Factoring nonnegative matrices with linear programs

Victor Bittorf
bittorf@cs.wisc.edu

Benjamin Recht
brecht@cs.wisc.edu

Christopher Ré
chrisre@cs.wisc.edu

Computer Sciences
University of Wisconsin

Joel A. Tropp
Computing and Mathematical Sciences
California Institute of Technology
tropp@cms.caltech.edu

Abstract

This paper describes a new approach, based on linear programming, for computing nonnegative matrix factorizations (NMFs). The key idea is a data-driven model for the factorization where the most salient features in the data are used to express the remaining features. More precisely, given a data matrix $\boldsymbol{X}$, the algorithm identifies a matrix $\boldsymbol{C}$ that satisfies $\boldsymbol{X} \approx \boldsymbol{C}\boldsymbol{X}$ and some linear constraints. The constraints are chosen to ensure that the matrix $\boldsymbol{C}$ selects features; these features can then be used to find a low-rank NMF of $\boldsymbol{X}$. A theoretical analysis demonstrates that this approach has guarantees similar to those of the recent NMF algorithm of Arora et al. (2012). In contrast with this earlier work, the proposed method extends to more general noise models and leads to efficient, scalable algorithms. Experiments with synthetic and real datasets provide evidence that the new approach is also superior in practice. An optimized C++ implementation can factor a multigigabyte matrix in a matter of minutes.

1 Introduction

Nonnegative matrix factorization (NMF) is a popular approach for selecting features in data [16–18, 23]. Many machine-learning and data-mining software packages (including Matlab [3], R [12], and Oracle Data Mining [1]) now include heuristic computational methods for NMF. Nevertheless, we still have limited theoretical understanding of when these heuristics are correct.

The difficulty in developing rigorous methods for NMF stems from the fact that the problem is computationally challenging. Indeed, Vavasis has shown that NMF is NP-Hard [27]; see [4] for further worst-case hardness results. As a consequence, we must instate additional assumptions on the data if we hope to compute nonnegative matrix factorizations in practice.

In this spirit, Arora, Ge, Kannan, and Moitra (AGKM) have exhibited a polynomial-time algorithm for NMF that is provably correct—provided that the data is drawn from an appropriate model, based on ideas from [8]. The AGKM result describes one circumstance where we can be sure that NMF algorithms are capable of producing meaningful answers. This work has the potential to make an impact in machine learning because proper feature selection is an important preprocessing step for many other techniques. Even so, the actual impact is damped by the fact that the AGKM algorithm is too computationally expensive for large-scale problems and is not tolerant to departures from the modeling assumptions. Thus, for NMF, there remains a gap between the theoretical exercise and the actual practice of machine learning.

The present work presents a scalable, robust algorithm that can successfully solve the NMF problem under appropriate hypotheses. Our first contribution is a new formulation of the nonnegative feature selection problem that only requires the solution of a single linear program. Second, we provide a theoretical analysis of this algorithm. This argument shows that our method succeeds under the same modeling assumptions as the AGKM algorithm with an additional *margin constraint* that is common in machine learning. We prove that if there exists a unique, well-defined model, then we can recover this model accurately; our error bound improves substantially on the error bound for the AGKM algorithm in the high SNR regime. One may argue that NMF only "makes sense" (i.e., is well posed) when a unique solution exists, and so we believe our result has independent interest. Furthermore, our algorithm can be adapted for a wide class of noise models.

In addition to these theoretical contributions, our work also includes a major algorithmic and experimental component. Our formulation of NMF allows us to exploit methods from operations research and database systems to design solvers that scale to extremely large datasets. We develop an efficient stochastic gradient descent (SGD) algorithm that is (at least) two orders of magnitude faster than the approach of AGKM when both are implemented in Matlab. We describe a parallel implementation of our SGD algorithm that can robustly factor matrices with 10^5 features and 10^6 examples in a few minutes on a multicore workstation.

Our formulation of NMF uses a data-driven modeling approach to simplify the factorization problem. More precisely, we search for a small collection of rows from the data matrix that can be used to express the other rows. This type of approach appears in a number of other factorization problems, including rank-revealing QR [15], interpolative decomposition [20], subspace clustering [10, 24], dictionary learning [11], and others. Our computational techniques can be adapted to address large-scale instances of these problems as well.

2 Separable Nonnegative Matrix Factorizations and Hott Topics

Notation. For a matrix $\boldsymbol{M}$ and indices i and j, we write $\boldsymbol{M}_{i\cdot}$ for the ith row of $\boldsymbol{M}$ and $\boldsymbol{M}_{\cdot j}$ for the jth column of $\boldsymbol{M}$. We write M_{ij} for the (i, j) entry.

Let $\boldsymbol{Y}$ be a nonnegative $f \times n$ data matrix with columns indexing examples and rows indexing features. Exact NMF seeks a factorization $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{W}$ where the feature matrix $\boldsymbol{F}$ is $f \times r$, where the weight matrix $\boldsymbol{W}$ is $r \times n$, and both factors are nonnegative. Typically, $r \ll \min\{f, n\}$.

Unless stated otherwise, we assume that each row of the data matrix $\boldsymbol{Y}$ is normalized so it sums to one. Under this hypothesis, we may also assume that each row of $\boldsymbol{F}$ and of $\boldsymbol{W}$ also sums to one [4].

It is notoriously difficult to solve the NMF problem. Vavasis showed that it is NP-complete to decide whether a matrix admits a rank-r nonnegative factorization [27]. AGKM proved that an exact NMF algorithm can be used to solve 3-SAT in subexponential time [4].

The literature contains some mathematical analysis of NMF that can be used to motivate algorithmic development. Thomas [25] developed a necessary and sufficient condition for the existence of a rank-r NMF. More recently, Donoho and Stodden [8] obtained a related sufficient condition for uniqueness. AGKM exhibited an algorithm that can produce a nonnegative matrix factorization under a weaker sufficient condition. To state their results, we need a definition.

Definition 2.1 *A set of vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_r\} \subset \mathbb{R}^d$ is* simplicial *if no vector $\boldsymbol{v}_i$ lies in the convex hull of $\{\boldsymbol{v}_j : j \neq i\}$. The set of vectors is* α-robust simplicial *if, for each i, the ℓ_1 distance from $\boldsymbol{v}_i$ to the convex hull of $\{\boldsymbol{v}_j : j \neq i\}$ is at least α. Figure 1 illustrates these concepts.*

These ideas support the uniqueness results of Donoho and Stodden and the AGKM algorithm. Indeed, we can find an NMF of $\boldsymbol{Y}$ efficiently if $\boldsymbol{Y}$ contains a set of r rows that is simplicial and whose convex hull contains the remaining rows.

Definition 2.2 *An NMF $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{W}$ is called* separable *if the rows of $\boldsymbol{W}$ are simplicial and there is a permutation matrix $\boldsymbol{\Pi}$ such that*

$$\boldsymbol{\Pi}\boldsymbol{F} = \begin{bmatrix} \mathbf{I}_r \\ \boldsymbol{M} \end{bmatrix}. \tag{1}$$

Algorithm 1: AGKM: Approximably Separable Nonnegative Matrix Factorization [4]

1: Initialize $R = \emptyset$.
2: Compute the $f \times f$ matrix $\boldsymbol{D}$ with $D_{ij} = \|\boldsymbol{X}_{i\cdot} - \boldsymbol{X}_{j\cdot}\|_1$.
3: **for** $k = 1, \ldots f$ **do**
4: Find the set $\mathcal{N}_k$ of rows that are at least $5\epsilon/\alpha + 2\epsilon$ away from $\boldsymbol{X}_{k\cdot}$.
5: Compute the distance δ_k of $\boldsymbol{X}_{k\cdot}$ from $\text{conv}(\{\boldsymbol{X}_{j\cdot} : j \in \mathcal{N}_k\})$.
6: **if** $\delta_k > 2\epsilon$, add k to the set R.
7: **end for**
8: Cluster the rows in R as follows: j and k are in the same cluster if $D_{jk} \leq 10\epsilon/\alpha + 6\epsilon$.
9: Choose one element from each cluster to yield $\boldsymbol{W}$.
10: $\boldsymbol{F} = \arg\min_{\boldsymbol{Z} \in \mathbb{R}^{f \times r}} \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{W}\|_{\infty,1}$

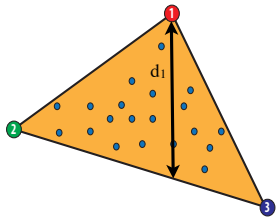

Figure 1: Numbered circles are hott topics. Their convex hull (orange) contains the other topics (small circles), so the data admits a separable NMF. The arrow d_1 marks the ℓ_1 distance from hott topic (1) to the convex hull of the other two hott topics; definitions of d_2 and d_3 are similar. The hott topics are α-robustly simplicial when each $d_i \geq \alpha$.

To compute a separable factorization of $\boldsymbol{Y}$, we must first identify a simplicial set of rows from $\boldsymbol{Y}$. Afterward, we compute weights that express the remaining rows as convex combinations of this distinguished set. We call the simplicial rows *hott* and the corresponding features *hott topics*.

This model allows us to express all the features for a particular instance if we know the values of the instance at the simplicial rows. This assumption can be justified in a variety of applications. For example, in text, knowledge of a few keywords may be sufficient to reconstruct counts of the other words in a document. In vision, localized features can be used to predict gestures. In audio data, a few bins of the spectrogram may allow us to reconstruct the remaining bins.

While a nonnegative matrix one encounters in practice might not admit a separable factorization, it may be *well-approximated* by a nonnnegative matrix with separable factorization. AGKM derived an algorithm for nonnegative matrix factorization of a matrix that is well-approximated by a separable factorization. To state their result, we introduce a norm on $f \times n$ matrices:

$$\|\boldsymbol{\Delta}\|_{\infty,1} := \max_{1 \leq i \leq f} \sum_{j=1}^{n} |\Delta_{ij}| .$$

Theorem 2.3 (AGKM [4]) *Let ϵ and α be nonnegative constants satisfying $\epsilon \leq \frac{\alpha^2}{20+13\alpha}$. Let $\boldsymbol{X}$ be a nonnegative data matrix. Assume $\boldsymbol{X} = \boldsymbol{Y} + \boldsymbol{\Delta}$ where $\boldsymbol{Y}$ is a nonnegative matrix whose rows have unit ℓ_1 norm, where $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{W}$ is a rank-r separable factorization in which the rows of $\boldsymbol{W}$ are α-robust simplicial, and where $\|\boldsymbol{\Delta}\|_{\infty,1} \leq \epsilon$. Then Algorithm 1 finds a rank-r nonnegative factorization $\hat{\boldsymbol{F}}\hat{\boldsymbol{W}}$ that satisfies the error bound $\left\|\boldsymbol{X} - \hat{\boldsymbol{F}}\hat{\boldsymbol{W}}\right\|_{\infty,1} \leq 10\epsilon/\alpha + 7\epsilon$.*

In particular, the AGKM algorithm computes the factorization exactly when $\epsilon = 0$. Although this method is guaranteed to run in polynomial time, it has many undesirable features. First, the algorithm requires a priori knowledge of the parameters α and ϵ. It may be possible to calculate ϵ, but we can only estimate α if we know which rows are hott. Second, the algorithm computes all ℓ_1 distances between rows at a cost of $O(f^2 n)$. Third, for every row in the matrix, we must determine its distance to the convex hull of the rows that lie at a sufficient distance; this step requires us to solve a linear program for each row of the matrix at a cost of $\Omega(fn)$. Finally, this method is intimately linked to the choice of the error norm $\|\cdot\|_{\infty,1}$. It is not obvious how to adapt the algorithm for other noise models. We present a new approach, based on linear programming, that overcomes these drawbacks.

3 Main Theoretical Results: NMF by Linear Programming

This paper shows that we can factor an approximately separable nonnegative matrix by solving a linear program. A major advantage of this formulation is that it scales to very large data sets.

Algorithm 2 Separable Nonnegative Matrix Factorization by Linear Programming

Require: An $f \times n$ nonnegative matrix $\boldsymbol{Y}$ with a rank-r separable NMF.
Ensure: An $f \times r$ matrix $\boldsymbol{F}$ and $r \times n$ matrix $\boldsymbol{W}$ with $\boldsymbol{F} \geq \boldsymbol{0}$, $\boldsymbol{W} \geq \boldsymbol{0}$, and $\boldsymbol{Y} = \boldsymbol{FW}$.
1: Find the unique $\boldsymbol{C} \in \Phi(\boldsymbol{Y})$ to minimize $\boldsymbol{p}^T \operatorname{diag}(\boldsymbol{C})$ where $\boldsymbol{p}$ is any vector with distinct values.
2: Let $I = \{i \,:\, C_{ii} = 1\}$ and set $\boldsymbol{W} = \boldsymbol{Y}_{I\cdot}$ and $\boldsymbol{F} = \boldsymbol{C}_{\cdot I}$.

Here is the key observation: Suppose that $\boldsymbol{Y}$ is any $f \times n$ nonnegative matrix that admits a rank-r separable factorization $\boldsymbol{Y} = \boldsymbol{FW}$. If we pad $\boldsymbol{F}$ with zeros to form an $f \times f$ matrix, we have

$$\boldsymbol{Y} = \boldsymbol{\Pi}^T \begin{bmatrix} \mathbf{I}_r & \boldsymbol{0} \\ \boldsymbol{M} & \boldsymbol{0} \end{bmatrix} \boldsymbol{\Pi Y} =: \boldsymbol{CY}$$

We call the matrix $\boldsymbol{C}$ *factorization localizing*. Note that any factorization localizing matrix $\boldsymbol{C}$ is an element of the polyhedral set

$$\Phi(\boldsymbol{Y}) := \{\boldsymbol{C} \geq \boldsymbol{0} \,:\, \boldsymbol{CY} = \boldsymbol{Y},\ \operatorname{Tr}(\boldsymbol{C}) = r,\ C_{jj} \leq 1\ \forall j,\ C_{ij} \leq C_{jj}\ \forall i, j\}.$$

Thus, to find an exact NMF of $\boldsymbol{Y}$, it suffices to find a feasible element of $\boldsymbol{C} \in \Phi(\boldsymbol{Y})$ whose diagonal is integral. This task can be accomplished by linear programming. Once we have such a $\boldsymbol{C}$, we construct $\boldsymbol{W}$ by extracting the rows of $\boldsymbol{X}$ that correspond to the indices i where $C_{ii} = 1$. We construct the feature matrix $\boldsymbol{F}$ by extracting the nonzero columns of $\boldsymbol{C}$. This approach is summarized in Algorithm 2. In turn, we can prove the following result.

Theorem 3.1 *Suppose $\boldsymbol{Y}$ is a nonnegative matrix with a rank-r separable factorization $\boldsymbol{Y} = \boldsymbol{FW}$. Then Algorithm 2 constructs a rank-r nonnegative matrix factorization of $\boldsymbol{Y}$.*

As the theorem suggests, we can isolate the rows of $\boldsymbol{Y}$ that yield a simplicial factorization by solving a single linear program. The factor $\boldsymbol{F}$ can be found by extracting columns of $\boldsymbol{C}$.

3.1 Robustness to Noise

Suppose we observe a nonnegative matrix $\boldsymbol{X}$ whose rows sum to one. Assume that $\boldsymbol{X} = \boldsymbol{Y} + \boldsymbol{\Delta}$ where $\boldsymbol{Y}$ is a nonnegative matrix whose rows sum to one, which has a rank-r separable factorization $\boldsymbol{Y} = \boldsymbol{FW}$ such that the rows of $\boldsymbol{W}$ are α-robust simplicial, and where $\|\boldsymbol{\Delta}\|_{\infty,1} \leq \epsilon$. Define the polyhedral set

$$\Phi_\tau(\boldsymbol{X}) := \left\{\boldsymbol{C} \geq \boldsymbol{0} \,:\, \|\boldsymbol{CX} - \boldsymbol{X}\|_{\infty,1} \leq \tau, \operatorname{Tr}(\boldsymbol{C}) = r, C_{jj} \leq 1\ \forall j, C_{ij} \leq C_{jj}\ \forall i, j\right\}$$

The set $\Phi(\boldsymbol{X})$ consists of matrices $\boldsymbol{C}$ that *approximately* locate a factorization of $\boldsymbol{X}$. We can prove the following result.

Theorem 3.2 *Suppose that $\boldsymbol{X}$ satisfies the assumptions stated in the previous paragraph. Furthermore, assume that for every row $\boldsymbol{Y}_{j,\cdot}$ that is not hott, we have the margin constraint $\|\boldsymbol{Y}_{j,\cdot} - \boldsymbol{Y}_{i,\cdot}\| \geq d_0$ for all hott rows i. Then we can find a nonnegative factorization satisfying $\left\|\boldsymbol{X} - \hat{\boldsymbol{F}}\hat{\boldsymbol{W}}\right\|_{\infty,1} \leq 2\epsilon$ provided that $\epsilon < \frac{\min\{\alpha d_0, \alpha^2\}}{9(r+1)}$. Furthermore, this factorization correctly identifies the hott topics appearing in the separable factorization of $\boldsymbol{Y}$.*

Algorithm 3 requires the solution of two linear programs. The first minimizes a cost vector over $\Phi_{2\epsilon}(\boldsymbol{X})$. This lets us find $\hat{\boldsymbol{W}}$. Afterward, the matrix $\hat{\boldsymbol{F}}$ can be found by setting

$$\hat{\boldsymbol{F}} = \arg\min_{\boldsymbol{Z} \geq \boldsymbol{0}} \left\|\boldsymbol{X} - \boldsymbol{Z}\hat{\boldsymbol{W}}\right\|_{\infty,1}. \tag{2}$$

Our robustness result requires a *margin-type* constraint assuming that the original configuration consists either of duplicate hott topics, or topics that are reasonably far away from the hott topics. On the other hand, under such a margin constraint, we can construct a considerably better approximation that guaranteed by the AGKM algorithm. Moreover, unlike AGKM, our algorithm does not need to know the parameter α.

Algorithm 3 Approximably Separable Nonnegative Matrix Factorization by Linear Programming

Require: An $f \times n$ nonnegative matrix $\boldsymbol{X}$ that satisfies the hypotheses of Theorem 3.2.
Ensure: An $f \times r$ matrix $\boldsymbol{F}$ and $r \times n$ matrix $\boldsymbol{W}$ with $\boldsymbol{F} \geq \boldsymbol{0}$, $\boldsymbol{W} \geq \boldsymbol{0}$, and $\|\boldsymbol{X} - \boldsymbol{F}\boldsymbol{W}\|_{\infty,1} \leq 2\epsilon$.

1: Find $\boldsymbol{C} \in \Phi_{2\epsilon}(\boldsymbol{X})$ that minimizes $\boldsymbol{p}^T \operatorname{diag} \boldsymbol{C}$ where $\boldsymbol{p}$ is any vector with distinct values.
2: Let $I = \{i \,:\, C_{ii} = 1\}$ and set $\boldsymbol{W} = \boldsymbol{X}_{I\cdot}$.
3: Set $\boldsymbol{F} = \arg\min_{\boldsymbol{Z} \in \mathbb{R}^{f \times r}} \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{W}\|_{\infty,1}$

The proofs of Theorems 3.1 and 3.2 can be found in the b version of this paper [6]. The main idea is to show that we can only represent a hott topic efficiently using the hott topic itself. Some earlier versions of this paper contained incomplete arguments, which we have remedied. For a signifcantly stronger robustness analysis of Algorithm 3, see the recent paper [13].

Having established these theoretical guarantees, it now remains to develop an algorithm to solve the LP. Off-the-shelf LP solvers may suffice for moderate-size problems, but for large-scale matrix factorization problems, their running time is prohibitive, as we show in Section 5. In Section 4, we turn to describe how to solve Algorithm 3 efficiently for large data sets.

3.2 Related Work

Localizing factorizations via column or row subset selection is a popular alternative to direct factorization methods such as the SVD. Interpolative decomposition such as Rank-Revealing QR [15] and CUR [20] have favorable efficiency properties as compared to factorizations (such as SVD) that are not based on exemplars. Factorization localization has been used in subspace clustering and has been shown to be robust to outliers [10, 24].

In recent work on dictionary learning, Esser et al. and Elhamifar et al. have proposed a factorization localization solution to nonnegative matrix factorization using group sparsity techniques [9, 11]. Esser et al. prove asymptotic exact recovery in a restricted noise model, but this result requires preprocessing to remove duplicate or near-duplicate rows. Elhamifar shows exact representative recovery in the noiseless setting assuming no hott topics are duplicated. Our work here improves upon this work in several aspects, enabling finite sample error bounds, the elimination of any need to preprocess the data, and algorithmic implementations that scale to very large data sets.

4 Incremental Gradient Algorithms for NMF

The rudiments of our fast implementation rely on two standard optimization techniques: dual decomposition and incremental gradient descent. Both techniques are described in depth in Chapters 3.4 and 7.8 of Bertsekas and Tstisklis [5].

We aim to minimize $\boldsymbol{p}^T \operatorname{diag}(\boldsymbol{C})$ subject to $\boldsymbol{C} \in \Phi_\tau(\boldsymbol{X})$. To proceed, form the Lagrangian

$$\mathcal{L}(\boldsymbol{C}, \beta, \boldsymbol{w}) = \boldsymbol{p}^T \operatorname{diag}(\boldsymbol{C}) + \beta(\operatorname{Tr}(\boldsymbol{C}) - r) + \sum_{i=1}^{f} w_i \left(\|\boldsymbol{X}_{i\cdot} - [\boldsymbol{C}\boldsymbol{X}]_{i\cdot}\|_1 - \tau\right)$$

with multipliers β and $\boldsymbol{w} \geq \boldsymbol{0}$. Note that we do not dualize out all of the constraints. The remaining ones appear in the constraint set $\Phi_0 = \{\boldsymbol{C} \,:\, \boldsymbol{C} \geq \boldsymbol{0},\ \operatorname{diag}(\boldsymbol{C}) \leq 1,\ \text{and } C_{ij} \leq C_{jj} \text{ for all } i, j\}$.

Dual subgradient ascent solves this problem by alternating between minimizing the Lagrangian over the constraint set Φ_0, and then taking a subgradient step with respect to the dual variables

$$w_i \leftarrow w_i + s\left(\|\boldsymbol{X}_{i\cdot} - [\boldsymbol{C}^\star\boldsymbol{X}]_{i\cdot}\|_1 - \tau\right) \quad \text{and} \quad \beta \leftarrow \beta + s(\operatorname{Tr}(\boldsymbol{C}^\star) - r)$$

where $\boldsymbol{C}^\star$ is the minimizer of the Lagrangian over Φ_0. The update of w_i makes very little difference in the solution quality, so we typically only update β.

We minimize the Lagrangian using projected incremental gradient descent. Note that we can rewrite the Lagrangian as

$$\mathcal{L}(\boldsymbol{C}, \beta, \boldsymbol{w}) = -\tau \boldsymbol{1}^T \boldsymbol{w} - \beta r + \sum_{k=1}^{n} \left(\sum_{j \in \operatorname{supp}(\boldsymbol{X}_{\cdot k})} w_j \|\boldsymbol{X}_{jk} - [\boldsymbol{C}\boldsymbol{X}]_{jk}\|_1 + \mu_j(p_j + \beta)C_{jj} \right).$$

Algorithm 4 HOTTOPIXX: Approximate Separable NMF by Incremental Gradient Descent

Require: An $f \times n$ nonnegative matrix $\boldsymbol{X}$. Primal and dual stepsizes s_p and s_d.
Ensure: An $f \times r$ matrix $\boldsymbol{F}$ and $r \times n$ matrix $\boldsymbol{W}$ with $\boldsymbol{F} \geq \boldsymbol{0}$, $\boldsymbol{W} \geq \boldsymbol{0}$, and $\|\boldsymbol{X} - \boldsymbol{FW}\|_{\infty,1} \leq 2\epsilon$.

1: Pick a cost $\boldsymbol{p}$ with distinct entries.
2: Initialize $\boldsymbol{C} = \boldsymbol{0}$, $\beta = 0$
3: **for** $t = 1, \ldots, N_{epochs}$ **do**
4: **for** $i = 1, \ldots n$ **do**
5: Choose k uniformly at random from $[n]$.
6: $\boldsymbol{C} \leftarrow \boldsymbol{C} + s_p \cdot \operatorname{sign}(\boldsymbol{X}_{\cdot k} - \boldsymbol{C}\boldsymbol{X}_{\cdot k})\boldsymbol{X}_{\cdot k}^T - s_p \operatorname{diag}(\boldsymbol{\mu} \circ (\beta \boldsymbol{1} - \boldsymbol{p}))$.
7: **end for**
8: Project $\boldsymbol{C}$ onto Φ_0.
9: $\beta \leftarrow \beta + s_d(\operatorname{Tr}(\boldsymbol{C}) - r)$
10: **end for**
11: Let $I = \{i \ : \ C_{ii} = 1\}$ and set $\boldsymbol{W} = \boldsymbol{X}_{I\cdot}$.
12: Set $\boldsymbol{F} = \arg\min_{\boldsymbol{Z} \in \mathbb{R}^{f \times r}} \|\boldsymbol{X} - \boldsymbol{ZW}\|_{\infty,1}$

Here, $\operatorname{supp}(\boldsymbol{x})$ is the set indexing the entries where $\boldsymbol{x}$ is nonzero, and μ_j is the number of nonzeros in row j divided by n. The incremental gradient method chooses one of the n summands at random and follows its subgradient. We then project the iterate onto the constraint set Φ_0. The projection onto Φ_0 can be performed in the time required to sort the individual columns of $\boldsymbol{C}$ plus a linear-time operation. The full procedure is described in the extended version of this paper [6]. In the case where we expect a unique solution, we can drop the constraint $C_{ij} \leq C_{jj}$, resulting in a simple clipping procedure: set all negative items to zero and set any diagonal entry exceeding one to one. In practice, we perform a tradeoff. Since the constraint $C_{ij} \leq C_{jj}$ is used solely for symmetry breaking, we have found empirically that we only need to project onto Φ_0 every n iterations or so.

This incremental iteration is repeated n times in a phase called an *epoch*. After each epoch, we update the dual variables and quit after we believe we have identified the large elements of the diagonal of $\boldsymbol{C}$. Just as before, once we have identified the hott rows, we can form $\boldsymbol{W}$ by selecting these rows of $\boldsymbol{X}$. We can find $\boldsymbol{F}$ just as before, by solving (2). Note that this minimization can also be computed by incremental subgradient descent. The full procedure, called HOTTOPIXX, is described in Algorithm 4.

4.1 Sparsity and Computational Enhancements for Large Scale.

For small-scale problems, HOTTOPIXX can be implemented in a few lines of Matlab code. But for the very large data sets studied in Section 5, we take advantage of natural parallelism and a host of low-level optimizations that are also enabled by our formulation. As in any numerical program, memory layout and cache behavior can be critical factors for performance. We use standard techniques: in-memory clustering to increase prefetching opportunities, padded data structures for better cache alignment, and compiler directives to allow the Intel compiler to apply vectorization.

Note that the incremental gradient step (step 6 in Algorithm 4) only modifies the entries of $\boldsymbol{C}$ where $\boldsymbol{X}_{\cdot k}$ is nonzero. Thus, we can parallelize the algorithm with respect to updating either the rows or the columns of $\boldsymbol{C}$. We store $\boldsymbol{X}$ in large contiguous blocks of memory to encourage hardware prefetching. In contrast, we choose a dense representation of our localizing matrix $\boldsymbol{C}$; this choice trades space for runtime performance.

Each worker thread is assigned a number of rows of $\boldsymbol{C}$ so that all rows fit in the shared L3 cache. Then, each worker thread repeatedly scans $\boldsymbol{X}$ while marking updates to multiple rows of $\boldsymbol{C}$. We repeat this process until all rows of $\boldsymbol{C}$ are scanned, similar to the classical block-nested loop join in relational databases [22].

5 Experiments

Except for the speedup curves, all of the experiments were run on an identical configuration: a dual Xeon X650 (6 cores each) machine with 128GB of RAM. The kernel is Linux 2.6.32-131.

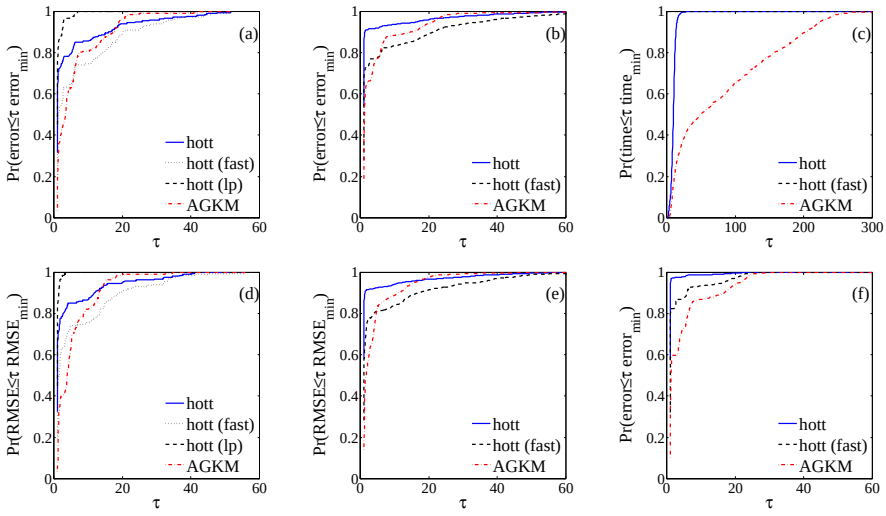

Figure 2: Performance profiles for synthetic data. (a) $(\infty, 1)$-norm error for 40×400 sized instances and (b) all instances. (c) is the performance profile for running time on all instances. RMSE performance profiles for the (d) small scale and (e) medium scale experiments. (f) $(\infty, 1)$-norm error for the $\eta \geq 1$. In the noisy examples, even 4 epochs of HOTTOPIXX is sufficient to obtain competitive reconstruction error.

In small-scale, synthetic experiments, we compared HOTTOPIXX to the AGKM algorithm and the linear programming formulation of Algorithm 3 implemented in Matlab. Both AGKM and Algorithm 3 were run using CVX [14] coupled to the SDPT3 solver [26]. We ran HOTTOPIXX for 50 epochs with primal stepsize 1e-1 and dual stepsize 1e-2. Once the hott topics were identified, we fit $\boldsymbol{F}$ using two cleaning epochs of incremental gradient descent for all three algorithms.

To generate our instances, we sampled r hott topics uniformly from the unit simplex in $\mathbb{R}^n$. These topics were duplicated d times. We generated the remaining $f - r(d+1)$ rows to be random convex combinations of the hott topics, with the combinations selected uniformly at random. We then added noise with $(\infty, 1)$-norm error bounded by $\eta \cdot \frac{\alpha^2}{20+13\alpha}$. Recall that AGKM algorithm is only guaranteed to work for $\eta < 1$. We ran with $f \in \{40, 80, 160\}$, $n \in \{400, 800, 1600\}$, $r \in \{3, 5, 10\}$, $d \in \{0, 1, 2\}$, and $\eta \in \{0.25, 0.95, 4, 10, 100\}$. Each experiment was repeated 5 times.

Because we ran over 2000 experiments with 405 different parameter settings, it is convenient to use the *performance profiles* to compare the performance of the different algorithms [7]. Let $\mathcal{P}$ be the set of experiments and $\mathcal{A}$ denote the set of different algorithms we are comparing. Let $Q_a(p)$ be the value of some performance metric of the experiment $p \in \mathcal{P}$ for algorithm $a \in \mathcal{A}$. Then the performance profile at τ for a particular algorithm is the fraction of the experiments where the value of $Q_a(p)$ lies within a factor of τ of the minimal value of $\min_{b \in \mathcal{A}} Q_b(p)$. That is,

$$P_a(\tau) = \frac{\#\left\{p \in \mathcal{P} \ : \ Q_a(p) \leq \tau \min_{a' \in \mathcal{A}} Q_{a'}(p)\right\}}{\#(\mathcal{P})}.$$

In a performance profile, the higher a curve corresponding to an algorithm, the more often it outperforms the other algorithms. This gives a convenient way to contrast algorithms visually.

Our performance profiles are shown in Figure 2. The first two figures correspond to experiments with $f = 40$ and $n = 400$. The third figure is for the synthetic experiments with all other values of f and n. In terms of $(\infty, 1)$-norm error, the linear programming solver typically achieves the lowest error. However, using SDPT3, it is prohibitively slow to factor larger matrices. On the other hand, HOTTOPIXX achieves better noise performance than the AGKM algorithm in much less time. Moreover, the AGKM algorithm must be fed the values of ϵ and α in order to run. HOTTOPIXX does not require this information and still achieves about the same error performance.

We also display a graph for running only four epochs (hott (fast)). This algorithm is by far the fastest algorithm, but does not achieve as optimal a noise performance. For very high levels of noise, however, it achieves a lower reconstruction error than the AGKM algorithm, whose performance

data set	features	documents	nonzeros	size (GB)	time (s)
jumbo	1600	64000	1.02e8	2.7	338
clueweb	44739	351849	1.94e7	0.27	478
RCV1	47153	781265	5.92e7	1.14	430

Table 1: Description of the large data sets. Time is to find 100 hott topics on the 12 core machines.

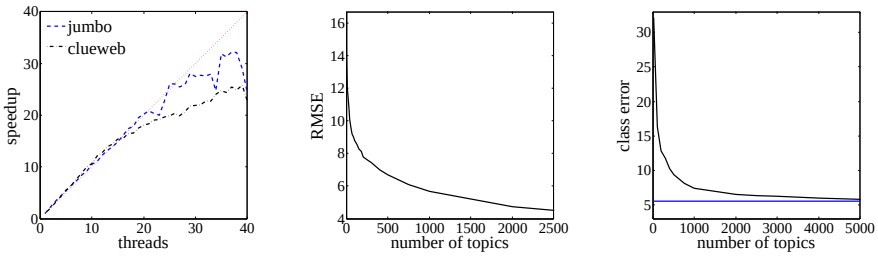

Figure 3: (left) The speedup over a serial implementation for HOTTOPIXX on the jumbo and clueweb data sets. Note the superlinear speedup for up to 20 threads. (middle) The RMSE for the clueweb data set. (right) The test error on RCV1 CCAT class versus the number of hott topics. The horizontal line indicates the test error achieved using all of the features.

degrades once η approaches or exceeds 1 (Figure 2(f)). We also provide performance profiles for the root-mean-square error of the nonnegative matrix factorizations (Figure 2 (d) and (e)). The performance is qualitatively similar to that for the $(\infty, 1)$-norm.

We also coded HOTTOPIXX in C++, using the design principles described in Section 4.1, and ran on three large data sets. We generated a large synthetic example (jumbo) as above with $r = 100$. We generated a co-occurrence matrix of people and places from the ClueWeb09 Dataset [2], normalized by TFIDF. We also used HOTTOPIXX to select features from the RCV1 data set to recognize the class CCAT [19]. The statistics for these data sets can be found in Table 1.

In Figure 3 (left), we plot the speed-up over a serial implementation. In contrast to other parallel methods that exhibit memory contention [21], we see superlinear speed-ups for up to 20 threads due to hardware prefetching and cache effects. All three of our large data sets can be trained in minutes, showing that we can scale HOTTOPIXX on both synthetic and real data. Our algorithm is able to correctly identify the hott topics on the jumbo set. For clueweb, we plot the RMSE Figure 3 (middle). This curve rolls off quickly for the first few hundred topics, demonstrating that our algorithm may be useful for dimensionality reduction in Natural Language Processing applications. For RCV1, we trained an SVM on the set of features extracted by HOTTOPIXX and plot the misclassification error versus the number of topics in Figure 3 (right). With 1500 hott topics, we achieve 7% misclassification error as compared to 5.5% with the entire set of features.

6 Discussion

This paper provides an algorithmic and theoretical framework for analyzing and deploying any factorization problem that can be posed as a linear (or convex) factorization localizing program. Future work should investigate the applicability of HOTTOPIXX to other factorization localizing algorithms, such as subspace clustering, and should revisit earlier theoretical bounds on such prior art.

Acknowledgments

The authors would like to thank Sanjeev Arora, Michael Ferris, Rong Ge, Nicolas Gillis, Ankur Moitra, and Stephen Wright for helpful suggestions. BR is generously supported by ONR award N00014-11-1-0723, NSF award CCF-1139953, and a Sloan Research Fellowship. CR is generously supported by NSF CAREER award under IIS-1054009, ONR award N000141210041, and gifts or research awards from American Family Insurance, Google, Greenplum, and Oracle. JAT is generously supported by ONR award N00014-11-1002, AFOSR award FA9550-09-1-0643, and a Sloan Research Fellowship.

References

[1] docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_nmf.htm.

[2] lemurproject.org/clueweb09/.

[3] www.mathworks.com/help/toolbox/stats/nnmf.html.

[4] S. Arora, R. Ge, R. Kannan, and A. Moitra. Computing a nonnegative matrix factorization – provably. To appear in STOC 2012. Preprint available at \arxiv.org/abs/1111.0952, 2011.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, Belmont, MA, 1997.

[6] V. Bittorf, B. Recht, C. Ré, and J. A. Tropp. Factoring nonnegative matrices with linear programs. Technical Report. Available at arxiv.org/1206.1270, 2012.

[7] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series A*, 91:201–213, 2002.

[8] D. Donoho and V. Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In *Advances in Neural Information Processing Systems*, 2003.

[9] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *Proceedings of CVPR*, 2012.

[10] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[11] E. Esser, M. Möller, S. Osher, G. Sapiro, and J. Xin. A convex model for non-negative matrix factorization and dimensionality reduction on physical space. *IEEE Transactions on Image Processing*, 2012. To appear. Preprint available at arxiv.org/abs/1102.0844.

[12] R. Gaujoux and C. Seoighe. NMF: A flexible R package for nonnegative matrix factorization. *BMC Bioinformatics*, 11:367, 2010. doi:10.1186/1471-2105-11-367.

[13] N. Gillis. Robustness analysis of hotttopixx, a linear programming model for factoring nonnegative matrices. arxiv.org/1211.6687, 2012.

[14] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, May 2010.

[15] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17:848–869, 1996.

[16] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[17] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[18] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

[19] D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[20] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106:697–702, 2009.

[21] F. Niu, B. Recht, C. Ré, and S. J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2011.

[22] L. D. Shapiro. Join processing in database systems with large main memories. *ACM Transactions on Database Systems*, 11(3):239–264, 1986.

[23] P. Smaragdis. Non-negative matrix factorization for polyphonic music transcription. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 177–180, 2003.

[24] M. Soltanolkotabi and E. J. Candès. A geometric analysis of subspace clustering with outliers. Preprint available at arxiv.org/abs/1112.4258, 2011.

[25] L. B. Thomas. Problem 73-14, rank factorization of nonnegative matrices. *SIAM Review*, 16(3):393–394, 1974.

[26] K. C. Toh, M. Todd, and R. H. Tütüncü. *SDPT3: A MATLAB software package for semidefinite-quadratic-linear programming*. Available from http://www.math.nus.edu.sg/~mattohkc/sdpt3.html.

[27] S. A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Joural on Optimization*, 20(3):1364–1377, 2009.