Efficient Document Clustering via Online Nonnegative Matrix Factorizations

Fei Wang
Dept. of Statistical Science
Cornell University
Ithaca, NY 14853, USA
fw83@cornell.edu

Chenhao Tan
Dept. of Computer Science
Cornell University
Ithaca, NY 14853, USA
chenhao@cs.cornell.edu

Arnd Christian König
Microsoft Research
Microsoft Coopration
Redmond, WA 98052, USA
chrisko@microsoft.com

Ping Li
Department of Statistical Science
Cornell University
Ithaca, NY 14853, USA
pingli@cornell.edu

Abstract

In recent years, Nonnegative Matrix Factorization (NMF) has received considerable interest from the data mining and information retrieval fields. NMF has been successfully applied in document clustering, image representation, and other domains. This study proposes an online NMF (ONMF) algorithm to efficiently handle very large-scale and/or streaming datasets. Unlike conventional NMF solutions which require the entire data matrix to reside in the memory, our ONMF algorithm proceeds with one data point or one chunk of data points at a time. Experiments with one-pass and multi-pass ONMF on real datasets are presented.

1 Introduction

The recent years have witnessed a surge of interest in document clustering in the field of information retrieval (IR) [6, 41], since document clustering can serve as an important and fundamental technique for automatic topic extraction [34], content summarization [35] and cluster-based information retrieval [31]. Effective document clustering can help automatically organize the document corpus into a meaningful cluster hierarchy for efficient browsing and navigation.

As pointed out by [10], the area of information retrieval spans a wide spectrum ranging from narrow keyword-matching based *search* to broad information *browsing*, e.g., to discover the major recent international events. Conventional document retrieval engines tend to fit well with the *search* end of the spectrum, in that they can provide documents relevant to users' queries. However, they often do not perform well in scenarios where, for example, it is either difficult to formulate a request as a search query or the user is not looking for a specific page but rather wants to obtain a more general overview over (parts of) a text corpus. In such cases, efficient browsing through a good cluster hierarchy would often be helpful. For this reason, there exist a large number of so-called *web clustering engines* which organize search results by topics, offering a complementary view to the flat-ranked lists returned by conventional search engines [8].

Many document clustering methods have been proposed. For example, *hierarchical agglomerative clustering* [16] iteratively builds a bottom-up hierarchy of clustering structures. [12] adapts the traditional K-means algorithm to efficiently handle high-dimensional sparse text data. [18] models documents as vertices and the relationship among documents by hyper-edges and formulates the document clustering problem as a partitioning problem on the hyper-graph. Recently, [38] proposed to first transform the data similarity matrix into a *bi-stochastic* matrix before applying clustering algorithms; and they reported encouraging results.

As an algorithmic tool, *Nonnegative Matrix Factorization* (NMF) [23] has received considerable interest from the data mining and information retrieval fields in recent years. NMF has been applied to document clustering and shows superior results over traditional methods [41, 33]. Moreover, it has been shown that NMF is a relaxed K-means clustering [14], and it also has a close relationship with spectral clustering [13] and probabilistic latent semantic analysis [17].

Well-known NMF solutions such as the multiplicative updates [24] or the projected gradients [29] require to hold the entire data matrix in the memory throughout the solution process. For very large datasets, the resulting space requirement can be prohibitive. Moreover, in modern applications, the data often arrive in a streaming fashion and may not be stored on disk [42].

For very large-scale and/or streaming datasets, similarity-based clustering can often be efficiently solved by *random projections* [25, 26], or by various *sketch/sampling* algorithms, e.g., [27]. In fact, recent papers [37, 36] already applied *random projections* for efficiently computing NMF and Sparse Coding.

The scalability issue can also by addressed by parallelization [30, 39], i.e., by either implementing the clustering technique in a shared-memory multi-node environment or by distributing the computation across large numbers of machines using MapReduce [9].

In this paper, we tackle the scalability and data streaming issue of NMF by developing efficient *online* algorithms based on *stochastic approximations* [21], which process one chunk of data points at a time. Our approach aims to reduce the computational overhead and memory requirements of the clustering algorithm itself, thereby potentially resulting in significant savings in the required hardware and machine-time. For example, utilizing parallelization by distributing the data across machines necessarily means a significant slow-down in processing speed. Instead, if cluster computation can be kept local via a reduction in memory footprint, this may result in very noticeable speed-up.

The ability to incrementally update clusters as new data points arrive is also crucial in the context of modern IR and browsing. Consider a real-life scenario of computing personalized news recommendations [11]. Here, the recommendations are computed using clustering of the news items and users, for very large sets of news documents and users (both on the order of millions) and constant churn of news and new click-information (as news articles are crawled and accessed continuously). In scenarios such as this, an efficient incremental clustering method would certainly be helpful.

Organization Section 2 briefly reviews existing NMF algorithms. Our online NMF algorithm is elaborated in Section 3. Section 4 and Section 5 present the experimental results. Section 6 is devoted to comparing our method with another online NMF algorithm proposed in [7]. Finally Section 7 concludes the paper.

2 A Brief Review of NMF

This section briefly reviews some existing NMF algorithms. We first introduce some basic notations.

Consider a data matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the i-th data point, d is the data dimensionality, n is the number of data points. The goal of NMF is to factorize $\mathbf{X}$ into two low-rank nonnegative matrices $\mathbf{F} \in \mathbb{R}^{d\times r}$ and $\mathbf{G} \in \mathbb{R}^{n\times r}$ by solving

$$(2.1) \qquad \min_{\mathbf{F}\in\mathbb{R}_+^{d\times r}, \mathbf{G}\in\mathbb{R}_+^{n\times r}} \mathcal{L}(\mathbf{F}, \mathbf{G})$$

where $\mathbb{R}_+^{d\times r}$ and $\mathbb{R}_+^{n\times r}$ denote the sets of nonnegative matrices of sizes $d \times r$ and $n \times r$, respectively. $\mathcal{L}(\mathbf{F}, \mathbf{G})$ is some matrix loss function. This paper concentrates on the Frobenius loss:

$$(2.2) \qquad \mathcal{L}(\mathbf{F}, \mathbf{G}) = \left\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\right\|_F^2$$

where $\|\cdot\|_F$ is the matrix Frobenius norm.

In general, solving (2.1) is difficult as the objective $\mathcal{L}(\mathbf{X}, \mathbf{F}, \mathbf{G})$ is not convex with $\mathbf{F}$ and $\mathbf{G}$ jointly. A general solution is to adopt the following alternative block coordinate descent rules [3]:

- Initialize $\mathbf{F}$, $\mathbf{G}$ with nonnegative $\mathbf{F}^{(0)}$, $\mathbf{G}^{(0)}$; $t \leftarrow 0$
- Repeat until a stopping criterion is satisfied:
  Find $\mathbf{F}^{(t+1)}$: $\mathcal{L}(\mathbf{F}^{(t+1)}, \mathbf{G}^{(t)}) \leqslant \mathcal{L}(\mathbf{F}^{(t)}, \mathbf{G}^{(t)})$,
  Find $\mathbf{G}^{(t+1)}$: $\mathcal{L}(\mathbf{F}^{(t+1)}, \mathbf{G}^{(t+1)}) \leqslant \mathcal{L}(\mathbf{F}^{(t+1)}, \mathbf{G}^{(t)})$.

One of the most well-known algorithms for implementing the above rules is Lee and Seung's multiplicative update approach [24], which updates $\mathbf{F}$ and $\mathbf{G}$ by

$$(2.3) \qquad \mathbf{F}_{ij} \longleftarrow \mathbf{F}_{ij}\frac{(\mathbf{X}\mathbf{G})_{ij}}{(\mathbf{F}\mathbf{G}^T\mathbf{G})_{ij}}$$

$$(2.4) \qquad \mathbf{G}_{ij} \longleftarrow \mathbf{G}_{ij}\frac{(\mathbf{X}^T\mathbf{F})_{ij}}{(\mathbf{G}\mathbf{F}^T\mathbf{F})_{ij}}$$

Assuming $r \ll \min(d, n)$, the cost of one-round updating of $\mathbf{F}$ is $O(dnr)$. Similarly, the computational complexity of one-round updating of $\mathbf{G}$ is also $O(dnr)$. Hence the total computational complexity of Lee and Seung's algorithm would be $O(Tndr)$, where T is the number of iterations. At each iteration, Lee and Seung's method needs to hold the data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$ in the memory, at a storage cost of $O(nd)$.

Another algorithm that solves the NMF problem with Frobenius loss (2.2) is *Projected Gradient* method (PGD) [29], which also follows the alternative block coordinate descent rules. Specifically, when we have $\mathbf{F}^{(t)}$ and $\mathbf{G}^{(t)}$, PGD finds $\mathbf{F}^{(t+1)}$ by solving the following nonnegative least square problem

$$(2.5) \qquad \min_{\mathbf{F}\in\mathbb{R}_+^{d\times r}} \left\|\mathbf{X} - \mathbf{F}\left(\mathbf{G}^{(t)}\right)^\top\right\|_F^2$$

which can be efficiently solved via the projected gradient iterations, starting a feasible initialization [3]:

$$\mathbf{F}_{k+1} = P\left[\mathbf{F}_k - \alpha_k \nabla\mathcal{L}\left(\mathbf{F}_k, \mathbf{G}^{(t)}\right)\right] \tag{2.6}$$

where $\nabla\mathcal{L}\left(\mathbf{F}_k, \mathbf{G}^{(t)}\right)$ is the gradient of $\mathcal{L}$ defined in Eq. (2.2) with respect to $\mathbf{F}$ evaluated on $\mathbf{F}_k$, $\mathbf{G}^{(t)}$. $P[u_i]$ is the gradient projection onto the nonnegative constraint set with

$$P[u_i] = \begin{cases} u_i, & \text{if } u_i \geqslant 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

k is the index of the projected gradient iterations, and $\alpha_k > 0$ is the step size determined by the Armijo rule [3].

Similarly, we can apply PGD to update $\mathbf{G}$ with $\mathbf{F}$ fixed. According to [29], the total computational complexity of PGD is $O(Tndr + TKJr^2(d+n))$, where T is the number of iterations, K is the average number of PGD iterations for updating $\mathbf{F}$ or $\mathbf{G}$ in one round, and J is the average number of trials needed for implementing the Armijo rule. Thus, even with small K and J, the time complexity of PGD is still $O(Tndr)$, which is the same as Lee and Seung's approach. Again, in each step, PGD needs to hold $\mathbf{X}$ in the memory and therefore requires a storage cost of $O(nd)$.

For real-world applications with large-scale high-dimension data, where nd is extremely large, it would be difficult to apply a conventional NMF algorithm directly. For example, in Web scale data mining, one may commonly encounter datasets of size $n = O\left(10^{10}\right)$ (or even more) Web pages and each page may be represented using a vector with $d = O\left(10^6\right)$ dimensions (using single words) or 2^{64} dimensions (using *shingles* [6, 15, 28]). Then $n \times d$ will become $O\left(10^{16}\right)$ (single words) or even $O\left(10^{29}\right)$ (shingles), which clearly will not fit in the memory of a single machine, even with sparse representations. In fact, even a smaller corpus like Wikipedia has around 10^7 pages and 10^7 distinct terms.

In comparison, our online approach will process one chunk of data points (e.g., one chunk of Web pages) at a time, incurring much lower memory/IO costs.

3 Online NMF (ONMF)

Our *online NMF (ONMF)* algorithm processes the data in a streaming fashion with low computational and storage complexity. We first describe some insights.

3.1 The Insight

Note that the Frobenius loss (2.2) can be decomposed as

$$\mathcal{L}(\mathbf{F}, \mathbf{G}) = \left\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\right\|_F^2 = \sum\nolimits_{i=1}^n \|\mathbf{x}_i - \mathbf{F}\mathbf{g}_i\|_F^2$$

where $\mathbf{g}_i \in \mathbb{R}^r$ is the i-th column of $\mathbf{G}^{\mathrm{T}}$. Consider a problem of clustering the data set into r clusters. We can view $\mathbf{F} = [\mathbf{f}_1, \cdots, \mathbf{f}_r]$ as the cluster representatives (e.g., they can be cluster centers as in K-means clustering, or cluster concepts as in [40]), and $\mathbf{g}_i = [g_{i1} \cdots, g_{ir}]^{\mathrm{T}}$ as the reconstruction weights of $\mathbf{x}_i$ from those representatives.

Clearly, when $\mathbf{F}$ is fixed, the minimum value of $\mathcal{L}(\mathbf{F}, \mathbf{G})$ can be reached if and only if

$$\mathcal{L}(\mathbf{F}, \mathbf{g}_i) = \|\mathbf{x}_i - \mathbf{F}\mathbf{g}_i\|_F^2 \tag{3.8}$$

is minimized for all i. Thus, we can solve n independent *Nonnegative Least Square* (NLS) problems

$$\min\nolimits_{\mathbf{g}_i \geqslant 0} \|\mathbf{x}_i - \mathbf{F}\mathbf{g}_i\|_F^2 \qquad i = 1, 2, \cdots, n \tag{3.9}$$

and aggregate the solution as $\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_n]^{\mathrm{T}}$.

Intuitively, g_{ij} can be used to measure the possibility that $\mathbf{x}_i$ belongs to cluster j (not probability, as $\sum_j g_{ij} \neq 1$). Consider an extreme case where $\mathbf{f}_k = \mathbf{x}_i$, i.e., the representative of cluster k is $\mathbf{x}_i$ itself, then

$$g_{ij} = \begin{cases} 1, & \text{if } j = k \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

would be an optimal solution for minimizing $\mathcal{L}(\mathbf{F}, \mathbf{g}_i)$. In this sense, computing NMF corresponds to finding the optimal cluster representatives $\mathbf{F}$ and the optimal possibilities that each data point belongs to each cluster.

Based on this observation, we propose the following online NMF (ONMF) framework:

- Initialize the cluster representatives $\mathbf{F}$.
- Repeat until running out data points. At time t
  - Input a data point (or a chunk of data points) $\mathbf{x}^{(t)}$
  - Compute the optimal $\mathbf{g}^{(t)}$ by $\min_{\mathbf{g} \geqslant 0} \mathcal{L}(\mathbf{F}, \mathbf{g})$
  - Update $\mathbf{F}$

Thus, there are two sub-problems in ONMF: (1) how to solve problem (3.9); (2) how to update $\mathbf{F}$.

3.2 Computing Cluster Possibilities G

In this step, we assume the cluster representatives $\mathbf{F}$ is fixed. According to Eq. (3.9), we can solve the following optimization problem for the optimal $\mathbf{g}^{(t)}$

$$\min\nolimits_{\mathbf{g}^{(t)} \geqslant 0} \left\|\mathbf{x}^{(t)} - \mathbf{F}\mathbf{g}^{(t)}\right\|_F^2 \tag{3.11}$$

which is a typical nonnegative least square (NLS) problem. Many algorithms have been proposed to solve the NLS problem. For example, the active set method [22] searches for the optimal active and passive sets by exchanging a variable between the two sets. A drawback of active set methods is that typically only one variable is exchanged between active and passive sets per iteration, making the algorithm slow when the number of variables becomes large [20]. There are other iterative methods such as projected gradient descent [29] as reviewed in Section 2, the quasi-Newton approach [19] and the block principal block pivoting (BPB) algorithm [20].

As an option, one can also add constraints to further restrict $\mathbf{g}^{(t)}$. For example, we can constrain $\sum_j g_j^{(t)} = 1$. In this way, $g_j^{(t)}$ will become to the probability that $\mathbf{x}^{(t)}$ belongs to cluster j. One can also constrain the sparsity of $\mathbf{g}^{(t)}$ as in [32] to reduce the cluster ambiguity.

3.3 Updating the Cluster Representatives F

At time t, as $\mathbf{x}^{(t)}$ arrives, ONMF first solves for $\mathbf{g}^{(t)}$ using $\mathbf{F}^{(t-1)}$, and then updates $\mathbf{F}$ by minimizing the following loss function:

$$\mathcal{L}^{(t)}\left(\mathbf{F}^{(t)}\right) = \sum_{s=1}^{t} \left\|\mathbf{x}^{(s)} - \mathbf{F}^{(t)}\mathbf{g}^{(s)}\right\|_F^2 =$$

$$\sum_{s=1}^{t} tr\left[\left(\mathbf{x}^{(s)}\right)^\top \mathbf{x}^{(s)} - 2\left(\mathbf{x}^{(s)}\right)^\top \mathbf{F}^{(t)}\mathbf{g}^{(s)} + \left(\mathbf{F}^{(t)}\mathbf{g}^{(s)}\right)^\top \mathbf{F}^{(t)}\mathbf{g}^{(s)}\right]$$

Clearly, we can apply the projected gradient descent (PGD) [29] introduced in Section 2 for the optimal $\mathbf{F}^{(t)}$.

The gradient of $\mathcal{L}^{(t)}$ with respect to $\mathbf{F}^{(t)}$ is

$$\nabla_{\mathbf{F}^{(t)}}\mathcal{L}^{(t)}\left(\mathbf{F}^{(t)}\right) = -2\sum_{s=1}^{t}\left[\mathbf{x}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top - \mathbf{F}^{(t)}\mathbf{g}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top\right] \tag{3.12}$$

The Hessian matrix of $\mathcal{L}^{(t)}(\mathbf{F}^{(t)})$ with respect to $\mathbf{F}^{(t)}$ is

$$\mathcal{H}\left[\mathcal{L}^{(t)}\left(\mathbf{F}^{(t)}\right)\right] = 2\sum_{s=1}^{t}\mathbf{g}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top \tag{3.13}$$

For convenience, the following two terms represent the first- and second-order information at time t:

$$\mathbf{V}^{(t)} = \sum_{s=1}^{t}\mathbf{x}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top \tag{3.14}$$

$$\mathbf{H}^{(t)} = \sum_{s=1}^{t}\mathbf{g}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top \tag{3.15}$$

which can be computed incrementally with low storage.

3.3.1 First-Order PGD

The first-order PGD method updates $\mathbf{F}^{(t)}$ by the following rule starting with some initial $\mathbf{F}_0^{(t)}$.

$$\mathbf{F}_{k+1}^{(t)} = P\left[\mathbf{F}_k^{(t)} - \alpha_k \nabla_{\mathbf{F}^{(t)}}\mathcal{L}^{(t)}\left(\mathbf{F}_k^{(t)}\right)\right] \tag{3.16}$$

where α_k is the step size and $P[\cdot]$ is the projection onto the nonnegative constraint set in Eq. (2.7). Bringing Eq. (3.12) into Eq. (3.16) yields

$$\mathbf{F}_{k+1}^{(t)} = P\left[\mathbf{F}_k^{(t)} + 2\alpha_k\sum_{s=1}^{t}\left[\mathbf{x}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top - \mathbf{F}_k^{(t)}\mathbf{g}^{(s)}\left(\mathbf{g}^{(s)}\right)^\top\right]\right] \tag{3.17}$$

There are multiple ways to select the step size α_k [3]. We can use a *constant* step size α, which has to be small enough to guarantee convergence and often results in very inefficient algorithms.

The Armijo rule is popular, which chooses $\alpha_k = \alpha^{t_k}$, where t_k is the first nonnegative integer satisfying

$$\begin{aligned} &\mathcal{L}^{(t)}(\mathbf{F}_{k+1}^{(t)}) - \mathcal{L}^{(t)}(\mathbf{F}_k^{(t)}) \\ &\geq \lambda\left[\nabla_{\mathbf{F}^{(t)}}\mathcal{L}^{(t)}\left(\mathbf{F}_k^{(t)}\right)\right]^\top (\mathbf{F}_{k+1}^{(t)} - \mathbf{F}_k^{(t)}) \end{aligned} \tag{3.18}$$

where $\lambda \in (0, 1)$ is some constant. Armijo-rule based step size can guarantee objective descent at each iteration. Note that the PGD NMF method [29] reviewed in Section 2 adopted the Armijo rule, which also discussed how to choose α and λ.

We implemented the first-order PGD and found the performance is clearly not as good as the second-order PGD methods.

3.3.2 Second-Order PGD

One disadvantage of the first-order PGD is that we need to carefully choose the step size and its convergence could be quite slow. To make the PGD algorithm parameter free with faster convergence, we can make use of the second-order information [2] by utilizing the Hessian (3.13) with the following updating rule:

$$\mathbf{F}_{k+1}^{(t)} = P\left[\mathbf{F}_k^{(t)} - \nabla_{\mathbf{F}_k^{(t)}}\mathcal{L}^{(t)}\left(\mathbf{F}_k^{(t)}\right)\mathcal{H}^{-1}\left[\mathcal{L}^{(t)}\left(\mathbf{F}_k^{(t)}\right)\right]\right] \tag{3.19}$$

where $\mathcal{H}^{-1}\left[\mathcal{L}^{(t)}\left(\mathbf{F}^{(t)}\right)\right]$ is the inverse of the Hessian matrix. Note that the exact calculation of the inverse of the Hessian matrix becomes time-consuming when the number of clusters (r) becomes large. There are two common strategies for approximating the Hessian inverse:

- **Diagonal Approximation** (**DA**) [4]. It only uses the diagonal line of the Hessian matrix to approximate the whole matrix.

- **Conjugate Gradient** (**CG**) [1]. This method exploits the fact that what we really need is the product of a matrix and the inverse of the Hessian.

Algorithm 1 summarizes the detailed procedure of second-order PGD for updating $\mathbf{F}^{(t)}$, where three methods, exact computation, DA and CG, are integrated.

Algorithm 1 SECOND-ORDER PGD FOR UPDATING $\mathbf{F}^{(t)}$

```
Require: $K$, $\mathbf{F}_0^{(t)}, \mathbf{V}^{(t)}, \mathbf{H}^{(t)}$
  for $k = 1 : K$ do
    Compute the gradient $\mathbf{\Delta}_k = \mathbf{V}^{(t)} - \mathbf{F}_{k-1}^{(t)}\mathbf{H}^{(t)}$;
    if Diagonal Approximation (DA) then
      $\mathbf{U} = \mathbf{\Delta}_k \mathrm{diag}^{-1}\left(\mathbf{H}^{(t)}\right) + \mathbf{F}_{k-1}^{(t)}$;
      $\mathbf{F}_k^{(t)} = \max(\mathbf{0}, \mathbf{U})$;
    else if Conjugate Gradient (CG) then
      Solve $\mathbf{Q}$ s.t. $\mathbf{Q}\mathbf{H}^{(t)} = \mathbf{\Delta}_k$ with CG;
      $\mathbf{F}_k^{(t)} = \max(\mathbf{0}, \mathbf{Q} + \mathbf{F}_{k-1}^{(t)})$;
    else
      $\mathbf{F}_k^{(t)} = \max\left(\mathbf{0}, \mathbf{\Delta}_k \left(\mathbf{H}^{(t)}\right)^{-1} + \mathbf{F}_{k-1}^{(t)}\right)$;
    end if
  end for
```

3.4 One-Pass ONMF The complete (one-pass) algorithm procedure is shown in Algorithm 2. Several implementation issues should be highlighted:

- At each time t, we do not need to recompute new $\mathbf{V}^{(t)}$ and $\mathbf{H}^{(t)}$. We only need to compute $\mathbf{x}^{(t)}\left(\mathbf{g}^{(t)}\right)^\top$ and $\mathbf{g}^{(t)}\left(\mathbf{g}^{(t)}\right)^\top$, and add them to the old $\mathbf{V}^{(t-1)}$ and $\mathbf{H}^{(t-1)}$ respectively.

- Our algorithm is very close to the spirit of *Stochastic Gradient Descent* (*SGD*) style methods [5]. There is a compromise model between SGD and traditional batch gradient descent algorithms called *mini-batch implementation* [5], which imports m data points at each step. In this way, the convergence speed of ONMF can be expected to improve considerably. Consequently the updating rules of $\mathbf{V}^{(t)}$ and $\mathbf{H}^{(t)}$ can be given by

$$\begin{aligned}\mathbf{V}^{(t)} &= \mathbf{V}^{(t-1)} + \sum_{i=1}^{m} \mathbf{x}^{(t,i)}\left(\mathbf{g}^{(t,i)}\right)^\top \\ \mathbf{H}^{(t)} &= \mathbf{H}^{(t-1)} + \sum_{i=1}^{m} \mathbf{g}^{(t,i)}\left(\mathbf{g}^{(t,i)}\right)^\top\end{aligned}$$

3.5 Multi-Pass ONMF In data stream applications, often only a single pass over the data is feasible. Nevertheless, many applications still allow multiple passes. The data size may be too large for the memory, but in many cases, the data can be stored on the disk and passed to the memory (in chunks).

In the one-pass ONMF, the clustering possibility matrix $\mathbf{G} = [\mathbf{g_1}, \mathbf{g_2}, ..., \mathbf{g_n}]$ is computed in a sequential greedy fashion. It is expected that at the beginning, the errors would be high, but the one-pass algorithm has no easy way to correct them later on.

Algorithm 2 One-pass ONMF with the mini-batch mode. n is the total number of data points.

```
Require: $\mathbf{X}$, $\mathbf{F}^{(0)}, m, S = \lceil n/m \rceil$
  $\mathbf{V}^{(0)} = \mathbf{0}$, $\mathbf{H}^{(0)} = \mathbf{0}$
  for $t = 1 : S$ do
    Draw $\mathbf{x}^{(t)}$ (i.e., $m$ data points) from $\mathbf{X}$
    Compute $\mathbf{g}^{(t)}$ by solving Problem (3.11).
    Update $\mathbf{V}^{(t)}$ and $\mathbf{H}^{(t)}$.
    Update $\mathbf{F}^{(t)}$ by Algorithm 1
  end for
```

With the multi-pass ONMF, $\mathbf{G}$ can be updated using the $\mathbf{F}$ in the previous pass. In addition, the first- and second-order information $\mathbf{V}$ and $\mathbf{H}$ (which are small enough to be stored) in the previous pass can be utilized and updated. Therefore, it is expected that, if multiple passes are feasible, we can achieve (often) considerably more accurate results than one-pass ONMF.

3.6 Computational Savings of ONMF It is clear that, after completing one pass over the entire data matrix, ONMF will incur a computational cost of roughly $O(ndr)$, which is on the same order as the cost of executing one iteration of the Lee and Seung's multiplicative update algorithm (if the data can fit in memory). As verified by our experiments, after one pass over the data, the Frobenius loss of ONMF is often very close to the Frobenius loss of the Lee and Seung's algorithm after T iterations. This would be a considerable computational saving if T is large, even if the data can fit in memory.

When the data size is large, the IO cost can be a significant (sometimes dominating) portion of the total cost. Our one-pass ONMF only loads the data matrix once and hence incurs low IO cost. Our experiment results will verify that we often do not need many passes to achieve very accurate results.

4 Experiments on Small Data Sets (WebKB)

This section presents experiments on 4 small WebKB datasets [43], as described in Table 1.

Table 1: Information of the 4 WebKB data sets

Dataset	# points (n)	# dim. (d)	# clusters (r)
Cornell	827	4134	7
Texas	814	4029	7
Washington	1166	4165	7
Wisconsin	1210	4189	6

We let the dimensionality of the NMF output (r) equal the number of clusters.

4.1 Results of Lee and Seung's Algorithm

Figure 1 presents the averaged Frobenius loss

$$(4.20) \qquad \mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{F}\mathbf{g}_i\|_F^2$$

for $T = 500$ iterations and 100 different initializations. The losses are fairly insensitive to the initializations.

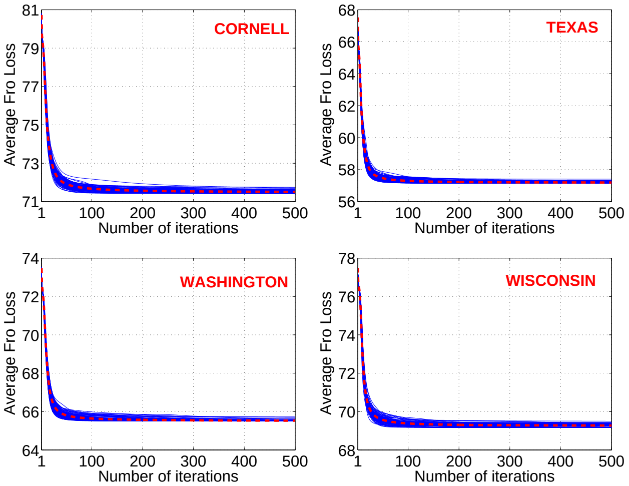

Figure 1: The averaged Frobenius losses (4.20) on the four WebKB datasets, using Lee and Seung's (L-S) algorithm. Each panel plots the results for one dataset and 100 different initializations, with the (red) dashed curve representing the average over 100 initializations.

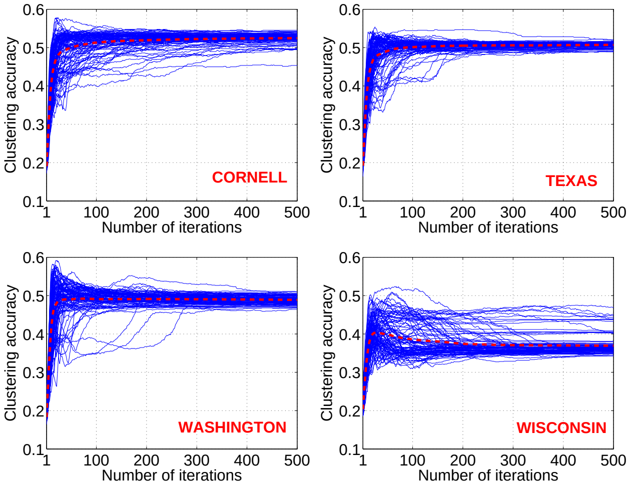

Figure 2: The clustering accuracies on the four WebKB datasets, using Lee and Seung's (L-S) algorithm. Each panel plots the results for 100 different initializations, with the (red) dashed curve representing the average.

Figure 2 presents the clustering accuracies. Here, we simply predict the cluster membership of $\mathbf{x}_i$ by

$$(4.21) \qquad \pi(\mathbf{x}_i) = \arg\max_j \quad \mathbf{g}_i(j)$$

In other words, we cluster $\mathbf{x}_i$ according to the highest possibility represented by $\mathbf{G}$. Since we obtain 100 different cluster accuracy results (using the last iterations) for each dataset (from 100 initializations), we can compute useful summary statistics (mean, minimum, maximum, and standard deviation) as in Table 2.

Table 2: **NMF** clustering accuracy

	Mean	Min	Max	Std
Cornell	0.525	0.453	0.545	0.013
Texas	0.507	0.489	0.521	0.007
Washington	0.489	0.464	0.510	0.010
Wisconsin	0.369	0.342	0.469	0.026

For comparisons, we also run the standard K-means (using the Matlab build-in implementation) algorithms on the original datasets with 100 random initializations and present the summary statistics in Table 3.

Table 3: **K-means** clustering accuracy

	Mean	Min	Max	Std
Cornell	0.345	0.282	0.414	0.030
Texas	0.369	0.322	0.410	0.022
Washington	0.358	0.289	0.458	0.049
Wisconsin	0.372	0.306	0.410	0.027

The results on clustering accuracies suggest that, as far as these four datasets are concerned, NMF does have noticeable advantages in producing better (and somewhat more stable) clustering results than the traditional K-means algorithm. Here, we should state that it is not our intention to make a general statement comparing NMF with K-means.

4.2 ONMF with Diagonal Approximation (DA)

We mainly focus on ONMF with second-order methods using diagonal approximation (DA, in this sub-section) and conjugate gradient methods (CG, in the next sub-section). We noticed that the performance of ONMF using first-order PGD is not satisfactory and hence we did not present the results (due to the space constraint).

In the experiments, we let m (i.e., size of the mini-batch) be $m = 5, 10, 50$. Figure 3 presents the results of Frobenius losses on the **Cornell** dataset for $m = 10$, using 4 different initializations and two passes.

For the second and later passes, we always compute the (averaged) Frobenius loss using (4.20) (i.e., averaged

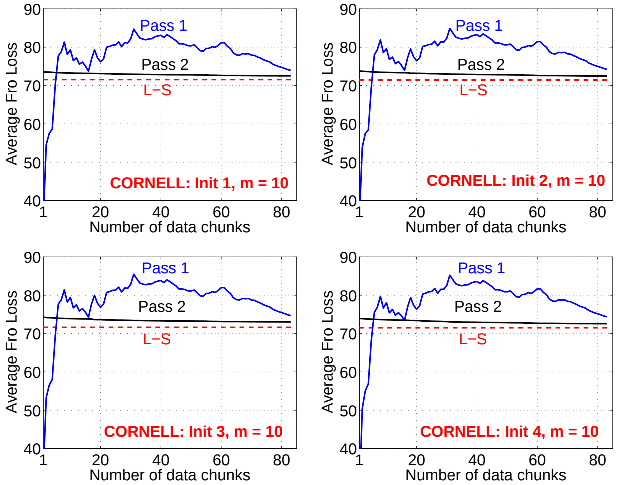

Figure 3: Averaged Frobenius loss of ONMF using diagonal approximation (DA) on the **Cornell** dataset, for four different initializations, two passes, and $m = 10$. At each time, m data points are drawn from the dataset. The x-axis represents the number of draws. The **Cornell** dataset has $n = 827$ data points. With $m = 10$, one-pass of the data needs 83 draws. In each panel, the dashed horizontal line (labeled by "L-S") is the result of the Lee and Seung's algorithm at the last (i.e., $T = 500$) iteration.

over all n data points). However, for the first pass, it is more reasonable to compute the loss using only the data points seen so far at time t, i.e.,

$$\mathcal{L}^{(t)} = \frac{1}{\min\{m \times t, n\}} \sum_{s=1}^{t} \left\| \mathbf{x}_i^{(s)} - \mathbf{F}^{(t)} \mathbf{g}_i^{(s)} \right\|_F^2 \tag{4.22}$$

The x-axis of Figure 3 (and other figures), i.e., "number of data chunks," denotes the parameter "t" in the formula above.

Figure 4 presents similar results for the **Texas**, **Washington**, and **Wisconsin** datasets. Figures 3 and 4 suggest that the performance of our ONMF with DA is not sensitive to the initializations. Figure 5 presents the losses for $m = 5$ and 50 and only one initialization.

From Figures 3, 4, and 5, we can see our ONMF with DA works well even in the first pass, because at the end of the data input, the Frobenius loss of ONMF is already very close to the Frobenius loss of Lee and Seung's algorithm after $T = 500$ iterations. The second pass further improves (and stabilizes) the results.

More experiment results with $m = 1$ on these datasets are available in Section 6 where we compare our ONMF with the related work [7].

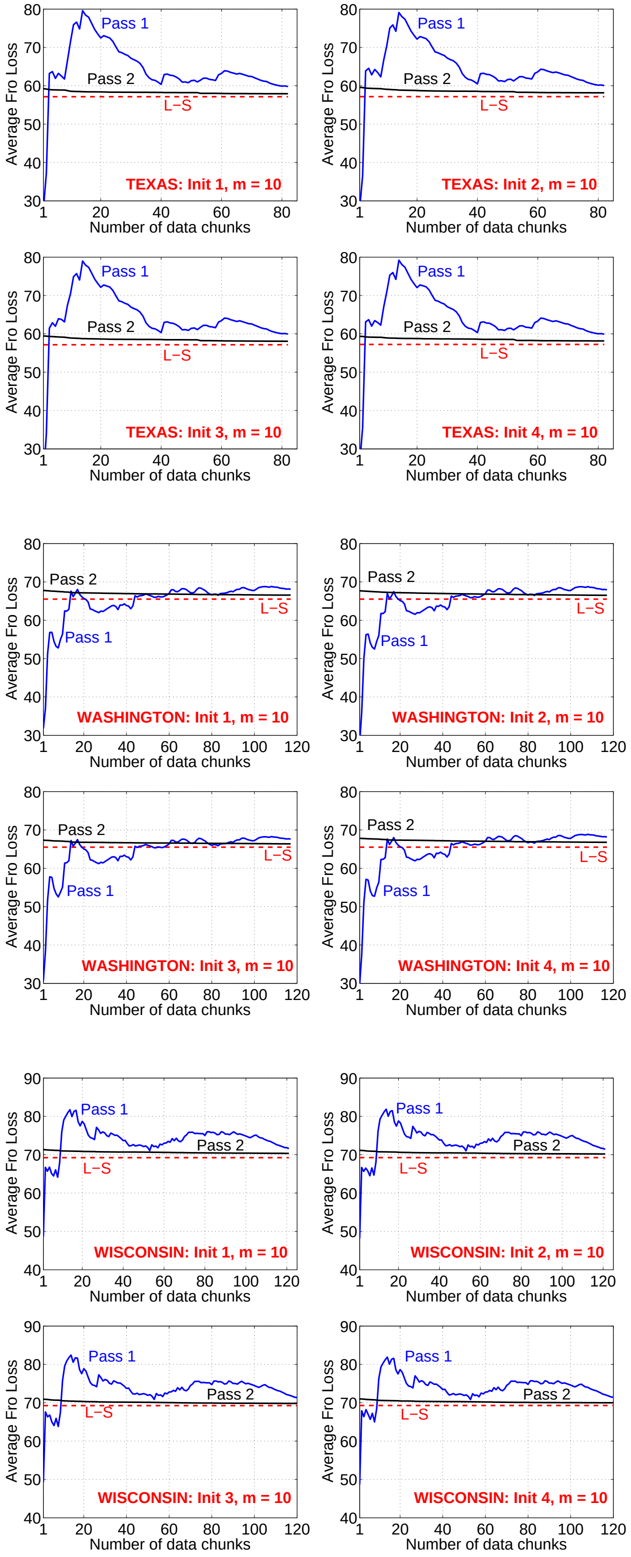

Figure 4: Averaged Frobenius loss of ONMF using diagonal approximation (DA) on the **Texas**, **Washington**, and **Wisconsin** datasets, for four different initializations, two passes, and $m = 10$.

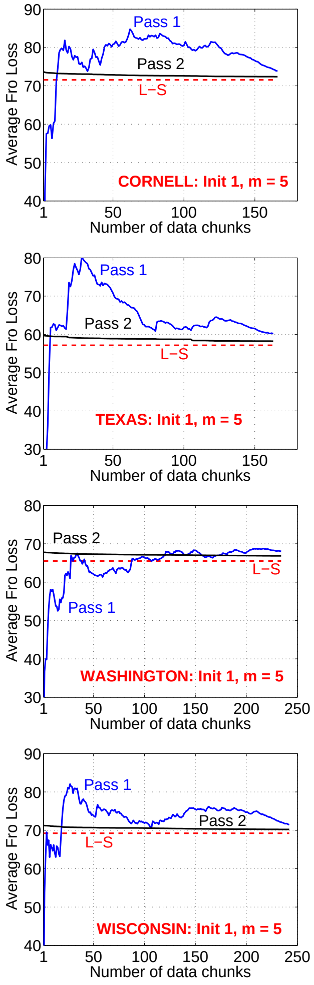

Figure 5: Averaged Frobenius loss of ONMF using diagonal approximation (DA) on the 4 WebKB datasets, for only one initialization, two passes, and $m = 5, 50$.

The left panels of Figure 6 demonstrate that the Frobenius loss continues to decrease as we increase the number of passes, with a diminishing return. The right panels of Figure 6 reveal some interesting phenomena of the clustering accuracies. It is not necessarily true that lower Frobenius losses always correspond to higher clustering accuracies. This may be partially due to the simple clustering assignment algorithm, i.e., Eq. (4.21).

4.3 ONMF with Conjugate Gradient (CG)

We also implement ONMF using conjugate gradient (CG) and present the Frobenius loss in Figure 7. In the first pass, the losses of CG are considerably higher than the losses of DA. The differences between CG and DA become much smaller in the second and later passes.

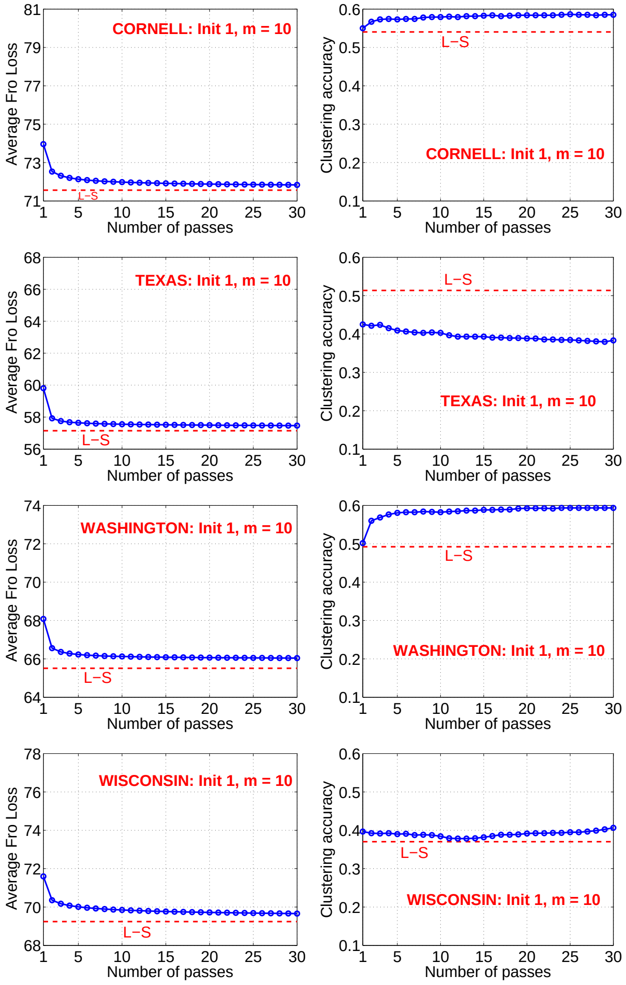

Figure 6: Left panels: averaged Frobenius loss measured at the end of each pass, for 30 passes, $m = 10$, and only one initialization. Right panels: clustering accuracy measured at the end of each pass. Note that the plots are scaled identically as Figure 1 and Figure 2. In each panel, the dashed horizontal line (labeled by "L-S") is the result of the Lee and Seung's algorithm at the last (i.e., $T = 500$) iteration.

5 Experiment on a Larger Dataset: NYTimes

The experiments based on these small WebKB datasets are valuable because they are easy to reproduce, suitable for verifying the correctness of the proposed algorithms.

This section presents more experimental results on (UCI) **NYTimes**, a much larger text data set with $n = 300000$ documents and $d = 102660$ dimensions. Because the class labels are not available, we fix $r = 5$ in our experiments.

Figure 8 presents the experimental results. The result of the (averaged) Frobenius loss using Lee and

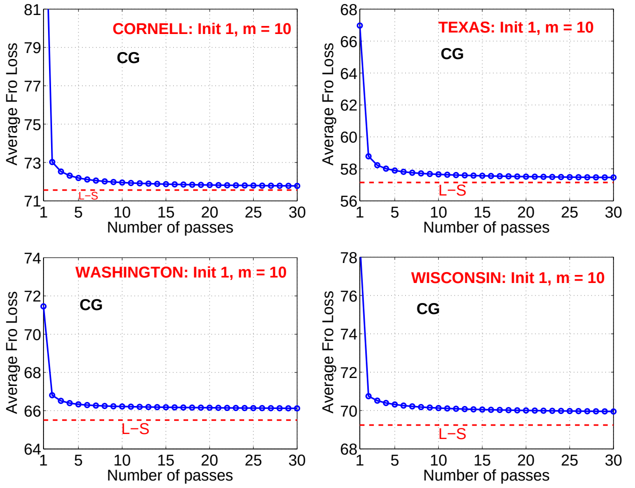

Figure 7: Averaged Frobenius loss using ONMF with conjugate gradient (CG) measured at the end of each pass, for 30 passes, $m = 10$, and one initialization. Note that the plots are scaled identically as Figure 1 and Figure 6 for easy comparisons.

Seung's algorithm is shown on the left top panel. The other five panels plot the Frobenius loss using our ONMF with $m = 100, 200, 500, 1000, 2000$, respectively. Note that even $m = 2000$ is still relatively very small compared to $n = 300000$ for this dataset.

These plots demonstrate that our ONMF works well for this large-scale dataset. One pass of the data already leads to comparable results as Lee and Seung's algorithm. The second pass further improves and stabilizes the accuracies.

Before concluding the presentation of the experiments, we should re-iterate that our ONMF is mostly advantageous when the datasets do not fit in the memory. Publicly available datasets usually do not reach such a scale. In fact, the UCI NYTimes dataset is one of the largest collections available in the public domains. On the other hand, if the dataset did not fit in the memory, then we would not be able to run Lee and Sung's method to verify our algorithms.

6 Comparison with A Prior Online Algorithm

A related[1] work in IJCAI 2007 [7] also proposed an online NMF algorithm, from an interesting perspective. Here, we first try to explain their algorithm, to the best of our understanding.[2]

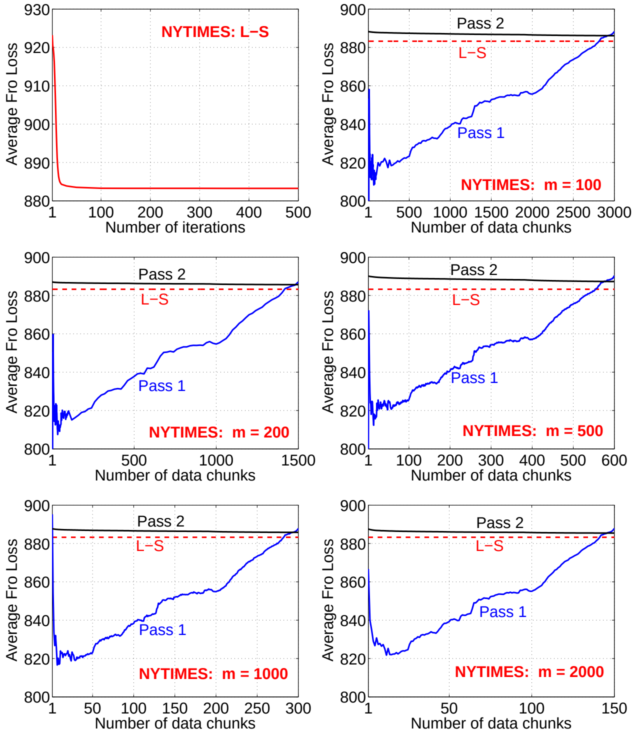

Figure 8: **NYTimes.** Left top panel: Averaged Frobenius loss of NMF using Lee and Seung's algorithm, for $T = 500$ iterations. Other panels: Averaged Frobenius losses of ONMF with diagonal approximation (DA) for two passes and $m = 100, 200, 500, 1000, 2000$.

[7] viewed the NMF solution as[3]

$$\mathbf{X} \approx \mathbf{F}\mathbf{G}^{\mathbf{T}} = (\mathbf{F}\mathbf{\Lambda})(\mathbf{G}\mathbf{\Lambda}^{-\mathbf{T}})^{\mathbf{T}} \tag{6.23}$$

where $\mathbf{\Lambda}$ is an invertible matrix.

Suppose we have already computed NMF using the old data, i.e.,

$$\mathbf{X}_{\mathbf{old}} \approx \mathbf{F}_{\mathbf{old}}\mathbf{G}_{\mathbf{old}}^{\mathrm{T}} \tag{6.24}$$

When additional data points, denoted by $\mathbf{X}_{\mathbf{new}}$, arrive, the ultimate goal is to find $\mathbf{F}$ and $\mathbf{G}$ by NMF

$$[\mathbf{X}_{\mathbf{old}}\ \mathbf{X}_{\mathbf{new}}] \approx \mathbf{F}_{\mathbf{new}}\mathbf{G}_{\mathbf{new}}^{\mathrm{T}}. \tag{6.25}$$

which, however, is not possible because $\mathbf{X}_{\mathbf{old}}$ is not stored, according to our basic assumption.

Their paper [7] suggested an interesting solution by concatenating $\mathbf{X}_{\mathbf{new}}$ with $\mathbf{F}_{\mathbf{old}}$ to form a new data matrix and then applying ordinary NMF on the new matrix. Instead of simply using $[\mathbf{F}_{\mathbf{old}}\ \mathbf{X}_{\mathbf{new}}]$ as the new

[1] The Reviewers suggested us to add this section for a direct comparison with [7]. We appreciate their constructive comments.

[2] We are grateful to Cao, Author of [7], for his explanation.

[3] Their paper [7] used (e.g.,) $\mathbf{X} = \mathbf{F}\mathbf{G}^{\mathbf{T}}$ to represent the NMF approximation. We change their "=" to "≈".

data matrix, they actually used $[\mathbf{F_{old}}\boldsymbol{\Lambda}\ \mathbf{X_{new}}]$ where $\boldsymbol{\Lambda}$ was chosen to be **diagonal**. By private communication, the Author of [7] suggested to use the l_2 norms of the columns of $\mathbf{G_{old}}$ for the diagonals of $\boldsymbol{\Lambda}$, i.e.,

$$\boldsymbol{\Lambda}_{\mathbf{jj}} = \|\,[\mathbf{G_{old}}]_{\mathbf{j}}\,\|_{\mathbf{2}}, \quad j = 1, 2, ..., r \tag{6.26}$$

Therefore, the key step is to apply NMF to obtain[4] $\mathbf{F_{temp}}$ and $\mathbf{G_{temp}}$:

(6.27)

$$[\mathbf{F_{old}}\boldsymbol{\Lambda}\ \mathbf{X_{new}}] \approx \mathbf{F_{temp}}\mathbf{G^T_{temp}} = \mathbf{F_{temp}}\left[\mathbf{W^T_1}\ \mathbf{W^T_2}\right]$$

which implies:

(6.28)

$$\mathbf{F_{old}} \approx \mathbf{F_{temp}}\mathbf{W^T_1}\boldsymbol{\Lambda}^{-1} \qquad\qquad \mathbf{X_{new}} \approx \mathbf{F_{temp}}\mathbf{W^T_2}$$

Note that

$$\mathbf{X_{old}} \approx \mathbf{F_{old}}\mathbf{G^T_{old}} \approx \mathbf{F_{temp}}\mathbf{W^T_1}\boldsymbol{\Lambda}^{-1}\mathbf{G^T_{old}} \tag{6.29}$$

Recall the ultimate goal is to compute the (approximate) NMF such that $[\mathbf{X_{old}}\ \mathbf{X_{new}}] \approx \mathbf{F_{new}}\mathbf{G^T_{new}}$. By combining

$$\mathbf{X_{old}} \approx \mathbf{F_{temp}}\left[\mathbf{W^T_1}\boldsymbol{\Lambda}^{-1}\mathbf{G^T_{old}}\right]$$
$$\mathbf{X_{new}} \approx \mathbf{F_{temp}}\left[\mathbf{W^T_2}\right]$$

we know that it is reasonable to use the following $\mathbf{F_{new}}$ and $\mathbf{G_{new}}$ as the final output:

$$\mathbf{F_{new}} = \mathbf{F_{temp}} \tag{6.30}$$

$$\mathbf{G_{new}} = \left[\begin{array}{c} \mathbf{G_{old}}\boldsymbol{\Lambda}^{-1}\mathbf{W_1} \\ \mathbf{W_2} \end{array}\right] \tag{6.31}$$

where $\mathbf{F_{temp}}$, $\mathbf{W_1}$ and $\mathbf{W_2}$ are computed from Eq. (6.27), $\boldsymbol{\Lambda}$ is the diagonal matrix defined in Eq. (6.26).

6.1 Experiments

Figure 9 compares, in terms of the averaged Frobenius loss as defined in Eq. (4.22), our ONMF (using the diagonal approximation) with the previous algorithm [7] (labeled by "IJCAI" in the figures), on the four WebKB datasets. We report the experiments for one initialization (for each dataset) and $m = 1, 5, 10$.

Figure 10 compares our ONMF with [7] on the NYTimes dataset, for $m = 100, 200, 500, 1000$.

Note that, Figures 9 and 10 only report the performance of one-pass ONMF, because it is not exactly clear to us how to implement [7] as a multi-pass algorithm.

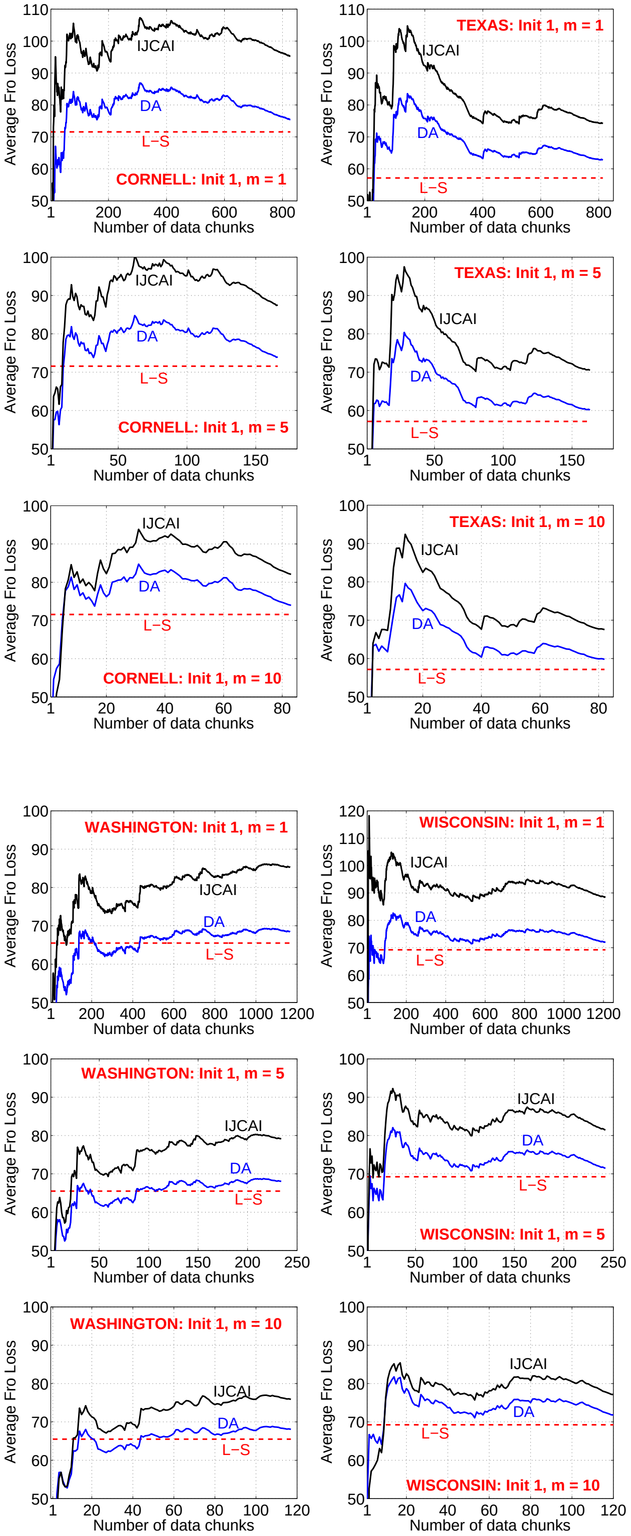

Figure 9: The averaged Frobenius losses (4.20) on the four **WebKB** datasets, for comparing our ONMF (using diagonal approximation, "DA") with the prior work in [7] (labeled by "IJCAI"), using $m = 1, 5, 10$.

[4] Note that in the paper [7], the authors also added an additional regularization term $\alpha\mathbf{\Gamma}\mathbf{F^T}\mathbf{F}$ which resulted in an additional term in the NMF multiplicative updating formula. The chosen α was quite small (and appeared to be heuristic). We also implemented this step but we found the impact of this additional term was very minimal. Therefore, to simplify the presentation, we let $\alpha = 0$ in the formulation and presented the experimental results with $\alpha = 0$.

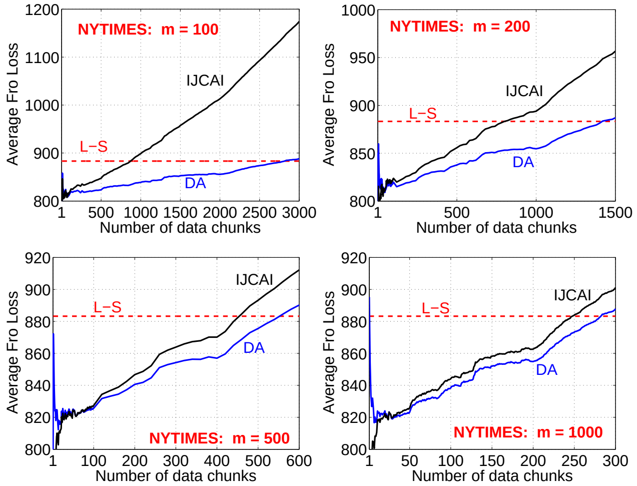

Figure 10: The averaged Frobenius losses (4.20) on the **NYTimes** dataset, for comparing our ONMF (using diagonal approximation, "DA") with the prior work in [7] (labeled by "IJCAI"), using $m = 100, 200, 500, 1000$.

7 Conclusions

The method of Nonnegative Matrix Factorization (NMF) has been widely used in many data mining and information retrieval applications. In particular, NMF has been applied in various clustering tasks. Conventional NMF algorithms require the data matrix to reside in the memory during the solution process, which is problematic when the datasets are very large (e.g., they may not even fit in the memory). Also, for modern data stream applications, if the data are transient (e.g., not even stored on disk) or the applications require real-time updates, conventional NMF methods would not be able to meet the requirement.

In this paper, we proposed an efficient online NMF algorithm, which processes the incoming data incrementally, one data point (or one chunk of data points) at a time. Our method scales gracefully to very large datasets. Our experiments demonstrate that even only with one pass of the data, our ONMF algorithm can achieve nearly the same performance as the conventional NMF method. The accuracies continue to improve if multiple passes of the data are allowed.

Finally, as suggested by the Reviewers, we implemented a previously proposed online NMF algorithm and compared its performance with ours.

8 Acknowledgement

Chenhao Tan and Ping Li are supported by NSF (DMS-0808864), ONR (YIP-N000140910911), and a grant from Microsoft. Fei Wang was supported by ONR (YIP-N000140910911) and the grant from Microsoft.

References

[1] N. Andrei. Accelerated conjugate gradient algorithm with finite difference hessian/vector product approximation for unconstrained optimization. *Journal of Computational and Applied Mathematics*, 230(2):570–582, 2009.

[2] D. P. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20:221–246, 1982.

[3] D. P. Bertsekas. *Nonlinear Programming.* Belmont, MA, 1999.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[5] L. Bottou. Stochastic learning. In O. Bousquet and U. von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[7] B. Cao, D. Shen, J. Sun, X. Wang, Q. Yang, and Z. Chen. Detect and track latent factors with online nonnegative matrix factorization. In *Proc. International Joint Conference on Artificial Intelligence*, pages 2689–2694, 2007.

[8] C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of web clustering engines. *ACM Comput. Surv.*, 41(3):1–38, 2009.

[9] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems (NIPS)*, pages 281–288, 2007.

[10] D. R. Cutting, D. R. Karger, J. O. Pederson, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 318–329, 1992.

[11] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 271–280, 2007.

[12] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.

[13] C. Ding, X. He, and H. D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 5th SIAM Int'l Conf. Data Mining (SDM)*, pages 606–610, 2005.

[14] C. Ding, T. Li, and M. I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[15] D. Fetterly, M. Manasse, and M. Najork. On the evolution of clusters of near-duplicate web pages. In

Proc. of the 12th International Conference on World Wide Web (WWW), pages 37–45, 2003.

[16] B. C. M. Fung, K. Wang, C. M. Benjamin, F. Ke, and M. Ester. Hierarchical document clustering using frequent itemsets. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM)*, 2003.

[17] E. Gaussier and C. Goutte. Relation between plsa and nmf and implications. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 601–602, 2005.

[18] T. Hu., H. Xiong, W. Zhou, S. Y. Sung, and H. Luo. Hypergraph partitioning for document clustering: A unified clique perspective. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 871–872, 2008.

[19] D. Kim, S. Sra, and I. Dhillon. Fast newton-type methods for the least squares nonnegative matrix approximation problem. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 343–354, 2007.

[20] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Proceedings of the 8th International Conference on Data Mining (ICDM)*, pages 353–362, 2008.

[21] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications.* New York: Springer-Verlag, 1997.

[22] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Society for Industrial Mathematics, 1995.

[23] D. D. Lee and H. S. Seung. Learning the parts of objects with nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

[24] D. D. Lee and H. S. Seung. Algorithms for nonnegative matrix factorization. In *Advances in Neural Information Processing System (NIPS)*, pages 556–562, 2000.

[25] P. Li. Very sparse stable random projections for dimension reduction in l_α ($0< \alpha \leqslant 2$) norm. In *Proocddings of the ACM SIGKDD Conference (SIGKDD)*, pages 440–449, 2007.

[26] P. Li. Computationally efficient estimators for dimension reductions using stable random projections. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, pages 403–412, 2008.

[27] P. Li, K. W. Church, and T. Hastie. One sketch for all: Theory and application of conditional random sampling. In *Advances in Neural Information Processing System (NIPS)*, pages 953–960, 2008.

[28] P. Li, A. C. König, and W. Gui. b-bit minwise hashing for estimating three-way similarities. In *Advances in Neural Information Processing System (NIPS)*, 2010.

[29] C. J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779.

[30] C. Liu, H. Yang, J. Fan, L. He, and Y. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *Proceedings of the 19th ACM International World Wide Web Conference (WWW)*, pages 681–690, 2010.

[31] X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 186–193, 2004.

[32] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of The 26th International Conference on Machine Learning (ICML)*, pages 689–696, 2009.

[33] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing and Management*, 42(2):373–386, 2006.

[34] J. Silva, J. Mexia, A. Coelho, and G. Lopes. Document clustering and cluster topic extraction in multilingual corpora. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM)*, pages 513–520, 2001.

[35] X. Wan and J. Yang. Multi-document summarization using cluster-based link analysis. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 299–306, 2007.

[36] F. Wang and P. Li. Compressed nonnegative sparse coding. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, 2010.

[37] F. Wang and P. Li. Efficient non-negative matrix factorization with random projections. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM)*, pages 281–292, 2010.

[38] F. Wang, P. Li, and A. C. König. Learning a bistochastic data similarity matrix. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, 2010.

[39] S. Xu and J. Zhang. A hybrid parallel web document clustering algorithm performance study. *Journal of Supercomputing*, 30(2):117–131, 2004.

[40] W. Xu and Y. Gong. Document clustering by concept factorization. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR)*, pages 202–209, 2004.

[41] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR)*, pages 267–273, 2003.

[42] S. Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5-6):790–798, 2005.

[43] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 487–494, 2007.