Report No. UIUCDCS-R-2006-2715 **UILU-ENG-2006-1747**

Tensor Space Model for Document Analysis

by

Deng Cai, Xiaofei He, and Jiawei Han

April 2006

Tensor Space Model for Document Analysis*

Deng Cai† Xiaofei He‡ Jiawei Han†

† Department of Computer Science, University of Illinois at Urbana-Champaign

‡ Yahoo! Research Labs

Abstract

Vector Space Model (VSM) has been at the core of information retrieval for the past decades. VSM considers the documents as vectors in high dimensional space. In such a vector space, techniques like Latent Semantic Indexing (LSI), Support Vector Machines (SVM), Naive Bayes, etc., can be then applied for indexing and classification. However, in some cases, the dimensionality of the document space might be extremely large, which makes these techniques infeasible due to the *curse of dimensionality*. In this paper, we propose a novel **Tensor Space Model** for document analysis. We represent documents as the second order tensors, or matrices. Correspondingly, a novel indexing algorithm called **Tensor Latent Semantic Indexing** (TensorLSI) is developed in the tensor space. Our theoretical analysis shows that TensorLSI is much more computationally efficient than the conventional Latent Semantic Indexing, which makes it applicable for extremely large scale data set. Several experimental results on standard document data sets demonstrate the efficiency and effectiveness of our algorithm.

1 Introduction

Document indexing and representation has been a fundamental problem in information retrieval for many years. Most of previous works are based on the Vector Space Model (VSM, [9]). The documents are represented as vectors, and each word corresponds to a dimension. Learning techniques such as Latent Semantic Indexing (LSI, [2]), Support Vector Machines (SVM, [10]), Naive Bayes, etc., can be then applied in such a vector space. The main reason of the popularity of VSM is probably due to the fact that most of the existing learning algorithms can only take vectors as their inputs, rather than tensors.

When VSM is applied, one if often confronted with a document space $\mathbb{R}^n$ with a extremely large n. Let $\mathbf{x} \in \mathbb{R}^n$ denotes the document vector. Let us consider $n = 1,000,000$ and learning

* The work was supported in part by the U.S. National Science Foundation NSF IIS-03-08215/IIS-05-13678. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

a linear function $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. In most cases, learning g in such a space is infeasible in the sense of computability. For example, when LSI is applied in such a space, one needs to compute eigen-decomposition of a $1M \times 1M$ matrix [1].

Different from traditional Vector Space Model based document indexing and representation, in this paper, we consider documents as matrices, or the second order tensors. For a document set with n words, we represent the documents as the second order tensors (or, matrices) in $\mathbb{R}^{n_1} \otimes \mathbb{R}^{n_2}$, where $n_1 \times n_2 \approx n$. For examples, a $1,000,000$-dimensional vector can be converted into a 1000×1000 matrix. Let $X \in \mathbb{R}^{n_1} \otimes \mathbb{R}^{n_2}$ denotes the document matrix. Naturally, a linear function in the tensor space can be represented as $f(X) = \mathbf{u}^T X \mathbf{v}$, where $\mathbf{u} \in \mathbb{R}^{n_1}$ and $\mathbf{v} \in \mathbb{R}^{n_2}$. Clearly, $f(X)$ has only $n_1 + n_2$(=2000 in our case) parameters which is much less than $n(= 1,000,000)$ of $g(\mathbf{x})$.

Based on the tensor representation of documents, we propose a novel indexing algorithm called **Tensor Latent Semantic Indexing** (TensorLSI) operated in the tensor space rather than vector space. Sharing similar properties as the conventional Latent Semantic Indexing (LSI) [2], TensorLSI tries to find the principal components of the tensor space. Let $\{\mathbf{u}_i\}_{i=1}^{n_1}$ be a set of basis functions of $\mathbb{R}^{n_1}$ and $\{\mathbf{v}_j\}_{j=1}^{n_2}$ be a set of basis functions of $\mathbb{R}^{n_2}$. It is easy to show that $\{\mathbf{u}_i\mathbf{v}_j^T\}$ forms a basis of $\mathbb{R}^{n_1} \otimes \mathbb{R}^{n_2}$. Thus, TensorLSI aims at finding bases $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_j\}$ such that the projections of the documents onto $\{\mathbf{u}_i\mathbf{v}_j^T\}$ can best represent the documents in the sense of reconstruction error.

It would be important to note that, while searching for the optimal bases $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_j\}$, we need only to compute the eigen-decompositions of two $n_1 \times n_1$ and $n_2 \times n_2$ matrices. This property makes our algorithm particularly applicable for the case when the number of words is extremely large.

The rest of the paper is organized as follows: Section 2 provides a brief description of the algebra of tensors and Latent Semantic Indexing. The *TensorLSI* approach for document indexing and representation is described in Section 3. Section 4 provides some theoretical analysis on the computational complexity of TensorLSI as well as the memory requirement and storage. The experimental results on Reuters-21578 dataset are presented in Section 5. Finally, we provide some concluding remarks and suggestions for future work in Section 6.

2 Preliminaries

The algorithm and analysis presented in this paper is fundamentally based on the algebra of tensors and Latent Semantic Indexing. In this section, we provide a brief overview of them. For a detailed treatment of tensor algebra please see [7].

[1] Note, we assume that the number of documents (m) is larger than n. When $m < n$, it suffices to compute the eigen-decomposition of a $m \times m$ matrix.

2.1 The Algebra of Tensors

A tensor with order k is a real-valued multilinear function on k vector spaces:

$$T : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k} \to \mathbb{R}$$

The number k is called the *order* of T. A multilinear function is linear as a function of each variable separately. The set of all k-tensors on $\mathbb{R}^{n_i}, i = 1, \cdots, k$, denoted by $\mathcal{T}^k$, is a vector space under the usual operations of pointwise addition and scalar multiplication:

$$(aT)(\mathbf{a}_1, \cdots, \mathbf{a}_k) = a\left(T(\mathbf{a}_1, \cdots, \mathbf{a}_k)\right),$$

$$(T + T')(\mathbf{a}_1, \cdots, \mathbf{a}_k) = T(\mathbf{a}_1, \cdots, \mathbf{a}_k) + T'(\mathbf{a}_1, \cdots, \mathbf{a}_k)$$

where $\mathbf{a}_i \in \mathbb{R}^{n_i}$. Given two tensors $S \in \mathcal{T}^k$ and $T \in \mathcal{T}^l$, define a map:

$$S \otimes T : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_{k+l}} \to \mathbb{R}$$

by

$$S \otimes T(\mathbf{a}_1, \cdots, \mathbf{a}_{k+l}) = S(\mathbf{a}_1, \cdots, \mathbf{a}_k)T(\mathbf{a}_{k+1}, \cdots, \mathbf{a}_{k+l})$$

It is immediate from the multilinearity of S and T that $S \otimes T$ depends linearly on each argument $\mathbf{a}_i$ separately, so it is a $(k+l)$-*tensor*, called the tensor product of S and T.

For the first order tensors, they are simply the covectors on $\mathbb{R}^{n_1}$. That is, $\mathcal{T}^1 = \mathcal{R}^{n_1}$, where $\mathcal{R}^{n_1}$ is the dual space of $\mathbb{R}^{n_1}$. The second order tensor space is a product of two first order tensor spaces, i.e. $\mathcal{T}^2 = \mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$. We have the following proposition:

Proposition 1 *Let $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{n_1}$ be the standard basis of $\mathcal{R}^{n_1}$, and $\mathbf{v}_1, \cdots, \mathbf{v}_{n_2}$ be the basis of $\mathcal{R}^{n_2}$. $\{\boldsymbol{u}_i \boldsymbol{v}_j^T\}(i = 1, \cdots, n_1, j = 1, \cdots, n_2)$ forms a basis of $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$.*

Proof Let $< \cdot >$ denote the standard inner product operator for matrices and tr the trace operator. It suffices to show the following:

$$< \mathbf{u}_i \mathbf{v}_j^T, \mathbf{u}_k \mathbf{v}_l^T >= 0, (i, j) \neq (k, l)$$

By simple algebra formulation, we have:

$$\begin{aligned} & < \mathbf{u}_i \mathbf{v}_j^T, \mathbf{u}_k \mathbf{v}_l^T > \\ = \; & tr\left((\mathbf{u}_i \mathbf{v}_j^T)(\mathbf{u}_k \mathbf{v}_l^T)^T\right) \\ = \; & tr\left(\mathbf{u}_i \mathbf{v}_j^T \mathbf{v}_l \mathbf{u}_k^T\right) \\ = \; & \left(\mathbf{v}_j^T \mathbf{v}_l\right) tr\left(\mathbf{u}_i \mathbf{u}_k^T\right) \\ = \; & \left(\mathbf{v}_j^T \mathbf{v}_l\right)\left(\mathbf{u}_i^T \mathbf{u}_k\right) \end{aligned}$$

Since $(i, j) \neq (k, l)$, we have $i \neq k$ or $j \neq l$, either of which makes $< \mathbf{u}_i \mathbf{v}_j^T, \mathbf{u}_k \mathbf{v}_l^T >= 0$.

The above proposition shows that in order to find a basis of $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$, one needs only to find a basis of $\mathcal{R}^{n_1}$ and $\mathcal{R}^{n_2}$.

Let $\mathbf{e}_1, \cdots, \mathbf{e}_{n_1}$ be the corresponding dual basis of $\mathbb{R}^{n_1}$, and $\widetilde{\mathbf{e}}_1, \cdots, \widetilde{\mathbf{e}}_{n_2}$ be the corresponding dual basis of $\mathbb{R}^{n_2}$. We have,

$$\mathbf{u}_i(\mathbf{e}_j) = \delta_{ij} \text{ and } \mathbf{v}_i(\widetilde{\mathbf{e}}_j) = \delta_{ij}$$

where δ_{ij} is the kronecker delta function. For any 2-tensor T, we can write it as:

$$T = \sum_{\substack{1 \le i \le n_1 \\ 1 \le j \le n_2}} T_{ij} \mathbf{u}_i \otimes \mathbf{v}_j$$

This shows that every 2-tensor in $\mathcal{R}^{n_1} \otimes \mathcal{R}^{n_2}$ uniquely corresponds to a $n_1 \times n_2$ matrix.

Note that, in this paper our primary interest is focused on the second order tensors. However, the algebra presented here and the algorithm presented in the next section can also be applied to higher order tensors.

2.2 A Brief Review of Latent Semantic Indexing

Our proposed TensorLSI algorithm is fundamentally based on LSI. In this subsection, we give a brief review of LSI.

LSI is originally motivated to deal with the problem of *synonymy* and *polysemy*. The mathematics behind LSI is the Singular Value Decomposition (SVD). For a rank r term-document matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$, LSI decompose the X using SVD as follow:

$$\mathbf{X} = U \Sigma V^T$$

Where $\Sigma = diag(\sigma_1, \cdots, \sigma_r)$ and $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r$ are the singular values of $\mathbf{X}$, $U = [\mathbf{a}_1, \cdots, \mathbf{a}_r]$ and $\mathbf{a}_i$ is called left singular vectors, $V = [\mathbf{v}_1, \cdots, \mathbf{v}_r]$ and $\mathbf{v}_i$ is called right singular vectors. LSI use the first k vectors in U as the transformation matrix to embed the original document into a k dimensional subspace. It can be easily checked that the column vectors of U are the eigenvectors of $\mathbf{X}\mathbf{X}^T$. Thus, LSI tries to solve the maximum eigenvalue problem:

$$\mathbf{X}\mathbf{X}^T \mathbf{a} = \lambda \mathbf{a} \tag{1}$$

LSI can also be interpreted in terms of reconstruction error minimization. Let $\mathbf{a}$ be the transformation vector and $y_i = \mathbf{a}^T \mathbf{x}_i$. The objective function of LSI can be stated below:

$$\begin{aligned} \mathbf{a}_{opt} &= \arg\min_{\mathbf{a}} \|\mathbf{X} - \mathbf{a}\mathbf{a}^T\mathbf{X}\|^2 \\ &= \arg\max_{\mathbf{a}} \mathbf{a}^T \mathbf{X}\mathbf{X}^T \mathbf{a} \end{aligned}$$

with the constraint

$$\mathbf{a}^T \mathbf{a} = 1$$

It is easy to see that the solutions are the eigenvectors of the matrix $\mathbf{X}\mathbf{X}^T$. More details on theoretical interpretations of LSI using SVD can refer to [3].

3 TensorLSI for Document Indexing and Representation

In this section, we introduce a novel method for document indexing and representation, called TensorLSI.

3.1 Tensor Representation of Documents

In tensor space model, a document is represented as a tensor. Each element in the tensor corresponds to a feature (word in our case). For a document $\mathbf{x} \in \mathbb{R}^n$, we can convert it to the second order tensor (or matrix) $X \in \mathbb{R}^{n_1 \times n_2}$, where $n_1 \times n_2 \approx n$. Figure 1 shows an example of converting a vector to a tensor. There are two issues about converting a vector to a tensor.

The first one is how to choose the size of the tensor, i.e., how to select n_1 and n_2. In figure 1, we present two possible tensors for a 9-dimensional vector. Suppose $n_1 \geq n_2$, in order to have at least n entries in the tensor while minimizing the size of the tensor, we have $(n_1 - 1) \times n_2 < n \leq n_1 \times n_2$. With such requirement, there are still many choices of n_1 and n_2, especially when n is large. Generally all these (n_1, n_2) combinations can be used. However, it is worth noticing that the number of parameters of a linear function in the tensor space is $n_1 + n_2$. Therefore, one may try to minimize $n_1 + n_2$. In other words, n_1 and n_2 should be as close as possible. Moreover, the values of n_1 and n_2 determine the complexity of the learning algorithm which we will discuss later.

The second issue is how to sort the features in the tensor. In vector space model, we implicitly assume that the features are independent. A linear function in vector space can be written as $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Clearly, the change of the order of the features has no impact on the function learning. In tensor space model, a linear function can be written as $f(X) = \mathbf{u}^T X \mathbf{v}$. Thus, the independency assumption of the features no longer holds for the learning algorithms in the tensor space model. Different feature sorting will lead to different learning result in the tensor space model.

In this paper, we empirically sort the features (words) according to their document frequency and then convert the vector into a $n_1 \times n_2$ tensor such that $n_2 = 20$. The better ways of converting a document vector to a document tensor with theoretical guarantee will be left for our future work.

3.2 TensorLSI

TensorLSI is fundamentally based on LSI. It tries to project the data to the tensor subspace in which the reconstruction error is minimized. Given a set of document matrices $X_i \in \mathbb{R}^{n_1 \times n_2}$, $i = 1, \cdots, m$. Let $Y_i = U^T X_i V$ denote its projection in the tensor subspace $\mathcal{U} \otimes \mathcal{V}$. The reconstruction error for X_i can be written as $\|X_i - U Y_i V^T\|$. Thus, the objective function of TensorLSI can be described as follows:

$$\min_{U,V} \sum_{i=1}^{m} \|X_i - U Y_i V^T\|^2 \tag{2}$$

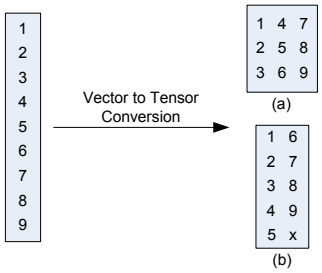

Figure 1: Vector to tensor conversion. 1∼9 denote the positions in the vector and tensor formats. (a) and (b) are two possible tensors. The 'x' in tensor (b) is a padding constant.

which is equivalent to the following:

$$\min_{U,V} \sum_{i=1}^{m} \|X_i - UU^T X_i V V^T\|^2 \tag{3}$$

Let $tr(A)$ denote the trace of matrix A, we have $\|A\|^2 = tr(AA^T)$ and $tr(AB) = tr(BA)$. Thus,

$$\begin{aligned}
& \sum_{i=1}^{m} \|X_i - UU^T X_i VV^T\|^2 \\
= & \sum_{i=1}^{m} tr\left((X_i - UU^T X_i VV^T)(X_i - UU^T X_i VV^T)^T\right) \\
= & \sum_{i=1}^{m} tr\Big(X_i X_i^T - UU^T X_i VV^T X_i^T - \\
& X_i(UU^T X_i VV^T)^T + (UU^T X_i VV^T)(UU^T X_i VV^T)^T\Big) \\
= & \sum_{i=1}^{m} \Big\{tr\Big(X_i X_i^T\Big) - tr\Big(UU^T X_i VV^T X_i^T\Big) - \\
& tr\Big(X_i VV^T X_i^T UU^T\Big) + tr\Big(UU^T X_i VV^T VV^T X_i^T UU^T\Big)\Big\} \\
= & \sum_{i=1}^{m} \Big\{tr\Big(X_i X_i^T\Big) - 2tr\Big(UU^T X_i VV^T X_i^T\Big) + \\
& tr\Big(UU^T X_i VV^T X_i^T UU^T\Big)\Big\} \\
= & \sum_{i=1}^{m} \Big\{tr\Big(X_i X_i^T\Big) - 2tr\Big(U^T X_i VV^T X_i^T U\Big) + \\
& tr\Big(U^T X_i VV^T X_i^T UU^T U\Big)\Big\} \\
= & \sum_{i=1}^{m} tr\Big(X_i X_i^T\Big) - \sum_{i=1}^{m} tr\Big(U^T X_i VV^T X_i^T U\Big) \\
= & \sum_{i=1}^{m} tr\Big(X_i X_i^T\Big) - \sum_{i=1}^{m} tr\Big(V^T X_i^T UU^T X_i V\Big)
\end{aligned}$$

Thus, minimizing the objective function (3) is equivalent to maximizing the following objective functions:

$$\max_U tr\left(U^T M_V U\right), M_V = \sum_{i=1}^{m} X_i VV^T X_i^T \tag{4}$$

$$\max_V tr\left(V^T M_U V\right), M_U = \sum_{i=1}^{m} X_i^T UU^T X_i \tag{5}$$

Adding the constraint that the column vectors of U and V have unit norm, the optimal solutions of U and V are given by solving the following two eigenvector problems:

$$M_V \mathbf{u} = \lambda^u \mathbf{u}$$

$$M_U \mathbf{v} = \lambda^v \mathbf{v}$$

As can be seen, the above two equations are dependent on each other, thus can not be solved independently. By noticing that the matrix M_V is symmetric, it is easy to show that the solutions

$\{\mathbf{u}_i\}(i = 1, \cdots, n_1)$ are orthonormal. That is, U is a orthogonal matrix, $UU^T = U^TU = I$. Likewise, V is also a orthogonal matrix, $VV^T = V^TV = I$. Thus, we have

$$M_V = \sum_{i=1}^{m} X_i X_i^T \tag{6}$$

and

$$M_U = \sum_{i=1}^{m} X_i^T X_i \tag{7}$$

Therefore, the optimal U and V are given by solving the following two eigenvector problems:

$$\left(\sum_{i=1}^{m} X_i X_i^T\right)\mathbf{u} = \lambda^u \mathbf{u} \tag{8}$$

$$\left(\sum_{i=1}^{m} X_i^T X_i\right)\mathbf{v} = \lambda^v \mathbf{v} \tag{9}$$

As we discussed in the previous subsection, there are different kinds of tensors. Actually, a vector $\mathbf{x} \in \mathbb{R}^n$ itself can be regarded as a 2-tensor $\in \mathcal{R}^n \otimes \mathcal{R}^1$. In such situation, the eigen-problem (9) becomes trivial and the eigen-problem (8) reduces to the eigen-problem (1). Thus, TensorLSI reduces to the traditional LSI.

After obtaining the basis vectors $\{\mathbf{u}_i\}$ $(i = 1, \cdots, n_1)$ and $\{\mathbf{v}_j\}$ $(i = 1, \cdots, n_2)$, each $\mathbf{u}_i\mathbf{v}_j^T$ is a basis of the transformed tensor space, and $\mathbf{u}_i^T X_t \mathbf{v}_j$ $(t = 1, \cdots, m)$ is the coordinate of X_t corresponding to $\mathbf{u}_i\mathbf{v}_j^T$ in this tensor space. When we want to keep the first k principle component of document in the transformed tensor space, we need to evaluate the importance of $\mathbf{u}_i\mathbf{v}_j^T$ with respect to i and j.

The way choosing basis functions for the tensor subspace (TensorLSI) is slightly different from that for the vector subspace (LSI). In LSI, we only have one set of transformation vectors $\{\mathbf{a}_i\}$ associated with a set of eigenvalues $\{\lambda_i\}$. The value of λ_i reflects the importance of $\mathbf{a}_i$ in terms of reconstruction error in vector space. In TensorLSI, we have two sets of transformation vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_j\}$. To evaluate the importance of $\mathbf{u}_i\mathbf{v}_j^T$, we still use the objective function (4) while replacing the U and V by $\mathbf{u}_i$ and $\mathbf{v}_j$. Thus, we get

$$f(\mathbf{u}_i, \mathbf{v}_j) = \sum_{t=1}^{m} tr(\mathbf{u}_i^T X_t \mathbf{v}_j \mathbf{v}_i^T X_t^T \mathbf{u}_j) = \sum_{t=1}^{m} (\mathbf{u}_i^T X_t \mathbf{v}_j)^2 \tag{10}$$

$f(\mathbf{u}_i, \mathbf{v}_j)$ reflects the importance of the tensor basis $\mathbf{u}_i\mathbf{v}_j^T$ in terms of reconstruction error. When we want to keep the first k principle component in the transformed tensor space, we sort $f(\mathbf{u}_i, \mathbf{v}_j)$ for all the i and j in decreasing order and choose the first k pairs.

4 Complexity Analysis

In this section, we provide complexity analysis in time, memory and storage for TensorLSI.

4.1 Time Complexity Analysis

The major computations in TensorLSI are listed below:

- The calculation of M_V in Equation (6) costs $O(m \times n_1^2 \times n_2)$; and the calculation of M_U in Equation (7) costs $O(m \times n_2^2 \times n_1)$.
- The eigen-decomposition of $M_V \in \mathbb{R}^{n_1 \times n_1}$ takes $O(n_1^3)$ [4]; and the eigen-decomposition of $M_U \in \mathbb{R}^{n_2 \times n_2}$ takes $O(n_2^3)$.
- Computing $f(\mathbf{u}_i, \mathbf{v}_j)$ for all $i = 1, \cdots, n_1$ and all $j = 1, \cdots, n_2$ in Equation (10) costs $O(n_1 \times n_2 \times m \times (n_1 + n_2 + 1))$.
- Sorting $n_1 \times n_2$ values of $f(\mathbf{u}_i, \mathbf{v}_j)$ needs $O\Big((n_1 \times n_2)\log(n_1 \times n_2)\Big)$.

Notice that $n_1 \times n_2 \approx n$, the total complexity of TensorLSI can be simplified as

$$\begin{aligned} &O(mnn_1 + mnn_2 + n_1^3 + n_2^3 + mn(n_1 + n_2 + 1) + n\log n) \\ &\quad = O(n_1^3 + n_2^3 + mn(2n_1 + 2n_2 + 1) + n\log n) \end{aligned}$$

For square tensors, we have $n_1 \approx n_2 \approx \sqrt{n}$, thus the total complexity of TensorLSI can be simplified as $O(mn^{1.5})$.

The time complexity of traditional LSI is $O\left((m+n)q^2\right)$, where $q = \min(m, n)$ [6].

For a large scale data set, the document number m can be bigger than n. Thus the time complexity of LSI becomes $O(mn^2 + n^3)$. Under this situation, TensorLSI achieves a significant improvement over LSI in time complexity.

For rectangular tensors, that is, we no longer have $n_1 \approx n_2 \approx \sqrt{n}$, the time complexity of TensorLSI will be larger that that of square tensor. However, in most cases, it is still faster than LSI. Our experiments will show this.

4.2 Memory Requirement

The memory requirement of the TensorLSI algorithm can be analyzed through Equation (6, 7, 8, 9 and 10). The eigen-decomposition steps of Equation (8) and (9) only involve a single matrix with dimension $n_1 \times n_1$ or $n_2 \times n_2$. The most space consuming steps are in Equation (6, 7 and 10), where all the X_i are involved. However, all these equations can be computed incrementally by reading X_i sequentially. Thus the required memory for TensorLSI can be as low as $O(\max(n_1, n_2) \times \max(n_1, n_2))$. For square tensors, it is $O(n)$.

Comparing to traditional LSI, in which the eigen-decompo-sition step requires at least $O(q \times q), q = \min(m, n)$ memory, TensorLSI achieves a major advantage by computing the optimal solutions without requiring all data points in the memory.

Table 1: Complexity Comparison of TensorLSI and LSI

	Time complexity	Minimum memory	Storage size
TLSI	$O(mn^{1.5})$	$O(n)$	$O(2n)$
LSI	$O((m+n)q^2)$	$O(q^2)$	$O(kn)$

n is the number of features and m is the number of documents

$q = \min(m, n)$

k is usually around several hundreds

4.3 Storage Size

The transformation vectors, $\mathbf{a}_1, \cdots, \mathbf{a}_k$ in LSI and U, V in TensorLSI, should be stored for future use. When a query or a new document comes, we should apply these transformation vectors on them to do search or classification.

In TensorLSI, the storage size for U and V is $O(n_1^2 + n_2^2)$, particularly, $O(2n)$ for square tensors. In LSI, the storage size for transformation vectors will be $O(kn)$, where k is the number of reduced dimension. In reality, k is around several hundreds. Thus, TensorLSI achieves significant storage advantage over LSI.

5 EXPERIMENTS

In this section, several experiments were performed on Reuters-21578 to show the effectiveness of our proposed algorithm. We compared our proposed algorithm TensorLSI with traditional LSI both on accuracy and efficiency.

Both LSI and TensorLSI algorithms are implemented in Matlab 7.04. The eigen-problems in LSI and TensorLSI are solved by using *eig* function in Matlab 7.04.

5.1 Data Corpora

Reuters-21578 corpus[2] contains 21578 documents in 135 categories. The ModApte version of Reuters-21578 is used in our experiments. Those documents with multiple category labels are discarded, and the categories with more than 10 documents are kept. It left us with 8213 documents in 41 categories as described in Table 2. For ModeApte split, there are 5899 training documents and 2314 testing documents. Note that, the training testing split is only used in the last classification experiment.

Each document is represented as a term-frequency vector and each document vector is normalized to 1. We simply removed the stop words, and no further preprocessing was done.

[2]Reuters-21578 corpus is at `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

Table 2: 41 semantic categories from Reuters-21578 used in our experiments

	ModeApte			**ModeApte**	
category	Train	Test	**category**	Train	Test
earn	2673	1040	ipi	27	9
acq	1435	620	nat-gas	22	11
crude	223	98	veg-oil	19	11
trade	225	73	tin	17	10
money-fx	176	69	cotton	15	9
interest	140	57	bop	15	8
ship	107	35	wpi	12	8
sugar	90	24	pet-chem	13	6
coffee	89	21	livestock	13	5
gold	70	20	gas	10	8
money-supply	70	17	orange	12	6
gnp	49	14	retail	15	1
cpi	45	15	strategic-metal	9	6
cocoa	41	12	housing	13	1
alum	29	16	zinc	8	4
grain	38	7	lumber	7	4
copper	31	13	fuel	4	7
jobs	32	10	carcass	6	5
reserves	30	8	heat	6	4
rubber	29	9	lei	8	2
iron-steel	26	11			

5.2 Similarity Evaluation

5.2.1 Data Preparation

From the <title> field of 300 TREC ad hoc topics (topic 251∼550), we chose 30 keywords that appear in our data collection with highest frequencies, say, q_i $(i = 1, 2, \cdots, 30)$. For each keyword q_i, let D_i denote the set of the documents containing q_i. Let $D = D_1 \cup \cdots \cup D_{30}$. Finally, we get 30 document subsets and each subset contains multiple topics. Note that, these subsets are not necessarily disjoint. The numbers of documents of these 30 document subsets ranged from 99 to 823 with an average of 316, and the number of topics ranged from 6 to 39 with an average of 24 (Table 3). The reason for generating such 30 document subsets is to split the data collection into small subsets so that we can compare our algorithm TensorLSI to LSI on each subset. In fact, the keywords can be thought of as queries in information retrieval. Thus, the comparison can be thought of as being performed on different queries [1][5].

5.2.2 Similarity Measure

The accuracy of similarity measure plays a crucial role in most of the information processing tasks, such as document clustering, classification, retrieval, etc. In this subsection, we evaluate the accuracy of similarity measure using TensorLSI and LSI. The similarity measure we used is the cosine similarity. In this experiment, we do not distinguish the training and testing data.

For the original document set D, we compute its lower dimensional representations D_{TLSI} and D_{LSI} by using TensorLSI and LSI respectively. Similarly, D_{TLSI} consists of 30 subsets, $D_{TLSI,1}$, $\cdots$, $D_{TLSI,30}$. D_{LSI} also consists of 30 subsets, $D_{LSI,1}$, $\cdots$, $D_{LSI,30}$.

For each document subset D_i (or, $D_{TLSI,i}$, $D_{LSI,i}$), we evaluate the similarity measure between the documents in D_i. Intuitively, we expect that similarity should be higher for any document pair related to the same topic (intra-topic pair) than for any pair related to different topics (cross-topic pair). Therefore, we adopted the average precision used in TREC, regarding an intra-topic pair as a relevant document and the similarity value as the ranking score. Specifically, we denote by p_i the document pair which has the i-th highest similarity value among all pairs of documents in the document set D_i. For each intra-topic pair p_k, its precision is evaluated as follows:

$$precision(p_k) = \frac{\#\text{ of intra-topic pairs } p_j \text{ where } j \leq k}{k}$$

The average of the precision values over all intra-topic pairs in D_i was computed as the average precision of D_i. Note that, the definition of precision we used here is the same as that used in [1][5].

5.2.3 Results

The experimental results on similarity are reported in this subsection. We compared TensorLSI (corresponding to document set D_{TLSI}) to LSI (corresponding to document set D_{LSI}) and the original document representation (corresponding to document set D as baseline algorithm).

In general, the performance of TesnorLSI and LSI varies with the number of dimensions. We showed the best results obtained by them. For each document subset, Figure 2 listed the average precision by using the baseline algorithm and the precision improvement by using LSI and TensorLSI. The computation time of LSI and TensorLSI are also listed in the right part of Figure 2. As can be seen, Both of LSI and TensorLSI achieved better precision than baseline. From precision point of view, LSI and TensorLSI are comparable. However, TensorLSI is much faster than LSI. We perform the statistical significance t-test on the results, which are shown in Table 4. The t-test results tell us that both TensorLSI and LSI are significantly outperform baseline. The TensorLSI and LSI are comparable with respect to precision while TensorLSI is significantly faster than LSI.

Table 3: 30 document subsets

Query	Num of Doc	Query	Num of Doc
agreement	802	industry	428
bank	823	investment	546
computer	134	levels	217
control	235	natural	110
country	178	petroleum	228
domestic	278	prices	657
economy	201	production	426
energy	232	rates	329
exploration	99	security	108
export	307	services	255
exports	343	systems	215
foreign	505	talks	294
formed	103	trade	636
germany	152	washington	137
home	121	world	388

Table 4: T-test for similarity evaluation

	sysA	sysB	t-test
Precision	LSI	Baseline	$\gg$
	TLSI	Baseline	$\gg$
	TLSI	LSI	$\sim$
Time	TLSI	LSI	$\ll$

"$\gg$" or "$\ll$" means P-value ≤ 0.01

"$>$" or "$<$" means $0.01 <$ P-value ≤ 0.05

"$\sim$" means P-value > 0.05

5.3 Clustering Evaluation

Document clustering is one of most crucial techniques to organize the documents in an unsupervised manner. In this subsection, we investigate the use of indexing algorithms for document clustering.

We chose k-means as our clustering algorithm and compared three methods. These three methods are listed below:

- k-means on original term-document matrix (Baseline)
- k-means after LSI (LSI)
- k-means after TensorLSI (TLSI)

5.3.1 Evaluation Metric

The clustering result is evaluated by comparing the obtained label of each document with that provided by the document corpus. The accuracy (AC) is used to measure the clustering performance [11]. Given a document $\mathbf{x}_i$, let r_i and s_i be the obtained cluster label and the label provided by the corpus, respectively. The AC is defined as follows:

$$AC = \frac{\sum_{i=1}^{n} \delta(s_i, map(r_i))}{n}$$

where n is the total number of documents and $\delta(x, y)$ is the delta function that equals one if $x = y$ and equals zero otherwise, and map(r_i) is the permutation mapping function that maps each cluster label r_i to the equivalent label from the data corpus. The best mapping can be found by using the Kuhn-Munkres algorithm [8].

5.3.2 Clustering Results

The evaluations were conducted with different number of clusters, ranging from 2 to 10. For each given cluster number k, 50 tests were conducted on different randomly chosen categories, and the average performance was computed over these 50 tests. For each test, k-means algorithm was applied 10 times with different start points and the best result in terms of the objective function of k-means was recorded.

After LSI or TensorLSI, how to determine the dimensions of the subspace to perform clustering is still an open problem. In our experiments, for each k, we randomly select 5 cases from the 50 tests. On these 5 selected cases, we iterate the dimension from 1 to 50 and get the dimension corresponding to the best clustering accuracy. Then, all other 45 cases used this dimension.

Table 5 shows the average accuracy of baseline, LSI and TensorLSI. The running time on k-means of baseline, LSI and TensorLSI are also recorded[3]. After dimension reduction, one would expect the running time on k-means be shorter. Meanwhile, the dimension reduction step will also cost time. Table 6 shows t-tests result on the clustering accuracy and k-means time for different methods. Both LSI and TensorLSI are significantly better than baseline with respect to both clustering accuracy and computation time. For $k = (2, 3, 4, 5, 6)$, LSI and TensorLSI achieved almost same clustering accuracy. For $k = (8, 9, 10)$. Clustering using LSI is slightly better than clustering using TensorLSI. However, in all cases, the computation time (time on dimension reduction plus time on k-means) of TensorLSI is extremely shorter than that of LSI. Figure 3 and 4 clearly show this.

The experiments show that the dimension reduction is necessary for clustering from consideration of both speed and accuracy.

[3]Note that, the time recorded here is the average over 10 k-means.

5.4 Classification Evaluation

The classification performance is evaluated by comparing the predicted label of each testing document with that provided by the document corpus. Besides the ModApte split of the corpus, which contains around 70% of training sample, we also tested on $l = (5\%, 10\%, 30\%, 50\%)$ training sample cases. For each given l, we average the results over 10 random splits.

For classification, the kNN algorithm is applied. We tried different values of k $(1, \cdots, 20)$ and found that, in all cases, $k = 4$ achieved the best performance.

5.4.1 Classification Results

Table 7 records the accuracy of baseline, LSI and TensorLSI. The time spent on kNN search of baseline, LSI and TensorLSI are also recorded. After dimension reduction, one would expect the time spent on kNN search will be shorter. Meanwhile, the dimension reduction step will also cost time. For classification, LSI and TensorLSI do not improve the accuracy very much. However, after dimension reduction, the kNN search does speed up a lot. After summing the time spent on dimension reduction and the time on kNN search, we can see the advantage of TensorLSI over LSI. Figure 5 shows the overall time of baseline, LSI and TLSI. As the training sample increases, the LSI spends a lot of time on dimension reduction, which makes the whole time of LSI very long. The TensorLSI is very efficient that in all cases, TensorLSI approach is very fast.

For classification using dimension reduction, we have to store the transformation vectors for transform the new coming (testing) data. Table 7 also records the storage size of transformation vectors of both LSI and TLSI. Figure 6 shows that the storage size of TLSI is significantly smaller than that of LSI.

6 Conclusions

A novel document representation and indexing method has been proposed in this paper, called Tensor Latent Semantic Indexing. Different from conventional LSI which considers documents as vectors, TensorLSI considers documents as the second order tensors, or matrices. Based on the tensor representation, TensorLSI tries to find an optimal basis for the tensor subspace in terms of reconstruction error. Also, our theoretical analysis shows that TensorLSI can be much more efficient than LSI in time, memory and storage especially when the number of documents is larger than the number of words.

There are several interesting problems that we are going to explore in the future work:

1. We empirically construct the tensor in this paper. The better ways of converting a document vector to a document tensor with theoretical guarantee need to be studied.

2. In this paper, we only extended the latent semantic indexing idea in the tensor space model. We expect more learning algorithms in vector space model can be extended to the tensor space model.

References

[1] R.K. Ando. Latent semantic space: Iterative scaling improves precision of inter-document similarity measurement. In *Proc. of ACM SIGIR*, 2000.

[2] 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.

[3] C. H. Ding. A similarity-based probability model for latent semantic indexing. In *Proc. of ACM SIGIR*, 1999.

[4] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 3rd edition, 1996.

[5] Xiaofei He, Deng Cai, Haifeng Liu, and Wei-Ying Ma. Locality preserving indexing for document representation. In *Proc. of ACM SIGIR*, 2004.

[6] Michael T. Heath. *Scientific computing: an introductory survey*. McGraw-Hill, 2nd edition, 2002.

[7] John M. Lee. *Introduction to Smooth Manifolds*. Springer-Verlag New York, 2002.

[8] L. Lovasz and M. Plummer. *Matching Theory*. Akadémiai Kiadó, North Holland, Budapest, 1986.

[9] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for information retrieval. *Communications of the ACM*, 18(11):613–620, 1975.

[10] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[11] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proc. of ACM SIGIR*, 2003.

	Precision (%)			Time (s)	
Query	Baseline	LSI	TLSI	LSI	TLSI
agreement	64.8	**+4.9**	+2.4	8.798	**0.981**
bank	35.7	**+9.7**	+8.3	8.608	**0.848**
computer	70.4	**+5.6**	+5.5	0.076	**0.024**
control	78.6	+0.1	**+0.6**	0.390	**0.066**
country	45.0	+0.7	**+3.7**	0.236	**0.060**
domestic	50.5	**+3.6**	+3.1	0.781	**0.137**
economy	42.7	+0.8	**+4.1**	0.359	**0.091**
energy	58.3	0	**+5.7**	0.373	**0.064**
exploration	56.9	**+17.9**	+8.3	0.041	**0.015**
export	61.7	+2.3	**+4.3**	0.917	**0.125**
exports	43.4	+2.9	**+3.6**	1.220	**0.150**
foreign	40.9	**+8.7**	+7.9	3.097	**0.593**
formed	72.8	0	**+1.9**	0.047	**0.013**
germany	54.3	0	**+2.9**	0.128	**0.042**
home	50.7	**+5.5**	+3.4	0.059	**0.024**
industry	42.8	**+5.1**	+2.8	2.242	**0.291**
investment	67.4	+0.7	**+2.8**	3.184	**0.288**
levels	51.0	0	**+1.5**	0.408	**0.082**
natural	55.2	**+5.8**	+2.7	0.053	**0.020**
petroleum	54.1	+0.3	**+4.5**	0.344	**0.060**
prices	45.5	**+5.3**	+4.1	5.776	**0.845**
production	47.7	+2.4	**+3.8**	2.022	**0.224**
rates	52.5	0	**+0.8**	0.942	**0.118**
security	51.4	+0.2	**+7.7**	0.053	**0.024**
services	52.7	**+5.7**	+5.2	0.461	**0.069**
systems	70.7	+3.3	**+4.1**	0.201	**0.029**
talks	68.5	**+3.6**	+1.6	0.808	**0.115**
trade	55.9	0	**+2.2**	5.305	**0.741**
washington	83.3	0	**+1.9**	0.088	**0.036**
world	55.3	**+4.3**	+3.6	1.818	**0.254**
Ave.	56.0	3.3	**+3.8**	1.628	**0.214**

Figure 2: Best average precision for each document set. For LSI and TLSI, improvements over baseline are shown. Boldface indicates the algorithm with best performance for each category. The two columns on the right list the running time of LSI and TLSI.

Table 5: Performance comparisons on clustering

	Baseline		LSI					TLSI				
k	AC (%)	t_2 (s)	Dim	AC (%)	t_1 (s)	t_2 (s)	t_1+t_2 (s)	Dim	AC (%)	t_1 (s)	t_2 (s)	t_1+t_2 (s)
2	87.1	1.262	4	94.6	0.229	0.011	0.240	5	94.3	0.031	0.010	0.041
3	77.5	6.070	8	83.3	0.421	0.013	0.434	10	83.5	0.051	0.015	0.066
4	73.2	12.24	12	79.0	0.810	0.025	0.835	30	78.5	0.077	0.026	0.103
5	67.1	37.15	15	71.6	1.990	0.026	2.016	30	71.1	0.174	0.051	0.225
6	65.5	54.72	18	69.3	3.684	0.039	3.723	26	68.7	0.310	0.059	0.369
7	62.3	83.42	28	67.7	5.017	0.076	5.092	40	66.1	0.440	0.110	0.550
8	58.2	118.6	22	62.3	6.301	0.084	6.386	50	61.0	0.528	0.165	0.693
9	55.3	249.6	24	59.5	8.621	0.113	8.735	50	57.4	0.812	0.239	1.051
10	54.5	236.2	28	58.7	10.98	0.158	11.13	50	56.4	0.995	0.425	1.420
Ave.	66.7	88.82		71.8	4.228	0.061	4.288		70.8	0.380	0.122	0.502

t_1 indicates time spent on dimensionality reduction using LSI or TLSI

t_2 indicates time spent on k-means clustering

Table 6: T-test for clustering

			k								
	sysA	sysB	2	3	4	5	6	7	8	9	10
AC	LSI	Baseline	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	TLSI	Baseline	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	TLSI	LSI	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\ll$	$<$	$\ll$	$\ll$
t_1+t_2	LSI	Baseline	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	TLSI	Baseline	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	TLSI	LSI	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$

$t_1 = 0$ for Baseline

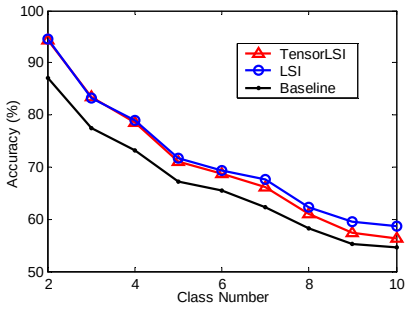

Figure 3: Clustering accuracy with respect to the number of classes. As can be seen, TensorLSI performs comparably with LSI.

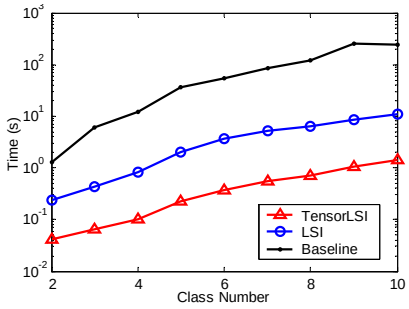

Figure 4: Clustering time with respect to the number of classes. As can been seen, TensorLSI is much faster than LSI.

Table 7: Performance comparisons on classification

Train	Baseline		LSI						TLSI					
	AC (%)	t_2 (s)	Dim	AC (%)	t_1 (s)	t_2 (s)	$t_1 + t_2$	s (MB)	Dim	AC (%)	t_1 (s)	t_2 (s)	$t_1 + t_2$	s (MB)
5%	81.1	123.6	400	82.5	1.069	4.570	5.638	7.620	1900	82.6	0.073	11.219	11.29	0.122
10%	84.3	139.9	800	85.1	6.324	14.10	20.43	23.98	2000	85.3	0.241	16.370	16.61	0.299
30%	87.7	177.2	800	88.3	115.1	27.76	142.8	53.47	2000	88.2	8.719	36.824	45.54	1.467
50%	89.1	177.2	900	89.5	455.1	31.66	486.7	83.13	2000	89.4	30.75	33.473	64.22	2.805
ModApte	88.1	142.3	800	89.6	884.9	26.43	911.3	93.91	1500	89.2	75.62	26.824	102.4	4.527
Ave.	86.1	152.0		87.0	292.5	20.90	313.4	52.42		86.9	23.08	24.942	48.02	1.844

t_1 indicates time spent on dimensionality reduction using LSI or TLSI
t_2 indicates time spent on KNN classification
s indicates storage size of the transformation vectors

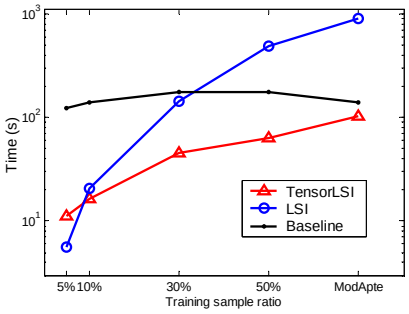

Figure 5: Classification time with respect to the training sample size. As can be seen, the classification time of TensorLSI is less sensitive to the training size.

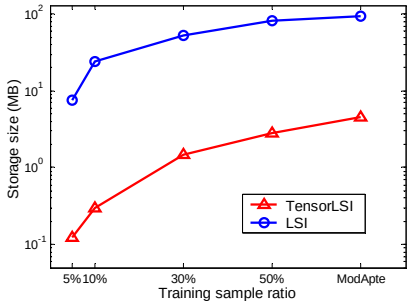

Figure 6: Storage size of transformation vectors with respect to training sample size. As can be seen, TensorLSI requires much less storage than LSI.