More Constraints, Smaller Coresets: Constrained Matrix Approximation of Sparse Big Data

Dan Feldman
Computer Science Department
University of Haifa
Haifa, Israel
dannyf.post@gmail.com

Tamir Tassa
Dep. of Mathematics and Computer Science
The Open University
Ra'anana, Israel
tamirta@openu.ac.il

ABSTRACT

We suggest a generic data reduction technique with provable guarantees for computing the low rank approximation of a matrix under some ℓ_z error, and constrained factorizations, such as the Non-negative Matrix Factorization (NMF). Our main algorithm reduces a given $n \times d$ matrix into a small, ε-dependent, weighted *subset* C of its rows (known as a *coreset*), whose size is independent of both n and d. We then prove that applying existing algorithms on the resulting coreset can be turned into $(1+\varepsilon)$-approximations for the original (large) input matrix. In particular, we provide the first linear time approximation scheme (LTAS) for the rank-one NMF.

The coreset C can be computed in parallel and using only one pass over a possibly unbounded stream of row vectors. In this sense we improve the result in [4] (Best paper of STOC 2013). Moreover, since C is a subset of these rows, its construction time, as well as its sparsity (number of non-zeroes entries) and the sparsity of the resulting low rank approximation depend on the maximum sparsity of an input row, and not on the actual dimension d. In this sense, we improve the result of Libery [21] (Best paper of KDD 2013) and answer affirmably, and in a more general setting, his open question of computing such a coreset.

We implemented our coreset and demonstrate it by turning Matlab's NMF off-line function that gets a matrix in the memory of a single machine, into a streaming algorithm that runs in parallel on 64 machines on Amazon's cloud and returns sparse NMF factorization. Source code is provided for reproducing the experiments and integration with existing and future algorithms.

1. INTRODUCTION

Matrix Factorization is a fundamental problem that was independently introduced in different contexts and applications. In Latent Semantic Analysis (LSA) or Probabilistic LSA (PLSA), the rows of an input matrix A correspond to documents in a large corpus of data, its columns correspond to terms that may appear in those documents, and the (i,j)th entry in A denotes the frequency of the jth term in the ith document (or other nonnegative statistics like `tf-idf`). Similarly, in computer vision applications the (i,j)th entry in A may represent the frequency of the ith feature in the jth image, and in social networks A may be the adjacency matrix of the graph.

An approximate k-ranked factorization such as the thin Singular Value Decomposition (SVD) of A enables to extract the k most important topics that are represented in that data and, by thus, obtain a concise representation of the data. In nonnegative matrix approximation, each topic is a linear combination of terms (columns of A), with only non-negative coefficients. Such techniques have been applied, for example, to image segmentation [20], information retrieval [15] and document clustering [26].

In this paper we deal with matrix factorization problems and their *constrained* variants. These problems can be described geometrically as computing a low-dimensional subspace S that minimizes the sum of distances in some power $z \geq 1$ to a given set P of n points in a d-dimensional Euclidean space, $\sum_{p\in P} \mathrm{dist}(p,S)^z$, where $\mathrm{dist}(p,S) := \min_{x\in S} \|p-x\|_2$. The input points are the rows of an $n \times d$ matrix A, and their projections on the subspace are the low rank approximation.

1.1 Problem Statement

Optimal (k,z)-subspace. For a given matrix $A \in \mathbb{R}^{n\times d}$ and a pair of integers $k, z \geq 1$, the *optimal (k,z)-subspace* of A is the k-dimensional subspace S^* of $\mathbb{R}^d$ that minimizes the sum of distances to the power of z from the set of rows $P = \{p_1, \cdots, p_n\}$ of A, i.e., S^* minimizes $\mathrm{cost}_z(P,S) := \sum_{p\in P}(\mathrm{dist}(p,S))^z$. A $(1+\varepsilon)$-approximation to this optimal (k,z)-subspace is a k-subspace S that minimizes $\mathrm{cost}_z(P,S)$, up to a multiplicative factor of $1+\varepsilon$,

$$\mathrm{cost}_z(P,S) \leq (1+\varepsilon)\mathrm{cost}_z(P,S^*).$$

Coreset. For a given $\varepsilon > 0$, a (k,z,ε)-coreset C for the matrix A is a subset of scaled rows from A such that using only C (without A) we can compute a $(1+\varepsilon)$-approximation the optimal (k,z)-subspace of A. In particular, if C is small, we can apply a (possibly inefficient) algorithm on C to get a $(1+\varepsilon)$-approximation for the optimal (k,z)-subspace of A.

Our main motivation is to compute such small coresets for obtaining the optimal subspace of A also under some additional constraints, as in the NMF problem.

Non-Negative Matrix Factorization (NMF). In one of the variants of the Non-Negative Matrix Factorization

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

DOI: http://dx.doi.org/10.1145/2783258.2783312.

problem we wish to project the rows of A on a line ℓ^* that minimizes their sum of distances to this line, with the additional constraint that ℓ^* is spanned by a vector u whose entries are non-negative, i.e, $\ell^* = \mathrm{Sp}(u)$ and $u \in L_{\geq 0}$ where

$$L_{\geq 0} = \left\{(x_1, \cdots, x_d) \in \mathbb{R}^d : x_1, \cdots, x_d \geq 0\right\}$$

is the non-negative orthant of $\mathbb{R}^d$. If the rows of A have only non-negative entries (as is common in many applications), then their projections on ℓ^* will also be non-negative. This results in a factorization $au^T \in \mathbb{R}^{n\times d}$ that approximates A where a and u are non-negative column vectors.

(L, k, z)-Subspace optimization. A natural generalization of NMF is the (L, k, z)-subspace optimization problem: given a convex cone L in $\mathbb{R}^d$, one looks for a k-dimensional subspace $S^*_L(P)$ that minimizes $\mathrm{cost}_z(P, S)$ over every k-dimensional subspace S that is spanned by k vectors in L. In the special case $k = 1$ we denote $S^*_L(P)$ by $\ell^*_L(P)$. A $(1+\varepsilon)$-approximation S to the optimal (L, k, z)-subspace is a subspace S that is spanned by k vectors in L such that $\mathrm{cost}_z(P, S) \leq (1+\varepsilon)\mathrm{cost}_z(P, S^*)$.

1.2 Related Work

Coresets. Following a decade of research, coresets (i.e., subsets of rows from the input matrix) that approximate *any* subspace in $\mathbb{R}^d$ were suggested in [6, 24] for sums of distances to the power of $z \geq 1$. However, their size is $|C| = O(dk^{O(z)}/\varepsilon^2)$, i.e, polynomial in d, which makes them less suitable for common large matrices (documents-terms, images-features, and adjacency matrices of graphs) where $d \sim n$. The dependency on d is due to the complexity, known as *pseudo-dimension*, of this family of subspaces.

Our work here is based on these results, but we show how to remove the d factor by reducing the family of subspaces and using a post-processing algorithm for extracting the desired subspace from the coreset. For the case $z = 2$, a deterministic coreset construction of size independent of d was very recently suggested in [11]. However, the construction is heavily tailored for this case of squared distances and SVD, unlike the above constructions and the construction in our paper.

To our knowledge, the only coreset for constrained factorization was given in [3] by Boutsidis and Drineas. They concentrated on the regression problem of minimizing $\|Ax - b\|$ over positive $x \in \mathbb{R}^d$. They used a coreset of size roughly $d\log(n)/\varepsilon^2$ to approximate this problem, and thus it is useful only if $d \ll n$. In addition, the coreset is proved to approximate only the optimal solution, and it is not clear how to apply it for the parallel or streaming setting without the required guarantee for merging a pair of coresets.

It was proved in [23] that for $L = \mathbb{R}^d$ and every $z \geq 1$, the optimal (L, k, z)-subspace is *spanned* by $\mathrm{poly}(k, 1/\varepsilon)$ input rows, using a non-constructive proof of existence. Our work generalizes this result to the constrained subspace case, and implies efficient constructions of these "spanning" sets. Other weaker versions of coresets appear in [7] in the context of k-means clustering.

Sketches. A *sketch* in our context is a set of vectors in $\mathbb{R}^d$ that can be used to compute a $(1+\varepsilon)$-approximation to the optimal subspace of a given matrix. Unlike coresets, if the input matrix is sparse, the sketch is in general not sparse.

A sketch of cardinality d that approximates the sum of squared distances ($z = 2$) for any subspace in $\mathbb{R}^d$ can be constructed with no approximation error ($\varepsilon = 0$) using the d rows of the matrix DV^T where $UDV^T = A$ is the thin SVD of A. It was proved in [10] that taking the first $O(k/\varepsilon)$ rows of DV^T yields such a sketch. This sketch can be constructed in both the streaming and parallel models by replacing ε with $1/\log n$ in the time and memory requirements.

In [21] (Best paper of KDD 2013) the above $\log n$ factor was removed from the streaming setting, as shown by the analysis of [13]. In [21, 13] the open question of whether we can construct such coresets (i.e., subsets of the input rows) of size independent of d was left. In this paper we answer this question affirmably, also for $z \geq 1$ and constrained optimization.

Sketches for $z \neq 2$ of size polynomial in d were suggested in [9, 5]. First sketch that can be constructed in input sparsity time was suggested in [4] (Best paper of STOC 2013). However, the sketch is dense and thus not applicable for merge-reduce techniques, and can not be computed in parallel or for streaming big data.

Non-negative Matrix Factorization (NMF). Unlike the optimal (unconstrained) subspace problem, Vavasis [25] has shown that for general values of k, NMF is NP-hard, even for $z = 2$ and $L = L_{\geq 0}$. In fact, there are almost no provable results in the study of the NMF problem. A notable exception is the paper by Arora et al. [1] which describes a solution under assumptions that may be hard to verify in practice and runs in time $2^{\left(\log(\frac{1}{\varepsilon})^{O(1)}\right)}\mathrm{poly}(n, d)$ where $\mathrm{poly}(n, d)$ is some implicit polynomial in n and d for $k = 1$.

We suggest a PTAS that takes time $O(nd^2)+O(d\cdot 2^{1/\varepsilon^{O(1)}})$ for this case and makes no assumptions on the input. For the case $k > 1$ running the algorithm in [1] on our corresponding coreset would improve their running time to $O(nd^2)$ for constants k and ε.

To our knowledge, all other papers that studied the NMF problem proposed heuristics for solving it based on greedy and rank-one downdating e.g. [2, 15, 16, 17, 18, 19, 20]. In the downdating approach, similarly to the SVD recursive algorithm for computing the (unconstrained) k-rank approximation, existing heuristics solve the (L, k, z)-subspace optimization problem by applying a solution to the optimal $(L, 1, z)$-subspace (line) in k iterations: one starts by solving the problem for $k = 1$, and using the obtained solution, the algorithm then proceeds to "update" the input matrix and then solve another 1-dimensional problem on the updated matrix.

All these off-line algorithms can now be applied on our small coresets to boost both their running time and performance. For example, these heuristics usually run a lot of iterations from an initial guess (seed) until they converge to a local minimum, and then repeat on other seeds for obtaining possibly better local minima. Running them on the coresets will allow using more iterations and seeds in less time, which in turn might improve the quality of the result. We can also run such off-line algorithms on streaming data and in parallel, by applying them on the coresets for the data seen so far as explained in [10].

1.3 Our results

Coresets for low-rank approximation. We prove that for every given matrix $A \in \mathbb{R}^{n\times d}$, an error parameter $\varepsilon > 0$, and an integer $z \geq 1$, there is a (k, z, ε)-coreset C of size

$\frac{k^{O(z)}}{\varepsilon}\log(\frac{1}{\varepsilon})$. In particular, unlike previous results, the size of the coreset C is independent of the input matrix size, d and n, and thus its sparsity depends on the maximum sparsity of each row and not on the size of A. Such a result is currently known only for the special case $z = 2$ [11]. In order to extract the approximated optimal subspace of A from the coreset C, one needs to compute the optimal subspace S^* of C. This subspace is not necessarily a good approximation to the optimal subspace of A. However, by applying our post-processing with C, S^* and $L = \mathbb{R}^d$ as its input yields the desired approximation (see Algorithm 1).

Coresets for constrained optimization. As in most of the existing algorithms for NMF (see related work), we use an algorithm for solving the case $k = 1$ in order to solve the more general (L, k, z)-subspace optimization for any $k > 1$. We prove that any such algorithm that computes a $(1+\varepsilon)$-approximation for the $(L, 1, z)$-subspace of a given matrix A, can be applied on our coreset C to obtain a $(1+\varepsilon)$-approximation ℓ to the optimal $(L, 1, z)$-subspace of A. This property of the coreset holds simultaneously for every cone L of $\mathbb{R}^d$. The $(L, 1, z)$-subspace (line) ℓ is computed by *(i)* applying the given approximation algorithm on the coreset C to get an output line ℓ^*, and then *(ii)* rotate ℓ^* for the given tuple $(C, L, \varepsilon, \ell^*)$ using our post-processing that returns the final approximated line ℓ; see Algorithm 1.

LTAS for constrained factorization. Unfortunately, although C is small, we could not find any polynomial time algorithm in the input that can provably compute the optimal subspace of C or its approximation, even for the case of Non-Negative Matrix Factorization where $k = 1$ and $L = L_{\geq 0}$. We thus design the first linear time approximation algorithm (LTAS) for computing such a $(1 + \varepsilon)$-approximation $\tilde{\ell}$ to the optimal $(L, 1, z)$-subspace, for any given cone L and $z \geq 1$. For the case $z = 2$ its running time is $O(nd^2) + O(d \cdot 2^{1/\varepsilon^{O(1)}})$ when applied on our coreset.

Unlike previous results that apply only for the Frobenius norm ($z = 2$), and non-negativity constraints, our coreset and LTAS can be computed for a more general family of constraints and any $z \geq 1$ to obtain a sparse approximation.

Streaming and parallel coreset construction. Via traditional map-reduce or merge-and-reduce techniques, our coresets can be computed using one pass over a possibly unbounded stream of rows, and in parallel over M machines (e.g. GPU, network, smartphones or a cloud). The size of the coreset and the required memory, as well as the update time per point is similar to the off-line case, where ε is replaced by $\varepsilon/\log n$ and n is the number of rows seen so far; see [14, 10] for details. Using M machines in parallel, the running time is reduced by a factor of M.

Experimental results. We implement both our coreset construction algorithm and the post-processing algorithm using Matlab. We then demonstrate how to boost this native off-line solver for NMF by applying it on our coreset and then using the post-processing algorithm on the resulting subspace to obtain an approximated solution for the original data. Note that we boost Matlab's solver only for demonstration: our coreset and post-processing algorithm can be applied on any existing or future heuristic that aims to compute approximation for a constrained factorization such as the NMF. Only the call to the Malab function should be replaced in our test code.

Although Matlab's NMF solver does not support parallel or streaming computation, by combining it with our coreset and the merge-and-reduce technique, we were able to compute the NMF of few datasets on 64 machines on Amazon cloud. On small datasets we obtained even *better* approximation compared to its result on the original (complete) data. While this phenomenon can not happen for running optimal (exact) solution, it usually occurs in the context of coresets.

Open source code. An open source code can be found in [12] for reproducing the experiments and applying the coreset for boosting existing and future algorithms.

1.4 Overview and organization

In Section 2 we provide the necessary background on coresets and give a birdseye view of our approach. In Section 3 we prove our main technical result: for the optimal constrained line ($(L, 1, z)$-subspace) that is spanned by L, there is always an approximated line that is spanned by few vectors in the union of L and the input points. In Section 4 we show the coreset construction which is similar to previous constructions except for the fact that its size is smaller and independent of d, and it approximates the constrained optimal subspace and not every subspace. Using exhaustive search to compute the optimal solution from the coreset we then obtain the first LTAS for the NMF problem. In Section 5 we demonstrate the efficiency of our algorithm and evaluate the quality of the results that it issues by replacing the LTAS by existing NMF heuristic. We conclude in Section 6. Due to space limitations, the proofs of most of the claims are omitted; they will be provided in the full version of this paper.

2. CORESETS

2.1 A General Framework for Coresets

In this section we summarize our main technical result and the new approach for computing coresets for constrained optimization problems. It is based on the general framework for coreset constructions from [6]. That framework assumes that we have a set P of n items that is called the *input set*, a (usually infinite) set $\mathbf{Q}$ which is called the family of *queries*, and a function dist that assigns every pair $p \in P$ and $q \in \mathbf{Q}$ a non-negative real number. For example, in the optimal $(1, 1)$-subspace problem, P is a set of n points in $\mathbb{R}^d$ (the rows of an $n \times d$ matrix), $\mathbf{Q}$ is the set of lines in $\mathbb{R}^d$, and $\text{dist}(p, \ell)$ is the distance from $p \in P$ to the line $\ell \in \mathbf{Q}$. Given an error parameter $\varepsilon \in (0, 1)$, the main algorithm of that framework outputs a pair (S, w), called an ε-coreset, which consists of a subset $S \subseteq P$ and a *weight function* $w : S \to [0, \infty)$ over the points in S such that the following holds. For every query $q \in \mathbf{Q}$, the sum of distances from P to q is approximated by the sum of weighted distances from C to q, up to $1 \pm \varepsilon$ factor, i.e.,

$$(1-\varepsilon)\sum_{p\in P}\text{dist}(p,q) \leq \sum_{p\in S} w(p)\text{dist}(p,q)$$
$$\leq (1+\varepsilon)\sum_{p\in P}\text{dist}(p,q).$$

It is proved in [6] that such a coreset (S, w) can be constructed by taking a non-uniform sample from P, where each sample $s \in S$ is chosen to be $p \in P$ with probability that is proportional to its *sensitivity* (sometimes called *importance*

or *leverage score*)

$$\sigma(p) := \max_{q \in \mathbf{Q}} \frac{\text{dist}(p,q)}{\sum_{p' \in P} \text{dist}(p',q)} .$$

Intuitively, if there is a query $q \in \mathbf{Q}$ where the distance from p to q dominates the sum of distances, then it is more important to select p to the coreset. Of course, the sensitivity of a point is at most 1, but we hope that not *all* points are important. The assigned weight for a sampled point $s = p$ is inversely proportional to $\sigma(p)$. It is then proved that given a probability of failure $\delta \in (0,1)$, this sampling procedure yields an ε-coreset (S, w) with probability at least $1 - \delta$, where the size of S is $|S| = O(\Sigma^2/\varepsilon^2)(v + \log(1/\delta))$. Here, $\Sigma := \sum_{p \in P} \sigma(p)$ is called the *total sensitivity* and v is called the *pseudo-dimension* of the pair $(P, \mathbf{Q})$ which represents the complexity of this pair in a VC dimension-type sense that is defined formally in [6]. Usually, (but not always) v is the number of parameters needed to define a query. For example, the family of lines in $\mathbb{R}^d$ has pseudo-dimension d; indeed, every such line can be defined by $O(d)$ parameters that represents its $O(d)$ degrees of freedom (direction and translation from the origin).

2.2 Weak Coresets

To obtain a coreset for the optimal (k, z)-subspace problem, we let P be the input set of n points in $\mathbb{R}^d$, $\mathbf{Q}$ be the set (family) of k-subspaces in $\mathbb{R}^d$, and $\text{dist}_z(p, X) = \min_{x \in X} \|p - x\|^z$, where $p \in P$ and $X \in \mathbf{Q}$. It was proved in [24] that the total sensitivity for this setting is independent of both n and d (see proof of Theorem 4.1 for details), but its pseudo-dimension is $O(d)$. Note that in order to compute an approximation to the optimal (k, z)-subspace for P using the coreset, we only need that the optimal (k, z)-subspace for the coreset will be a good approximation to the optimal (k, z)-subspace for P. In this sense, the requirement made in [24] that an ε-coreset must approximate *all* subspaces in $\mathbf{Q}$ is too strong.

In order to reduce the pseudo-dimension, a generalized version of a pseudo-dimension was suggested in [6]. It was used there to construct what is sometimes called a *weak coreset* [7, 8]. The resulting coreset C is weak in the sense that instead of approximating *every* query it approximates only a subset of queries $Q(C) \subseteq \mathbf{Q}$ that depends on the output coreset C. The *generalized pseudo-dimension* is then defined in [6] for such a function Q that assigns a subset $Q(S) \subseteq \mathbf{Q}$ for every subset $S \subseteq P$.

2.3 Novel approach: more constraints, smaller coresets

Variants of weak coresets for subspace approximation were used in [8, 6]. However, the size of the coreset in [8] was exponential in k, and, furthermore, the original input set P was needed to extract the approximate subspace from the coreset. In [6] the coreset was based on recursively projecting the points on their (k', z)-optimal subspace, for all $1 \leq k' \leq k$, and thus the output was a sketch (and not a subset of P). In addition, those weak coresets can not be used to approximate the constrained optimal subspace, because such an approximation may not be spanned by few input points.

To this end, we prove that a $(1 + \varepsilon)$-approximation to the constrained $(L, 1, z)$-subspace is spanned by a small set which is not a subset of the input, but still depends on only $O(2^{O(z)}/\varepsilon)$ input points and their projections on the cone L that represents the constraints. This implies a generalized pseudo-dimension of size independent of d for the constrained (L, k, z)-subspace problem. Based on the previous subsection and the bound on the total sensitivity from [24, 6] we thus obtain a coreset construction C of size independent of d.

While the construction is very similar to this in [24, 6] (for "strong" coresets), we can not simply apply an existing approximation algorithm on C, since the optimal (L, k, z)-subspace S^* for C is not necessarily a good approximation to the optimal (L, k, z)-subspace for P. This is because S^* may not be spanned by few points, i.e., $S^* \notin Q(C)$. Instead, we need to rotate S^* to obtain another subspace $\tilde{S}$ that is *(i)* a member in the set $Q(C)$ of queries that are approximated well in P, and *(ii)* approximates the optimal (k, z)-subspace S^* of C.

Indeed, our proof that there is a $(1 + \varepsilon)$-approximation to the constrained (L, k, z)-subspace is constructive. It gets as input an optimal subspace S^* that was computed for the small coreset C, and carefully and iteratively rotates it in order to obtain a subspace $\tilde{S}$ that satisfies properties *(i)* and *(ii)* above.

The subspace S^* can be computed using exhaustive search on C, which yields the first PTAS for the rank-1 NMF problem; See Theorem 4.2. Alternatively, we can apply existing heuristics to compute a k-subspace S that hopefully approximates the (L, k, z)-subspace S^*, and then apply our post-processing rotation; see Theorem 4.3.

For simplicity, in the next sections we focus on coresets for the case $z = 2$. The generalizations for $z \neq 2$ can be obtained by using the results from [24] that bound the total sensitivity for these cases. The generalization of our post-processing algorithm for $z \neq 2$ is also straightforward using the generalization of the triangle inequality to the weak triangle inequality (or Hölder inequality) as explained in the appendix of [23].

3. CONSTRAINED OPTIMIZATION

The constraints on the approximating subspaces have a geometric representation: the approximating subspace must have a basis of vectors that belong to some convex cone L in $\mathbb{R}^d$. A convex cone is a collection of half-lines in $\mathbb{R}^d$ that start at the origin, which forms a convex subset of $\mathbb{R}^d$; e.g., $L_{\geq 0}$ is a convex cone, and the corresponding constrained problem is NMF. See an illustration of a cone in Figure 1.

Let L be a convex cone, P be a set of points in $\mathbb{R}^d$, $\ell^* = \ell^*_L(P)$ be the solution to the $(L, 1, 2)$-subspace optimization problem, and $\varepsilon > 0$. In this section we construct a PTAS for finding a line ℓ that approximates ℓ^* in the sense that

$$\text{cost}(P, \ell) \leq (1 + \varepsilon)\text{cost}(P, \ell^*) , \qquad (1)$$

where hereinafter $\text{cost} := \text{cost}_z$ for $z = 2$ (namely, $\text{cost}(P, \ell) = \sum_{p \in P} (\text{dist}(p, \ell))^2$).

First (Section 3.3) we show that any arbitrary line ℓ^* (not necessarily $\ell^* = \ell^*_L(P)$) has a line ℓ that approximates it in the sense of inequality (1) and is spanned by a sparse subset of P; the construction of that sparse subset and of ℓ uses ℓ^* as an input. We then use this result in order to approximate the optimal line $\ell^* = \ell^*_L(P)$ even though it is not known upfront (Section 3.4).

We begin with some preliminary discussions in Sections 3.1–3.2. In what follows when we speak of lines in $\mathbb{R}^d$ we refer only to subspaces, namely lines that pass through the origin o of $\mathbb{R}^d$.

3.1 Projection of lines

An important notion in our analysis is that of a projection of a line onto a convex cone.

DEFINITION 3.1 (L-PROJECTION). *For any two lines in $\mathbb{R}^d$, ℓ_1 and ℓ_2, if α is the angle between them, $\beta(\ell_1,\ell_2) := \sin\alpha$. The function $\eta(\cdot)$ is called an* L-projection *if it gets a line ℓ and returns a line $\eta(\ell) \in L$, such that $\beta(\eta(\ell),\hat{\ell}) \le \beta(\ell,\hat{\ell})$ for all $\hat{\ell} \in L$.*

See in Figure 1 an example of a line ℓ, its projection $\eta(\ell)$ on the depicted cone, and the angle between $\eta(\ell)$ and ℓ, the sinus of which is denoted by $\beta(\eta(\ell),\ell)$.

LEMMA 3.2. *Let L be a convex cone. Let $\eta(\cdot)$ be a function that gets a line ℓ in $\mathbb{R}^d$ and returns a line $\ell' = \eta(\ell) \in L$ that minimizes $\beta(\ell',\ell)$ in L, where ties are broken arbitrarily. Then η is an L-projection.*

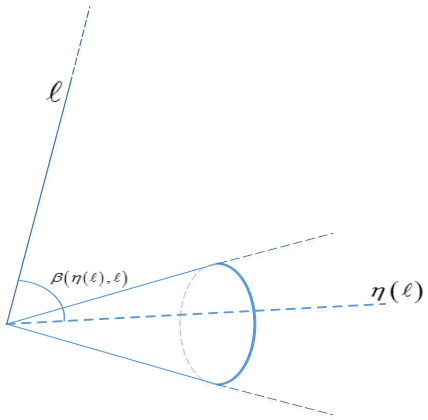


Figure 1: A cone and a projection of a line on the cone (Definition 3.1).

Lemma 3.2 can be used to compute an L-projection η for a given convex cone L. In what follows, we assume that $\eta(\ell)$ is computable in time $O(d)$ for any ℓ in $\mathbb{R}^d$.

3.2 Centroid sets

The following definitions play a central role in our analysis that follows.

DEFINITION 3.3. *Let ℓ and ℓ' be lines in $\mathbb{R}^d$ and let $0 < \varepsilon \le 1$. Denote by α and $\pi-\alpha$ the two angels between ℓ and ℓ'. Let $F_\alpha(\ell,\ell',\varepsilon)$ (resp. $F_{\pi-\alpha}(\ell,\ell',\varepsilon)$) denote the set of $\lceil 1/\varepsilon \rceil + 1$ lines in $\mathbb{R}^d$ that includes ℓ and ℓ' and partitions the angle α (resp. $\pi-\alpha$) to $\lceil 1/\varepsilon \rceil$ equal parts. Then $G(\ell,\ell',\varepsilon) := F_\alpha(\ell,\ell',\varepsilon) \cup F_{\pi-\alpha}(\ell,\ell',\varepsilon)$.*

An illustration of the fan of lines $G(\ell,\ell',\varepsilon)$ is given in Figure 2.

DEFINITION 3.4 (CENTROID SET). *Let $Q = (q_1,\cdots,q_m)$ be a sequence of m points from $\mathbb{R}^d \setminus \{o\}$, L be a convex cone, η be an L-projection, and $\varepsilon > 0$. Set $G_1 = \{\mathrm{Sp}(q_1)\}$ and then, for every $1 \le i \le m-1$ define $G_{i+1} = \bigcup_{\ell_i \in G_i} G(\eta(\ell_i), \mathrm{Sp}(q_{i+1}), \varepsilon)$. Then $\Gamma(Q,\eta,\varepsilon) := \bigcup_{\ell \in G_m} \eta(\ell)$ is called a* centroid set.

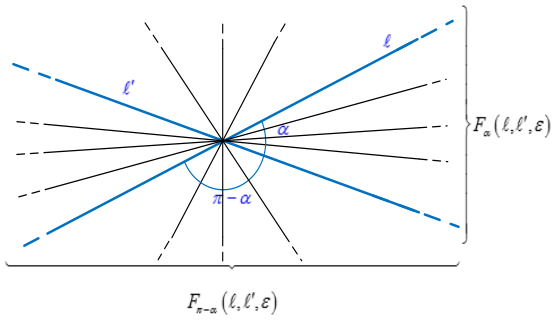


Figure 2: An illustration of the fan of lines defined in Definition 3.3.

LEMMA 3.5. *The size of the set $\Gamma(Q,\eta,\varepsilon)$ is $O(\frac{1}{\varepsilon^{m-1}})$, and the time to compute it is $O(\frac{d}{\varepsilon^{m-1}})$.*

3.3 Approximating an arbitrary line ℓ^*

Let P be a finite set of n points in $\mathbb{R}^d \setminus \{o\}$, L be a convex cone and η be an L-projection. Assume further that ℓ^* is any line in L and that $0 < \varepsilon < 1/2$. Algorithm ROTATELINE accepts inputs of the form (P,L,ε,ℓ^*) and outputs a line ℓ which approximates ℓ^* in the sense of inequality (1). The line ℓ belongs to L and it is contained in a centroid set (Definition 3.4) that could be described using a small (sparse) subset of points from P, where the size of that subset does not depend on n nor on d (Theorem 3.7).

Algorithm 1 ROTATELINE(P,L,ε,ℓ^*)

Input: A set $P = \{p_1,\cdots,p_n\}$ in $\mathbb{R}^d$; a convex cone L, $\varepsilon \in (0,\frac{1}{2})$, and a line $\ell^* \in L$.
Output: A line $\ell \in L$ that approximates ℓ^* in the sense of inequality (1), where $\ell \in \Gamma(Q,\eta,\varepsilon^2/216)$ for some sequence Q of points from P.

1: Set q_1 to be a point $q \in P$ that minimizes $\beta(\mathrm{Sp}(q),\ell^*)$.
2: Set $\ell_1 := \mathrm{Sp}(q_1)$.
3: Return IMPROVE$(P,L,\varepsilon,\ell^*,\ell_1)$

Algorithm 2 IMPROVE$(P,L,\varepsilon,\ell^*,\ell_i)$

1: Set $\ell := \eta(\ell_i)$.
2: **if** $\mathrm{cost}(P,\ell) \le (1+\varepsilon)\cdot\mathrm{cost}(P,\ell^*)$ **then**
3: Return ℓ.
4: **end if**
5: Compute a point $q_{i+1} \in P$ such that $\mathrm{dist}(q_{i+1},\ell) > (1+\frac{\varepsilon}{3})\cdot\mathrm{dist}(q_{i+1},\ell^*)$.
6: Compute a line $\ell_{i+1} \in G(\ell,\mathrm{Sp}(q_{i+1}),\frac{\varepsilon^2}{216})$ such that $\beta(\ell_{i+1},\ell^*) \le (1-\frac{\varepsilon}{12})\beta(\ell,\ell^*)$.
7: Return IMPROVE$(P,L,\varepsilon,\ell^*,\ell_{i+1})$.

Algorithm ROTATELINE calls the recursive sub-algorithm IMPROVE. The computations in Steps 5 and 6 of Algo-

rithm IMPROVE are well defined. Indeed, if we reach Step 5, then $\text{cost}(P,\ell) > (1+\varepsilon)\cdot\text{cost}(P,\ell^*)$. Therefore, there exists at least one point $q_{i+1} \in P$ for which $\text{dist}(q_{i+1},\ell)^2 > (1+\varepsilon)(\text{dist}(q_{i+1},\ell^*))^2$. Hence, $\text{dist}(q_{i+1},\ell) > (1+\varepsilon)^{1/2}\text{dist}(q_{i+1},\ell^*) > (1+\frac{\varepsilon}{3})\text{dist}(q_{i+1},\ell^*)$, where the latter inequality follows from our assumption that $\varepsilon < \frac{1}{2}$. As for Step 6, a line ℓ_{i+1} as sought there is guaranteed to exist in view of the next lemma.

LEMMA 3.6. *Let ℓ', ℓ^* be two lines in $\mathbb{R}^d$. Let $0 < \varepsilon \leq 1$ and $q \in \mathbb{R}^d$ be such that*

$$\text{dist}(q,\ell') > (1+\varepsilon)\cdot\text{dist}(q,\ell^*). \tag{2}$$

Then there exists a line $\ell \in G(\ell', \text{Sp}(q), \varepsilon^2/24)$ such that $\beta(\ell,\ell^*) \leq (1-\varepsilon/4)\cdot\beta(\ell',\ell^*)$.

THEOREM 3.7. *Assume that Algorithm* ROTATELINE *is applied on the input $(P, L, \varepsilon, \ell^*)$. Then:*

(i) The algorithm stops after at most

$$m(\varepsilon) = \left\lceil \frac{\log\frac{3}{\varepsilon}}{\log\left(\frac{1}{1-\frac{\varepsilon}{12}}\right)} \right\rceil + 1 = O\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\right) \tag{3}$$

recursive calls, and outputs a line $\ell \in L$.

(ii) The overall runtime of the algorithm is $O(\frac{nd\log\frac{1}{\varepsilon}}{\varepsilon}) + O(\frac{d}{\varepsilon^3}\log\frac{1}{\varepsilon})$.

(iii) $\text{cost}(P,\ell) \leq (1+\varepsilon)\cdot\text{cost}(P,\ell^*)$.

(iv) There exists a sequence Q of at most $m = m(\varepsilon)$ points from P, such that $\ell \in \Gamma(Q,\eta,\varepsilon^2/216)$.

Proof of (i). Let us denote $\alpha(\ell') := \beta(\ell',\ell^*)$ for every line ℓ' in $\mathbb{R}^d$. Fix i, $1 \leq i \leq m$, and consider the value of ℓ during the ith recursive call of the algorithm. As implied by Step 1 of IMPROVE, we have $\alpha(\ell) = \alpha(\eta(\ell_i))$. Therefore, by the properties of η, see Definition 3.1, we have $\alpha(\ell) \leq \alpha(\ell_i)$. By Step 6 of IMPROVE we have $\alpha(\ell_{i+1}) \leq (1-\frac{\varepsilon}{12})\alpha(\ell)$. Combining the last two inequalities yields $\alpha(\ell_{i+1}) \leq (1-\frac{\varepsilon}{12})\alpha(\ell_i)$. Hence, by Eq. (3),

$$\alpha(\ell_m) \leq \left(1-\frac{\varepsilon}{12}\right)^{m-1}\alpha(\ell_1) \leq \frac{\varepsilon}{3}\alpha(\ell_1). \tag{4}$$

We shall now prove that for every fixed $p \in P$, $\text{dist}(p,\ell_m) \leq (1+\frac{\varepsilon}{3})\cdot\text{dist}(p,\ell^*)$. By the triangle inequality,

$$\begin{aligned}\text{dist}(p,\ell_m) &\leq \text{dist}(p,\ell^*) + \text{dist}(\text{proj}(p,\ell^*),\ell_m)\\ &= \text{dist}(p,\ell^*) + \|\text{proj}(p,\ell^*)\|\cdot\alpha(\ell_m)\\ &\leq \text{dist}(p,\ell^*) + \|p\|\cdot\alpha(\ell_m).\end{aligned} \tag{5}$$

By the definition of ℓ_1 in Algorithm ROTATELINE, by the properties of η as an L-projection, and by the choice of q_1, we have

$$\alpha(\ell_1) = \alpha(\eta(\text{Sp}(q_1))) \leq \alpha(\text{Sp}(q_1)) \leq \alpha(\text{Sp}(p))\,. \tag{6}$$

By (4) and (6), we infer that $\alpha(\ell_m) \leq \varepsilon\alpha(\text{Sp}(p))/3$. Using the last inequality with (5) yields

$$\begin{aligned}\text{dist}(p,\ell_m) &\leq \text{dist}(p,\ell^*) + \|p\|\cdot\varepsilon\alpha(\text{Sp}(p))/3\\ &= \left(1+\frac{\varepsilon}{3}\right)\cdot\text{dist}(p,\ell^*).\end{aligned} \tag{7}$$

Squaring inequality (7) and summing over $p \in P$ yields

$$\begin{aligned}&\text{cost}(P,\ell_m) = \sum_{p\in P}(\text{dist}(p,\ell_m))^2\\ &\leq \sum_{p\in P}\left(1+\frac{\varepsilon}{3}\right)^2\cdot(\text{dist}(p,\ell^*))^2 < (1+\varepsilon)\cdot\text{cost}(P,\ell^*)\,,\end{aligned}$$

where the last inequality holds for all $\varepsilon < \frac{1}{2}$. Hence, if Algorithm IMPROVE did not stop before the mth recursive call, it would stop then. That concludes the proof of the first claim of the theorem. □

3.4 Approximating $\ell^* = \ell_L^*(P)$

In Section 3.3 we showed that every given line ℓ^* has an approximating line ℓ that can be described by a sparse subset of P. The construction of that sparse subset and of ℓ uses ℓ^* as an input. However, we wish to approximate the line $\ell^* = \ell_L^*(P)$ which is not known upfront. The claims of Theorem 3.7 enable us to overcome this problem. That theorem guarantees that ℓ^* has an approximating line ℓ in $\Gamma(Q,\eta,\varepsilon^2/216)$, where Q is some sequence of at most m (Eq. (3)) points from P and η is any L-projection. Hence, we infer that ℓ can be found in the set of lines $\hat{\Gamma}(P,\eta,\varepsilon)$ which is defined as follows.

DEFINITION 3.8. *Let P be a set of points in $\mathbb{R}^d \setminus \{o\}$, L be a convex cone, η be an L-projection, and $\varepsilon > 0$. Then $\hat{\Gamma}(P,\eta,\varepsilon) := \bigcup_Q \Gamma(Q,\eta,\varepsilon^2/216)$, where Γ is as defined in Definition 3.4, and the union is taken over all sequences $Q = (q_1,\ldots,q_m)$ of at most $m = m(\varepsilon)$ (see Eq. (3)) points from P.*

THEOREM 3.9. *There exists a PTAS for $\ell^* = \ell_L^*(P)$. Specifically, a line that approximates $\ell_L^*(P)$ in the sense of inequality (1) can be found in time $O(d\cdot poly(n))$ for any constant $\varepsilon > 0$.*

4. CORESET CONSTRUCTION

Algorithm 3, which we present and analyze below, describes how to construct a coreset C for the input set of points P and parameters $\varepsilon \in (0,1/2)$ and $\delta \in (0,1]$. A generalized version for distances to the power of $z \geq 1$ and for $k \geq 1$ is described in [24]. However, unlike [24], the size of C is independent of d.

To that end, the algorithm first constructs a pair (S,w), where S is a multiset of points from P (namely, it is a subset of P in which points may be repeated), and w is a weight function on S (Steps 1-12). Then, it converts the pair (S,w) into a set of points C in $\mathbb{R}^d$ (Step 13). That set is called the coreset.

The size of C is bounded by the size of S, which is shown in Step 7. That size depends on some sufficiently large constant, c_0, that can be determined from the analysis in this section and in the previous section. Our claims regarding the algorithm and the properties of the coreset C are summarized in Theorem 4.1

THEOREM 4.1. *Let P be a set of n points in $\mathbb{R}^d$, $\varepsilon \in (0,\frac{1}{2})$, $\delta \in (0,1]$, L be a convex cone, and η be an L-projection. Let C be the set that Algorithm 3 outputs when applied on the input $(P,\varepsilon;\delta)$. Then the following hold:*

(i) With probability at least $1-\delta$, every line $\ell \in \{\ell_L^*(P)\} \cup \hat{\Gamma}(C,\eta,\varepsilon)$ (Definition 3.8) satisfies

$$(1-\varepsilon)\cdot\text{cost}(P,\ell) \leq \text{cost}(C,\ell) \leq (1+\varepsilon)\cdot\text{cost}(P,\ell)\,, \tag{8}$$

Algorithm 3 CORESET$(P, \varepsilon; \delta)$

Input: $P = \{p_1, \cdots, p_n\} \subset \mathbb{R}^d$, $\varepsilon \in (0, 1/2)$, $\delta \in (0, 1]$.
Output: A set C of points in $\mathbb{R}^d$.

1: Use SVD to compute a line ℓ^* that minimizes $\text{cost}(P, \ell) := \sum_{p \in P} (\text{dist}(p, \ell))^2$ over all lines ℓ in $\mathbb{R}^d$ passing through the origin o.
2: For every $p \in P$: set $p' \in \mathbb{R}^d$ to be the projection of p onto ℓ^*.
3: Set $\mu := \frac{1}{n} \sum_{p \in P} p'$.
4: For every $p \in P$ set

$$\tilde{m}_p := \frac{2(\text{dist}(p, \ell^*))^2}{\sum_{q \in P} (\text{dist}(q, \ell^*))^2} + \frac{4\|p' - \mu\|^2}{\sum_{q \in P} \|q' - \mu\|^2} + \frac{16}{n}$$

5: For every $p \in P$ set $\Pr(p_i) := \dfrac{\tilde{m}_p}{\sum_{q \in P} \tilde{m}_q}$.
6: Set $S := \emptyset$.
7: Set $t := \frac{c_0}{\varepsilon^3} \log^2(1/\varepsilon) \log(1/\delta)$, where c_0 is a sufficiently large constant (that can be determined from the proof of Theorem 4.1).
8: **Repeat** t times:
9: Select a point $p \in P$ according to the probability distribution $\Pr(\cdot)$.
10: $S := S \cup \{p\}$.
11: Set $w(p) := \dfrac{n}{t \Pr(p)}$.
12: **End Repeat**
13: $C := \{\sqrt{j(p) w(p)}\, p \,|\, p \in S\}$, where $j(p)$ is the number of times that p appears in S.
14: Return C.

where $\text{cost}(P, \ell) = \sum_{p \in P} (\text{dist}(p, \ell))^2$.

(ii) $|C| = O\left(\frac{1}{\varepsilon^3} \log^2(\frac{1}{\varepsilon}) \log(\frac{1}{\delta})\right)$.

(iii) The runtime of CORESET *is* $O(nd^2) + O(\frac{1}{\varepsilon^3} \log^2(\frac{1}{\varepsilon}) \log(\frac{1}{\delta}) \log n)$.

Proof of (i) (sketch). The proof is based on [6, Theorem 4.1] that shows how to construct a set of size

$$O((v + \log(1/\delta)) \cdot t^2/\varepsilon^2), \tag{9}$$

which is an ε-coreset with probability at least $1 - \delta$. Here, v is the pseudo-dimension of the corresponding query space and t is its total sensitivity; see more details in Section 2. and full details in [6]. The total sensitivity t for the family of all possible k-dimensional subspaces of $\mathbb{R}^d$, and for distances to the power of z is $t = O(k^{O(z)})$ for $z \geq 1$. [24, Theorem 19].

The pseudo-dimension of this family is $v = O(d)$ and coresets of size linear in d that approximate *all* possible subspaces of $\mathbb{R}^d$ can be constructed by substituting $v = O(d)$ and the value of t above in (9). The coreset construction of [24] is essentially the same as in Algorithm 3 (for the case $z = 2$), except for the smaller size of the coreset that is output by our algorithm, which is independent of d. This is because our coreset aims to approximate only a constrained (smaller) subset of the family of lines, namely the lines in the centroid set $\Gamma(Q, \eta, \varepsilon)$ (Definition 3.4). This subset has pseudo-dimension v that depends on ε but not on d, as follows from Lemma 3.5. By Eq. (3), we conclude that the total sensitivity is $t = O\left(\frac{1}{\varepsilon^3} \log^2\left(\frac{1}{\varepsilon}\right) \log\left(\frac{1}{\delta}\right)\right)$. □

Applying exhaustive search on the small coreset yields a polynomial time approximation scheme:

THEOREM 4.2. *Let $\ell^* = \ell^*_L(P)$ be the line that minimizes the sum of squared distances to the points of P over every line ℓ in a convex cone L. Then, with probability at least $9/10$, a line ℓ such that*

$$\text{cost}(P, \ell) \leq (1 + \varepsilon) \cdot \text{cost}(P, \ell^*), \tag{10}$$

can be computed in $O(nd^2) + O(d \cdot 2^{1/\varepsilon^{O(1)}})$.

More generally, instead of applying our PTAS, we can use any existing heuristic that aims to compute the optimal line ℓ^*. In our experimental results we show that this approach can work even without a provable guarantee for the heuristics, given the proof above for the reduction to coresets. If the chosen heuristic has an approximation guarantee, then we get similar guarantees on its outcome when applied on the coreset. This is stated in the following theorem.

THEOREM 4.3. *Let $C :=$CORESET$(P, \varepsilon; \delta)$ and let ℓ' be a line that approximates the optimal solution of the coreset C, i.e., $\ell' \leq (1 + \varepsilon)\ell^*_L(C)$. Then, with probability at least $1 - \delta$, $\text{cost}(P, \ell') \leq (1 + \varepsilon) \cdot \ell^*_L(P)$.*

5. EXPERIMENTAL RESULTS

Hardware. We used Amazon EC2 AWS cloud service of 64 machines (workers) and its S3 storage support. The chosen machine type was Standard (cc2.8xlarge, 64 core). The service was run from a popular laptop Lenovo W520, CPU: Intel Core i7-3940XM, Memory: 32.0GB, DDR3-1600MHz SDRAM, Hard drive: 256GB 2.5“ (SATA3) Mobility Solid State Drive. Intelő vProŹ)

Software. All the code was written in Matlab 2013b (64bit), with the support of Matlab GPU toolbox (for most of the basic linear algebra functions). As for NMF heuristics, we use the nnmf function of Matlab's Statistics toolbox. This function supports mainly two heuristics: 'als' (the default) uses an alternating least-squares algorithm, and 'mult' that uses a multiplicative update algorithm. The other parameters of this function were assigned default values. The function returns a matrix H whose columns span a subspace of dimension at most k.

Data. We used a public dataset from the UCI repository, called “YouTube Multiview Video Games Dataset” [22]. This dataset contains instances, each described by 91 feature types, with class information for exploring multiview topics. The number of records (rows) is $n \sim 10^6$ and the number of features (columns, dimension) is $d = 91$.

Experiment. The rows of the original dataset were partitioned into batches on the local hard drive of our laptop computer. Then, an empty coreset tree was initialized in each worker on the cloud; see Fig. 3(a). The streaming was implemented by sending the batches one-by-one to the next worker on the cloud. This is done using a parallel execution of the command: “Send the ith batch to the ith worker” for $i = 1, \cdots, 64$, which is implemented in Matlab using the command “spmd”. Each of the 64 workers then computes its coreset, and the “spmd” command returned an array of 64 coresets. The main process then recomputes a coreset for these 64 coresets locally on the laptop, and runs Matlab's nnmf locally on the small coreset.

Evaluation. For comparison, we applied the experiment above also using uniform random sample T instead of our coreset S. In addition, we ran the nnmf function on the full original matrix A using a single computer. We then computed the sum of squared distances from the rows A to the subspaces that are spanned by the corresponding output matrices H_T, H_S and H_A, after applying the appropriate rotation. These sums are denoted by $\text{cost}(A, H_T)$, $\text{cost}(A, H_S)$, and $\text{cost}(A, H_A)$ respectively.

The empirical approximation error of the resulting coreset S is then $\varepsilon := \text{cost}(A, H_S)/\text{cost}(A, H_A) - 1$. Hence, $\text{cost}(A, H_S) = (1+\varepsilon)\text{cost}(A, H_A)$. Note that, since nnmf implements heuristics, the value of ε might be negative, which is indeed the case in Fig. 5(a) for the case of coresets. In this case the cost obtained by the coreset to the rows of A is actually *smaller* than the cost of a run on the full matrix A. Interestingly enough, we never got such a result by using uniform sampling.

We repeat the experiments for *(i)* uniform random sample (with replacement) instead of coresets, *(ii)* different values of k, *(iii)* different heuristics implementation of nnmf: 'als' (the default) uses an alternating least-squares algorithm, and 'mult' uses a multiplicative update algorithm. *(iv)* synthetic input data (standard Gaussian noise)

Results on synthetic data. We show results for one of the coreset constructions on one arbitrary worker. As expected, the properties of the coresets look essentially the same for the other workers and the resulting set. In Fig 3(b) we show how the memory (RAM) used by the worker increases when we add more points to its coreset from the synthetic. These results are common to other coresets papers. The "zig-zag" corresponds to the number of coresets in the tree that each new point needs to update.

For example, the first point in a streaming tree is updated in $O(1)$, however the 2^ith point for some $i \geq 1$ climbs up through $O(\log i)$ levels in the tree, so $O(\log i)$ coresets need to be merged. The error ε reduces roughly polynomially on $1/\varepsilon$ for the synthetic data, as shown in Fig. 5(b).

Results on the real dataset. Fig. 4 shows results for $k = 1$ dimensional subspace (number of columns of H). Note that *(i)* for small sample values the coreset already converges to zero. The error of the random sample is still very high and even increases with sample size for such values; *(ii)* the variance of the error seems to be much smaller for the coreset compared to the uniform sample; *(iii)* for large sample values (~ 2500), the coreset error is essentially zero and the ratio compared to the uniform sample seems to be very high.

Fig. 5(a) shows results for $k = 10$. Similar results were obtained for other values of k. The results on the coreset are almost always better than the uniform sample, on both of the nnmf algorithms. As noted above, for the case $k = 10$, the results on the coresets are sometimes better than the result on the original data ($\varepsilon < 0$). While we do not have a formal explanation, this is a common phenomena when running heuristics on coresets, although it cannot happened when computing optimal solutions. It seems that the heuristics compute local extreme points and sometimes converges to such "bad pitfalls" in the original data, while the coreset compressed the data and "cleaned" or smoothed a lot of these bad local extreme points.

6. CONCLUSION

We presented in this study a scalable and provable algorithm for general problems in constrained low rank approximation. Our technique is based on two cornerstones: A constructive proof that an approximation to the constrained 1-subspace is spanned by few input points and constraints; and a method to extract from the Big Data input a small coreset on which we may apply this construction, in lieu of the original large input. We extended the scope of previous art by considering general constraints and general metrics. Our algorithms significantly boost existing techniques with respect to runtime, applicability to streaming data, and by being embarrassingly parallelizable. Our experimental results indicate that coresets constitute a theoretical as well as a practical bridge that enables running traditional "small data" algorithms (which are typically off-line, non-parallel and sometimes inefficient) on modern Big Data. The provable guarantee which we offer is only with respect to the case where the approximating subspace is 1-dimensional. The extension of this algorithm to higher dimensions combines heuristical ingredients. An important goal for a future research is to devise a PTAS, or any other algorithm with a provable guarantee, also for the higher dimensional problems. Another target of future research is to demonstrate the benefits of our proposed techniques in various applications and settings.

7. REFERENCES

[1] Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization - provably. In *STOC*, 2012.

[2] S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review E*, 67:031902, 2003.

[3] C. Boutsidis and P. Drineas. Random projections for the nonnegative least-squares problem. *Linear Algebra and its Applications*, 431:760–771, 2009.

[4] K. L. Clarkson and D. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, 2013.

[5] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Sampling algorithms for l_2 regression and applications. In *SODA*, 2006.

[6] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, 2011.

[7] D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *SoCG*, 2007.

[8] D. Feldman, M. Monemizadeh, C. Sohler, and D. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *SODA*, 2010.

[9] D. Feldman, M. Monemizadeh, C. Sohler, and D. P. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *SODA*, 2010.

[10] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *SODA*, 2013.

[11] D. Feldman, M. Volkov, and D. Rus. Dimensionality reduction of massive sparse datasets using coresets. Technical report, 2015.

[12] Dan Feldman. NNMF coreset code in matlab. `http://people.csail.mit.edu/dannyf/`. Accessed: 2015-07-01.

[13] M. Ghashami, J. M. Liberty, E.and Phillips, and D. Woodruff. Frequent directions: Simple and deterministic matrix sketching. *arXiv preprint arXiv:1501.01711*, 2015.

[14] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, 2004.

[15] Thomas Hofmann. Probabilistic latent semantic analysis. In *UAI*, 1999.

[16] S. Hyvönen, P. Miettinen, and E. Terzi. Interpretable nonnegative matrix decompositions. In *KDD*, 2008.

[17] H. Kim and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity constrained least squares for microarray data analysis. *Bioinformatics*, 23:1495–1502, 2007.

[18] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *ICDM*, 2008.

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

[20] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2000.

[21] E. Liberty. Simple and deterministic matrix sketching. In *KDD*, 2013.

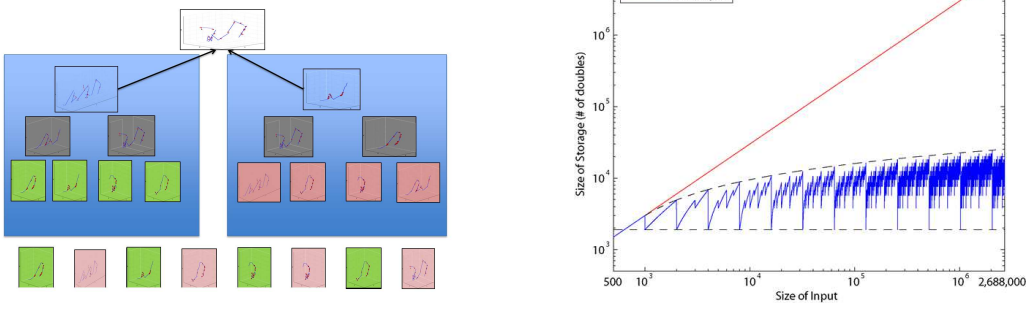
[22] Omid Madani, Manfred Georg, and David A. Ross. On using nearly-independent feature families for high precision and confidence. *Machine Learning*, 92:457–477, 2013.

[23] N. Shyamalkumar and K. Varadarajan. Efficient subspace approximation algorithms. In *SODA*, 2007.

[24] Kasturi Varadarajan and Xin Xiao. On the sensitivity of shape fitting problems. *arXiv preprint arXiv:1209.4893*, 2012.

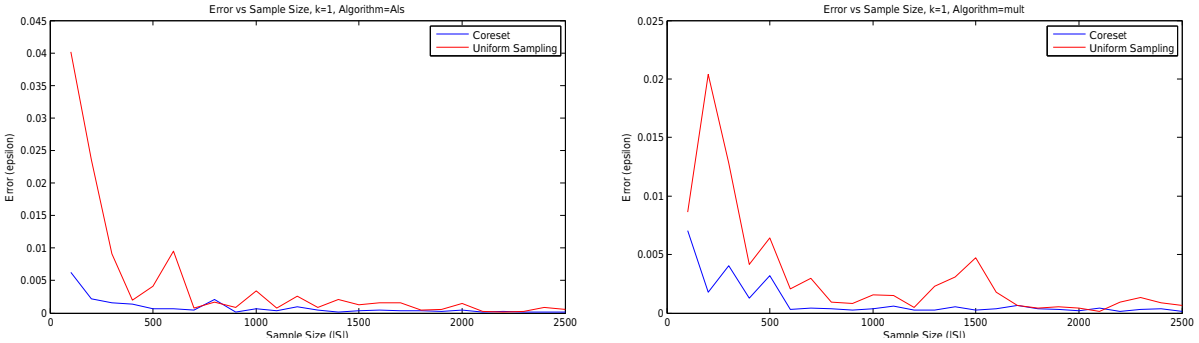
[25] Stephen A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20:1364–1377, 2009.

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



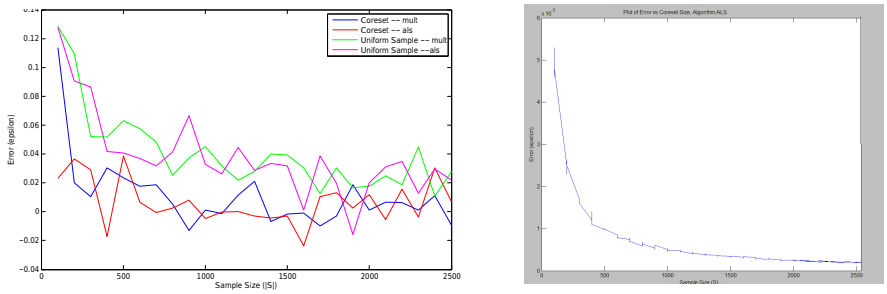
(a) *Coresets Tree* (b) *Memory Used*

Figure 3: (a) The input stream of points at the bottom is partitioned into odd and even subsets (green and pink). The odd subsets will be sent to the left worker and the even sets – to the right worker. Each worker maintains a merge-and-reduce tree of size roughly logarithmic in its input (blue frames). Each worker computes a single coreset for its tree and sends it to a central machine. The union of these two coresets forms a coreset for all the input points seen so far in the stream. (b) The memory that is used to save the coresets tree by each worker is only logarithmic in the input size n, since at most one coreset is stored in each of the $\log n$ levels.



(a) (b)

Figure 4: Approximation error ε of the input matrix A using the output subspace that was obtained from running Matlab's $nnmf(S, k, 'Algorithm', Alg)$ function. Here S is either the coreset or uniform random sample, $k = 1$, and Alg is the chosen Matlab's heuristic *als* (left) or *mult* (right).



(a) (b)

Figure 5: (a) Same as Figure 4, using $k = 10$ in both coreset construction and the call to $nnmf$. (b) Coreset empirical error on a fixed synthetic random Gaussian data and different coreset size.