Although the deterioration is slight, it does indicate that doing many updates that are very small relative to the size of the matrix may eventually have a negative affect on the average precision. However, the savings in computation time compared to recomputing, as shown in Table 1, may more than compensate for this small deficiency; in this case, updating is more than 100 times faster than recomputing.

Figure 1 shows that in this example, folding-up actually outperforms the other methods for much of the time, and the final average precision is not significantly different than recomputing the SVD ($p = 0.77$); it is also faster than either recomputing or just updating. See Table 1 for a comparison of CPU times.

5.2 Example 2

We partition $\mathbf{A}_{\text{MED}} \in \Re^{5735 \times 1033}$ so that the first 533 columns are used as the initial term-document matrix, and the remaining columns are added incrementally in groups of size 25. We compare the average precision for the four methods: recomputing the PSVD at each increment, folding-in at each increment, updating at each increment, and folding-up (as described in Example 5.1).

As in Figure 1, Figure 2 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 ($p = 0.02$).

The average precision for updating does not deteriorate relative to recomputing the PSVD, and indeed it is at times slightly better; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.84$). These results suggest that updating in larger increments, relative to the size of the matrix, can give better average precision. Again, Table 1 shows that updating the PSVD is much faster than recomputing each time the term-matrix changes, but in these examples, folding-in is by far the fastest method.

Folding-up again outperforms the other methods in terms of precision at various points of the experiment; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.88$). It also takes less computation time than recomputing the PSVD or simply updating it. Table 1 gives a comparison of the CPU times for the methods.

Method	CPU time	CPU time
	Increments of 10	Increments of 25
Recomputing	5001.60	2045.80
Updating	43.07	22.33
Folding-in	1.35	0.75
Folding-up	15.76	13.14

Table 1 Comparison of total CPU times (seconds) for the MEDLINE collection, with 500 documents added in groups of 10 and in groups of 25.

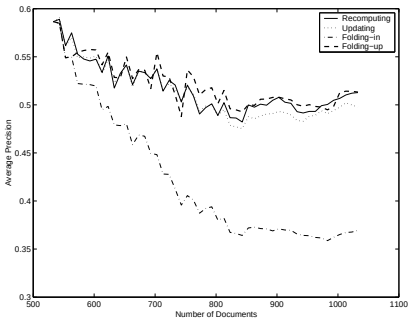

Fig. 1 Comparison of average precisions of four methods, for the MEDLINE collection, with 500 documents added in 50 groups of 10.

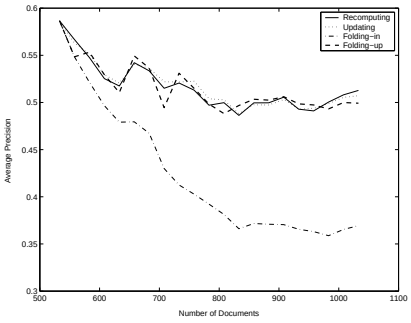

Fig. 2 Comparison of average precisions of four methods, for the MEDLINE collection, with 500 documents added in 20 groups of 25.

5.3 Example 3

We partition $\mathbf{A}_{\text{CRAN}} \in \Re^{5321\times 1400}$ such that the first 700 columns are used as the initial term-document matrix, and the remaining columns are added incrementally in groups of size 14. We compare the average precision for four methods: recomputing the PSVD at each increment, folding-in at each increment, updating at each increment, and folding-up with updates occurring when the number of documents folded-in reaches approximately 8% of the size of the initial matrix for the first update, and of the updated matrix thereafter. As expected, Figure 3 shows that the average precision for folding-in falls below that of the other methods; the final average precision is significantly different from that of recomputing the PSVD ($p = 0.03$). We note that the overall average precision is low because no stemming of terms was done when the text collection was processed. The average pre-

cisions for recomputing and for updating the PSVD are very similar, with the final precisions not being significantly different ($p = 0.94$), even though Table 2 shows that in this case, updating is more than 150 times faster than recomputing the PSVD. Figure 3 also shows that in this example, folding-up at times outperforms updating but otherwise performs similarly to both updating and recomputing the PSVD; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.86$). Table 2 shows that folding-up is more than three times faster than updating, and more than 580 times faster than recomputing the PSVD at each increment.

5.4 Example 4

We partition $\mathbf{A}_{\text{CRAN}} \in \Re^{5321 \times 1400}$ such that the first 700 columns are used as the initial term-document matrix, and the remaining columns are added incrementally in groups of size 28. We again compare the average precision for four methods: recomputing the PSVD at each increment, folding-in at each increment, updating at each increment, and folding-up (as described in Example 5.1). As in Figure 3, Figure 4 shows that the average precision for folding-in falls below that of the other methods; the final average precision is significantly different from that of recomputing the PSVD ($p = 0.03$). Again the average precision of recomputing and of updating the PSVD are again very similar; the final average precision is not significantly different than recomputing the SVD ($p = 0.88$). Folding-in is by far the fastest method, but as Table 2 shows, updating is still more than 150 times faster than

recomputing the PSVD. Folding-up gives similar overall average precision to that of recomputing or updating the PSVD; the final average precision is not significantly different than recomputing the SVD ($p = 0.90$). However, in this case is more than 300 times faster than recomputing, and it is almost twice as fast as updating.

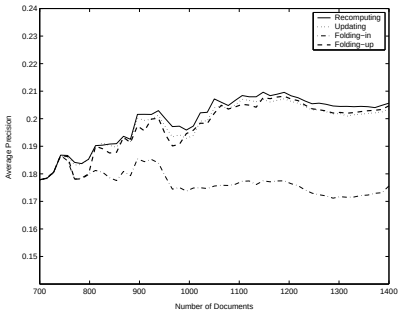

Fig. 3 Comparison of average precisions of four methods, for the CRANFIELD collection, with 700 documents added in 50 groups of 14.

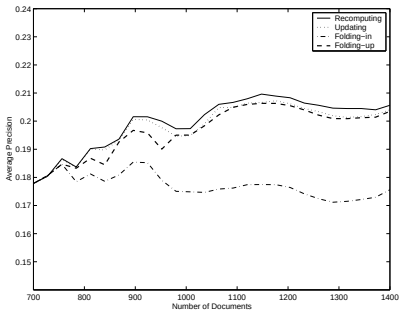

Fig. 4 Comparison of average precisions of four methods, for the CRANFIELD collection, with 700 documents added in 20 groups of 28.

6 Conclusions

LSI makes heavy use of the PSVD in its implementation. Often, the term-document matrix may need frequent changes when new documents and terms are added to the data collection. In such cases, it is beneficial to exploit the previously computed PSVD via updating. We have demonstrated that updating the PSVD of the term-document matrix each time these types of changes are made to the matrix is not only much faster (typically by an

Method	CPU time	CPU time
	Increments of 14	Increments of 28
Recomputing	51548.30	26335.33
Updating	294.28	162.11
Folding-in	7.27	4.13
Folding-up	88.82	81.57

Table 2 Comparison of total CPU times (seconds) for the CRANFIELD collection, with 700 documents added in groups of 14 and in groups of 28.

order of magnitude) than recomputing the PSVD, but it also gives better average precision than the traditional method of folding-in documents and terms. We have also demonstrated that folding-up, a new approach that is a hybrid of folding-in and updating, gives better average precision than folding-in, with less computation time (typically by a factor of 2 or 3) than updating alone. Our examples also illustrate a viable method for determining when to perform the updating in the folding-up procedure based on the number of documents that are being added as a percentage of the size of the current term-document matrix. The folding-up method offers an excellent speed-up in computation time (typically by a factor of between 20 and 30), with little or no loss of overall average precision compared to recomputing the PSVD.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.
2. M. W. Berry, S. T. Dumais, and T. A. Letsche. Computational methods for intelligent information access, 1995. Presented at the Proceedings of Supercomputing.
3. 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.
4. Cornell SMART System ftp://cs.cornell.edu/pub/smart.
5. 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.
6. H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21(2):782–791, 1999.