A Formal Model for Updating Rare Term Vector Replacement

Tobias Berka
Department of Computer Sciences, University of Salzburg, Salzburg, Austria
tberka@cosy.sbg.ac.at

Abstract
Dimensionality reduction is a means to alleviate the curse of dimensionality for statistical text models. To support dynamic corpora in real use, we need to consider three principle updating operations: adding new documents, changing existing documents, and deleting obsolete documents. Efficient algorithms that apply such updates to statistical models without requiring a complete rebuild are a prerequisite to large-scale, practical deployment in practice. In this paper, we present a formal framework for updating rare term vector replacement (RTVR), a recent dimensionality reduction technique for text. We develop a formal understanding of the updating operations, analyze how these affect the statistical model, and discuss how they lead to an efficient updating algorithm that minimizes data movement. We study the resulting algorithmic complexity in detail, and present measurements that demonstrate that we achieve a significant speed-up over a complete rebuild.

Keywords: Information Retrieval, Dimensionality Reduction, Model Updating

1 Introduction
Much research in natural language processing (NLP) may assume that corpora are static collections that do not change over time. But for many modern applications, intelligent corporate intranets, content management systems, groupware systems, and e-mail clients, the documents evolve under real-world conditions. Statistical and machine learning models that are extracted from document collections need to be updated to carry the changes to the corpus over into the model. We need to support three basic operations: adding, changing and deleting documents.

Our paper is concerned with updating rare term vector replacement (RTVR), a dimensionality reduction technique. Multivariate statistical models for text are commonly plagued by the curse of dimensionality. The sparse and irregular nature of text, combined with the high number of features and relatively high number of variables, creates a wide range of technical and statistical problems. Finding means to map the original, irregular and high-dimensional space to a low-dimensional, compact and regular representation is the task of dimensionality reduction techniques. Common techniques used in text mining include latent semantic indexing (LSI) [1] or COV [2], probabilistic latent semantic analysis (PLSA) [3], and extensions such as the kernel PCA [4], clustered LSI (CLSI) [5], latent semantic kernels (LSK) [6], and explicit semantic analysis (ESA) [7]. Updating operations for LSI are well understood [8], and are also being developed for PLSA [9], or other dimensionality reduction methods such as the kernel PCA [4].

Rare term vector replacement (RTVR) is a recent technique for the dimensionality reduction of term-frequency vectors [10]. It is a projection-based, linear mapping that is based on an index-vector representation. The method eliminates rare terms, i.e., terms with a very low number of containing documents, by constructing surrogate vectors that express their meaning as a vector of common term relevancies. An updating algorithm for RTVR has recently been presented [11].

Our contribution is the following. In this paper, we present a formal model for updating rare term vector replacement that fully justifies the updating algorithm. It introduces a terminology, along with a formal notation, that can be used to describe the complex interplay of changing occurrence counts, term sets, and vector spaces. This extensive formal treatment demystifies the updating process, explains the existing RTVR updating algorithm, and may enable others to develop and discuss novel, specialized updating algorithms for RTVR.

The remainder of this paper is structured as thus. In Section 2, we describe the formalisms for computing and updating rare term vector replacement. We discuss the updating algorithm
in Section 3, and analyze the algorithmic complexity in Section 4. Lastly, we summarize our findings in Section 5.

2 A Formal Updating Model

Formally, we have a set $D$ of $n$ documents and a set $T$ of $m$ terms. Every document is represented as a term frequency vector $d_i \in \mathbb{R}^m$. We use a corpus matrix $C \in \mathbb{R}^{m \times n}$, which contains the documents’ term frequency vectors as column vectors. We can determine which documents contain any particular term $t_i$ using the function $D(t_i) = \{d_j \in D \mid C_{i,j} \neq 0\}$. Analogously, we can define a function $T$ that returns all terms occurring in a document $d_j$, $T(d_j) = \{t_i \in T \mid C_{i,j} \neq 0\}$.

If we choose a threshold $\delta$ between 1% and 3% of all documents, the set of elimination terms $E \subseteq T$ is $E = \{t \in T \mid \|D(f)\| \leq \delta\}$, giving us $k = m - \|E\|$ common terms after replacement.

We will replace rare terms $t_i \in E$ by the weighted centroid $\rho(t_i)$ of the documents that contain them, formally

$$\rho(t_i) = \frac{1}{\lambda(t_i)} \sum_{d_j \in D(t_i)} C_{i,j} \tau_E(C_{i:m,j}), \quad (1)$$

where $\tau_E$ is a truncation operator that eliminates all terms in $E$ from the vector, and the scaling factor $\lambda(t_i)$ is the absolute sum of all contributions,

$$\lambda(t_i) = \sum_{d_j \in D(t_i)} |C_{i,j}|. \quad (2)$$

For any document $d_j$, the reduced document vector $\tilde{d}_j$ is formed as the linear combination of the truncated source document and the scaled replacement vectors:

$$\tilde{d}_j = \tau_E(d_j) + \sum_{t_i \in E} C_{i,j} \rho(t_i). \quad (3)$$

The geometric interpretation of this replacement is quite simple. If a rare term occurs in only a single document, that term is replaced by the truncated vector of this document. Should we now search for this term, the document containing the term is returned with a very low distance, followed by more, similar documents. If a term occurs in more than one document, the weighted centroid of these documents is used to represent that term. The performance of the replacement now depends on the applicability of the cluster hypothesis [12; 13], i.e., that documents in close proximity are semantically related. If the cluster hypothesis holds, then the centroid of the containing documents will act as a succinct representation of all documents containing the replacement term.

If we use the notation $e_1, ..., e_m \in \mathbb{R}^m$ to denote the standard base vectors, we can define replacement vectors $r_i$, which either preserve terms we do not wish to eliminate, or perform the vector replacement on terms $t_i \in E$,

$$r_i = \begin{cases} \rho(t_i) & \text{... } t_i \in E \\ \tau_E(e_i) & \text{... } t_i \notin E. \end{cases} \quad (4)$$

We now assemble these vectors column-wise to form a replacement matrix $R$

$$R = \begin{bmatrix} r_1 & \cdots & r_m \end{bmatrix} \in \mathbb{R}^{k \times m}. \quad (5)$$

An improved, dimensionality-reduced corpus matrix $\hat{C}$ can now be obtained by taking

$$\hat{C} = RC. \quad (6)$$

The replacement vectors are weighted centroids, or weighted average vectors, of the document vectors containing their terms. The obvious method for updating the replacement vectors for all affected terms is therefore the running average. Analogously, the reduced document vectors are also a linear combination of the truncated original document vectors and the replacement vectors. For promoted and demoted terms, we can update the reduced document vectors like a running average. But for affected terms, the replacement vector changes. This means that we need to remove the old replacement vector from the reduced vectors, before adding the new replacement vector. This involves all documents that contain the affected, promoted, and demoted terms in the old and/or new term frequency vectors.

The first step is to adjust the matrices $C$ and $R$, as well as the normalizing factor vector $\lambda$, by deleting any terms that no longer occur after the update, and appending terms that are newly introduced during the update.

Formally, we can represent changing a document by a tuple $(i, v)$ containing an identifier or corpus matrix column $i \in \{1, ..., n\}$ and a new term frequency vector $v \in \mathbb{R}^{m'}$, where $m'$ is the
new number of terms. We can cast all three operations in a single framework by introducing two symbols. We let $\nu$ denote the identifier or column to be used for adding new documents, i.e., $(\nu, d_{n + 1})$ denotes a new document that will be added as a new column of the corpus matrix. We let $\epsilon$ denote an empty document vector for the deletion of an old document, i.e., $(i, \epsilon)$ signifies the deletion of the document in column $i$. Using this framework, any update $u$ has the form

$$
u \in \{\nu, 1, 2, ..., n\} \times (\{\epsilon\} \cup \mathbb{R}^m).$$  

We can then apply an individual update $(i, v)$ to a corpus $C$ as

$$
\begin{bmatrix}
C & v \\
C_{*,1:i=1} & C_{*,i+1:m} \\
C_{*,1:i=1} & v & C_{*,i:n}
\end{bmatrix}
\cdots
\begin{cases}
\text{i = } \nu \\
\text{v = } \epsilon \\
\text{o/w}
\end{cases}
\tag{8}
$$

with additional row(s) of $C$ or $v$ padded with zeroes as required.

A batch update $U = \{u_1, ..., u_k\}$ is a (finite) set of updates, i.e.,

$$
U \subset \{\nu, 1, 2, ..., n\} \times (\{\epsilon\} \cup \mathbb{R}^m).
\tag{9}
$$

It can be applied recursively with the $\uparrow$ operator,

$$
\uparrow (C, \emptyset) = C, \tag{10}
$$

$$
\uparrow (C, \{(i, v), U\}) = \uparrow (\uparrow (C, U), i, v). \tag{11}
$$

We can now define the updated corpus $C'$ as

$$
C' = \uparrow (C, U). \tag{12}
$$

The updated set of elimination terms $E'$ with at most $\delta$ occurrences, reduced dimensionality $k'$, and the associated sets of documents $D'(f)$ can be defined as

$$
D'(t) = \{d_j \in D \mid C'_{i,j} \neq 0\}, \tag{13}
$$

$$
E' = \{t \in T \mid D'(t) \leq \delta\}, \tag{14}
$$

$$
k' = m' - ||E'||. \tag{15}
$$

We associate every update $u = (i, v)$ with two term frequency vectors. If the update is not an add document request, i.e., $i \neq \nu$, it is associated with an old document vector $C_{1:m,i}$, or an empty vector otherwise. If it is not a deletion, i.e., $v \neq \epsilon$, it is associated with a new document vector $v$, otherwise with the empty vector. Formally, we define the operators old and new as

$$
\text{old}(i, v) = \begin{cases}
C_{1:m,i} & \text{i} \neq \nu \\
0 & \text{otherwise},
\end{cases} \tag{16}
$$

$$
\text{new}(i, v) = \begin{cases}
v & v \neq \epsilon \\
0 & \text{otherwise}.
\end{cases} \tag{17}
$$

We let the terms in the old vector $T_\nu(i, v)$, the new vector $T_n(i, v)$, and both $T(i, v)$ be

$$
T_\nu(i, v) = \{t_j \in T \mid \text{old}(i, v)_j \neq 0\}, \tag{18}
$$

$$
T_n(i, v) = \{t_j \in T \mid \text{new}(i, v)_j \neq 0\}, \tag{19}
$$

$$
T(i, v) = T_\nu(i, v) \cup T_n(i, v). \tag{20}
$$

Now, we can define the set of all terms $T(U)$ contained in a batch update $U$ as

$$
T(U) = \bigcup_{(i,v) \in U} T(i, v). \tag{21}
$$

One key complicating factor is the fact that the occurrence count of the terms changes. This means that some rare terms may become common terms, and therefore become part of the reduced-dimensional, projected term space. Dually, some common terms may become rare terms, and drop out of the projected space. To develop a consistent vocabulary for our discussion, we will refer to these terms as promoted and demoted terms. We can define the set of promoted terms $P$ as

$$
P = \{t \in T \mid t \in E \land t \notin E'\}, \tag{22}
$$

and the set of demoted terms $Q$ as

$$
Q = \{t \in T \mid t \notin E \land t \in E'\}. \tag{23}
$$

We are going to reorganize the reduced space so that all affected terms $A_T$ are placed in the first $||A_T||$ rows. These terms were already present in the original reduces space. To this end, we introduce an additional variable $k''$ that indicates the number of rows that are preserved,

$$
k'' = ||A_T||. \tag{24}
$$

For demoted terms, we need to compute the replacement vectors from scratch during the update. But for all rare terms that are involved in an update, in both the original and new vector, we need to adjust the replacement vector. We
will refer to these terms as affected (rare) terms. Mathematically, the set of affected terms $A_T$ is defined as

$$A_T = \{ t \in E' \mid t \in T(U) \setminus (P \cup Q) \}. \quad (25)$$

Dually, all documents that are not updated, but contain the affected, promoted, and demoted terms in the old and/or new term frequency vectors are referred to as affected documents. The set $A_D$ of affected documents is defined as

$$A_D = \{ d_j \in D \mid \mathcal{T}(d_j) \cap (E \cup E') \neq \emptyset \wedge (d_j, v) \notin U \}. \quad (26)$$

For promoted terms, we need to remove the old replacement vector in $R$ from the reduced corpus $\tilde{C}$. We can then set the replacement vector to the appropriate standard base $e_i$.

For demoted terms, we have to compute a new replacement vector from scratch. All documents that contain these have to contribute their common terms to the weighted centroid. Additionally, we have to add the newly built replacement vector to all of these documents’ reduced representation in a second pass.

For both promoted and demoted terms, we need to adjust the reduced-dimensional space. We compact the rows of the reduced space to eliminate demoted terms, and append new, zero rows for the promoted terms. To this end, we construct an index permutation $\pi$ that is used in the compaction procedure to map new to old rows in the compacted space, and a new, global index permutation $\pi'$ that maps all features in $T$ to the new, reduced row space of $k'$ common terms.

For affected terms, we first need to subtract the old replacement vector from the reduced corpus. We refer to this as downdating the reduced corpus, and it requires a sweep over all documents. Additionally, we need to (downdate and) update the replacement vectors by subtracting old and adding new document vectors. Downdating the reduced corpus and updating the replacement vectors can be performed concurrently by keeping a copy of the old vectors in $R$ and building the updated replacement vectors in a separate matrix $R'$. Likewise, the unreduced corpus matrix $C$ can only be modified after all downdating has been completed. To update the reduced corpus $\tilde{C}$, we need to complete the build of $R'$, requiring a separate pass over all documents.

### 2.1 Updating Replacement Vectors

There are some auxiliary definitions that we will require to describe the update process. We need to determine the set of all updates $(j, v)$ that involve a particular term $t_i$, defined as $U(t_i) = \{ (j, v) \in U \mid |\text{new}(j, v)| + |\text{old}(j, v)| \neq 0 \}$. \quad (27)

Similarly, we require the set of all documents $\mathcal{D}(U)$ that are affected by an update $U$, formally

$$\mathcal{D}(U) = \{ d_j \in D \mid (j, v) \in U \}. \quad (28)$$

To eliminate demoted terms from the reduced space compact the rows we define a matrix row permutation operator $\Pi$. For a matrix $A \in \mathbb{R}^{m \times n}$, it is defined as

$$\Pi^\sigma(A) = A' \in \mathbb{R}^{m' \times n}, \quad (29)$$

$$A'_{i,j} = A_{\sigma(t_i),j}. \quad (30)$$

For notational convenience, let us assume that truncation of the new elimination terms $\tau_{E'}$ respects the new index order established with $\sigma$, i.e., the implementation uses the global index permutation $\pi'$ mapping all new features to indices in $\{1, \ldots, k'\}$.

For all promoted terms $(\forall t_i \in P)$, we simply set the replacement vector to an appropriate standard base vector,

$$\rho'_i = \tau_{E'}(e_i). \quad (31)$$

For all demoted terms $(\forall t_i \in Q$), we need to compute a replacement vector from scratch.

$$\chi'_i = \sum_{d_j \in \mathcal{D}(t_i) \setminus \mathcal{D}(U)} |C_{i,j}| + \sum_{(j,v) \in U(t_i)} |\text{new}(j, v)|. \quad (32)$$

$$\rho'_i = \chi'^{−1}_i \sum_{d_j \in \mathcal{D}(t_i) \setminus \mathcal{D}(U)} C_{i,j} \tau_{E'}(C_{1:m,j}) + \sum_{(j,v) \in U(t_i)} \text{new}(j, v) \tau_{E'}(\text{old}(j, v)). \quad (33)$$

For all affected terms $(\forall t_i \in A_T$), we need to downdate old vectors and update new vectors. To downdate the replacement vectors to account for changing or deleted documents, we have to subtract all old, truncated document vectors. To this end, we accumulate them in a downdate vector,

$$\text{DV}_i = \sum_{(j,v) \in U(t_i)} \text{old}(j, v) \tau_{E'}(\text{old}(j, v)). \quad (34)$$
Conversely, we must update all changing or added documents, and accumulate them in a dual update vector,

$$ U V_i = \sum_{(j, v) \in U(t_i)} \text{new}(j, v) \tau_E \left( \text{new}(j, v) \right). $$  \hspace{1cm} (35)

For the promoted terms, we have to analyze all invariant documents and update vectors and compute a promoted term vector $P V_i \in \mathbb{R}^{(k' - k'') \times n}$,

$$ P V_i = \sum_{d_j \in D(t_i) \setminus D(U)} C_{l, j} \tau_{E \cup A_T}(C_{l, m, j}) + \sum_{(j, v) \in U(t_i)} \text{new}(j, v) \tau_{E \cup A_T} \left( \text{new}(j, v) \right). $$ \hspace{1cm} (36)

We analogously compute an updated scaling factor $\lambda'$,

$$ \lambda'_i = \lambda_i + \sum_{(j, v) \in U(t_i)} \left| \text{new}(j, v) \right| - \left| \text{old}(j, v) \right|. $$ \hspace{1cm} (37)

With these definitions, we can formally describe the update process for the replacement vector as an assembly of a permuted document/update of the old vector and the part for the promoted terms on rows $k'' + 1$ to $k'$,

$$ \rho'_i = \frac{1}{\lambda'_i} \left[ \sum_{j \in C'} \left( \lambda_j \rho_i - D V_i + U V_i \right) \right]. $$ \hspace{1cm} (38)

All other terms were and remain common terms, and we only have apply the permutation $\sigma$ and pad them with zeroes,

$$ \rho'_i = \left[ \sum_{j \in C'} \left( \rho_i \right) \right]_0 \in \mathbb{R}^{k' \times m}. $$ \hspace{1cm} (39)

### 2.2 Updating the Reduced Corpus

Now that we can update the replacement vectors to obtain $R'$, let us examine how we can update the dimensionality-reduced corpus matrix and compute $C'$. For reasons of space, we will give a brief overview instead of the complete derivation, which can be constructed analogously to the derivation of updated replacement vectors in the previous section.

For **deleted documents** $(j, v) \in U$, we simply delete the associated column.

For **inserted documents** $(j, v) \in U$, we need to insert a new column into $C'$ with a reduced representation that has been compute by taking the product $R'v$.

1. delete or append terms in $T, C, N, R, \lambda$; $N' := N$; $E' := E$.
2. for $u \in U$ do
   - Used := Used $\cup T(u)$;
   - For $t_i \in (T_o(u) \setminus T_n(u))$ do $N'_t :=$;
   - For $t_i \in (T_n(u) \setminus T_o(u))$ do $N'_t :=$;
3. for $t_i \in T$ do
   - if $(N'_t \geq \delta) \land t_i \in E$ then
     - $P := P \cup \{t_i\}; E' := E' \setminus \{t_i\}$;
   - else if $(N'_t \leq \delta) \land t_i \notin E$ then
     - $Q := Q \cup \{t_i\}; E' := E' \cup \{t_i\}$;
   - else if $(N'_t \leq \delta) \land t_i \in \text{Used}$ then
     - $A_T := A_T \cup \{t_i\}$;
4. $\sigma := 1; j' := 1; l := k' + 1$.
   - for $t_i \in T$ do
     - if $t_i \in P$ then $\pi'(i) := l ++$;
     - else if $t_i \in A_T$ then
       - $\sigma(j) := \pi(i)$;
       - $\pi'(i) := j ++$;
     - else $\pi'(i) := -1$;

**Algorithm 1:** Preparing the Update

For **modified documents** $(j, v) \in U$, we proceed analogously and recompute the column vector from scratch.

For **affected documents** $d_j \in A_D$, we need to permute the original reduced document vector, before we downgrade the old replacement vectors and update the new replacement vectors including promoted terms. Formally, this follows directly from the definition of the reduced document vectors in Statement 3, and it can be derived analogously to the update of replacement vectors in Statement 38.

### 3 The Updating Algorithm

An effective algorithm should process the corpus sequentially. We believe that two passes over all documents are required. In the first pass, we update the new replacement vectors by subtracting old and adding new documents, and downgrade the reduced corpus by subtracting all changing replacement vectors. In the second pass, we add the new replacement vectors to the new reduced corpus. The update will need to be processed three times: once to prepare the update, and twice more together with the documents. We can subsume these activities in three phases.
1 for \( t_i \in A_T \) do \( R_{1:k',i} * = \lambda_i \); 
\( R' = \text{Compact}(R, k', k'', m, \sigma) \); 
for \( t_i \in Q \) do \( R'_{1:k',i} = 0 \);
2 for \( d_j \in D \) do 
\begin{align*}
&\text{for } t_i \in T(d_j) \text{ do} \\
&\quad \text{if } (j, d'_j) \in U \land t_i \in A_T \text{ then} \\
&\quad \quad \text{for } t \in \{1, \ldots, k''\} \text{ do} \\
&\quad \quad \quad R'_{t,i} = C_{t,i} \tau_{E'}(C_{*j}) \sigma(t); \\
&\quad \quad \lambda_i * = |C_{i,j}|; \\
&\quad \text{if } (j, d'_j) \notin U \text{ then} \\
&\quad \quad \text{if } t_i \in A_T \cup P \text{ then} \\
&\quad \quad \quad \hat{C}_{1:k,j} = C_{1,j} R_{1:k,j}; \\
&\quad \quad \quad \text{if } t_i \in Q \text{ then} \\
&\quad \quad \quad R'_{1:k',j} = C_{i,j} \tau_{E'}(C_{*j})_{1:k'}; \\
&\quad \quad \lambda_i * = |C_{i,j}|; \\
&\quad \quad \text{else if } t \in E' \text{ then} \\
&\quad \quad \quad R'_{k',k,\nu} = C_{i,j} \tau_{E'}(C_{*j})_{k',k}; \\
&\quad \quad \text{if } (T(d_j) \cap E') \subseteq P \text{ then} \\
&\quad \quad \quad \lambda_i * = |C_{i,j}|; \\
\end{align*}
3 for \((j, v) \in U \) do 
\begin{align*}
&\text{if } j = \nu \text{ then} \\
&\quad \text{for } t_i \in T_v(u, v) \cap E' \text{ do} \\
&\quad \quad R'_{1:k',i} = C_{i,j} \tau_{E'}(v)_{1:k'}; \\
&\quad \lambda_i * = |v_j|; \\
&\quad \text{else} \\
&\quad \quad \text{for } t_i \in \{t_i \in T \mid v_i \neq 0\} \text{ do} \\
&\quad \quad \quad \text{if } t_i \in Q \cup A_T \text{ then} \\
&\quad \quad \quad \quad R'_{1:k',i} = v_i \tau_{E'}(v)_{1:k'}; \\
&\quad \quad \quad \lambda_i * = |v_j|; \\
\end{align*}
4 for \( t_i \in E' \) do 
\begin{align*}
&\text{if } t_i \in P \text{ then} \\
&\quad R'_{1:k',i} = \tau_{E'}(e_i); \\
&\quad \text{else} \\
&\quad R'_{1:k',i} /= \lambda_i; \\
\end{align*}

Algorithm 2: Replacement Vector Update

<table>
<thead>
<tr>
<th>Algorithm 1: Preparing the update</th>
<th>Algorithm 2: Updating the replacement vectors and downdating the reduced corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>We analyze the batch update, prepare the various sets of features and documents, and compute an index permutation used to compact the old common terms and make space for newly promoted terms.</td>
<td>We update the replacement vectors by subtracting the old and adding new documents, and subtract the old replacement vectors from the reduced corpus.</td>
</tr>
</tbody>
</table>

Algorithm 3 - Updating the reduced corpus: Now that the new replacement vectors are available, we can add them to the reduced representation of all documents that contain them.

The analysis and preparation of the update is stated in Algorithm 1.

Line 1.1: To prepare the update, we need to integrate new terms and initialize \( N' \) and \( E' \).

Line 1.2: Then, we update the occurrence counts and determine the terms \( T(U) \).

Line 1.3: We can now determine the sets of promoted, demoted, and affected terms.

Line 1.4: Lastly, we build the index permutations.

We also need an auxiliary function \texttt{Compact} to apply this permutation.

\texttt{Compact}(A, m', m'', n, \sigma)

\( A' := 0 \in \mathbb{R}^{m' \times n} \); 
for \( j \in \{1, \ldots, n\} \) do 
\begin{align*}
&\quad \text{for } i \in \{1, \ldots, m''\} \text{ do} \\
&\quad \quad A'_{t,i} := A_{\sigma(i),j}; \\
&\quad \text{return } A'; \\
\end{align*}

We now update the replacement vectors and downdate the reduced document vectors, as shown in Algorithm 2.

Line 2.1: We begin by denormalizing and compacting the replacement vectors.

Line 2.2: Now, we downdate affected terms, and start building demoted terms.

Line 2.3: After scanning the corpus, we analyze all updates and add their contributions to the new replacement vectors.

Line 2.4: Lastly, we insert truncated unit vectors for all promoted terms and re-normalize the replacement vectors.

Now that we have updated the replacement vectors, we can use them to update the reduced corpus, as shown in Algorithm 3.

Line 3.1: We begin by compacting the original reduced vectors.

Line 3.2: We can then add all new replacement vectors to the documents that contain them, and rebuild any documents that have changed.

Line 3.3: For all deletions, we can simply remove the associated column.

Line 3.4: When adding documents, we need to compute the reduced vector from scratch.

Line 3.5: Updating the sparse corpus completes the update algorithm.
\[ \hat{C}' = \text{Compact}(\hat{C}', k', n, \sigma); \]
\[ \text{Old} = E' \setminus (P \cup A_T \cup Q); \]
\begin{enumerate}
\item We compact the replacement vectors in \( O(mk') \).
\item To downdate the reduced corpus and update the replacement vectors, we need to scan all \( \|D\| + \|U\| \) document and update vectors. For all expected \( \text{nnz} \) occurring terms, we need to add or subtract the \( k' \)-dimensional vector to or from the replacement vectors in \( R' \) or the reduced corpus \( \hat{C} \), yielding a complexity of \( O((\|D\| + \|U\|) \text{nnz} k') \).
\item We compact the corpus matrix in \( O(\|D\| k'). \)
\item Updating the reduced corpus requires one sweep over all \( \|D\| + \|U\| \) documents, where all expected \( \text{nnz} \) replacement vectors of length \( k' \) are added to the reduced corpus. The complexity for this step is also \( O((\|D\| + \|U\|) \text{nnz} k'). \)
\end{enumerate}

Thus, application of an update has a total time complexity of
\[ O((\|D\| + \|U\|) \text{nnz} k' + mk'). \]

Assuming that the update is smaller than the corpus, i.e., \( \|U\| < \|D\| \), and that the number of documents is greater than the number of terms, i.e., \( n > m \), the joint complexity of preparing and applying an update can be simplified to
\[ O((\|D\| + \|U\|) \text{nnz} k' + mk'). \]

Table 1 depicts some empirical measurements using an Intel i5-2557 with 4 GB or RAM running Max OS X 10.7.5 on the 23,149 documents in the pre-vectorized Reuters Corpus Volume I, version 2 [15]. We selected 12.5%–50% of the document at random as updates, and used the remaining 50%–87.5% as the initial corpus. Averaged over ten measurements, the updating algorithm outperforms a complete rebuild with the original construction algorithm by a factor of 3.18–4.49. As one might expect, users with small but frequent updates can benefit the most from an updating algorithm. A theoretic worst-case performance overhead of updating is approximately a factor of two if all documents in the corpus change, because the algorithm will first downdate old vectors by subtracting them.
5 Summary & Conclusions

In this paper, we have presented a formal framework that can be used to discuss and rigorously define the updating process for rare term vector replacement. We have modeled three document updating operations – adding, changing, and deleting documents – and described how these affect the term occurrence counts, the common and rare term sets, and the size and meaning of the reduced space. Based on this basic model, we have discussed the process of updating the replacement vectors and the reduced corpus. We have then derived an efficient algorithm that requires only two sequential scans over the corpus, and three passes over the update. It interleaves all update and downdate operations in these passes, and accesses the document vectors in a purely sequential manner. Random access is required only for the replacement vectors. We have given a detailed complexity analysis, and have presented an empirical performance evaluation based on the Reuters newswire corpus. The observed speed-up demonstrates that the updating algorithm is highly competitive with a complete rebuild. In our future research, we intend to parallelize the batch algorithm and study the use of approximate updating algorithms.

References


