Dimensionality Reduction of Massive Sparse Datasets Using Coresets

Dan Feldman Mikhali Volkov Daniela Rus

Abstract

In this paper we present a practical solution with performance guarantees to the problem of dimensionality reduction for very large scale sparse matrices. We show applications of our approach to computing the Principle Component Analysis (PCA) of any $n \times d$ matrix, using one pass over the stream of its rows. Our solution uses coresets: a scaled subset of the n rows that approximates their sum of squared distances to *every* k-dimensional *affine* subspace. An open theoretical problem has been to compute such a coreset that is independent of both n and d. An open practical problem has been to compute a non-trivial approximation to the PCA of very large but sparse databases such as the Wikipedia document-term matrix in a reasonable time. We answer both of these questions affirmatively. Our main technical result is a new framework for deterministic coreset constructions based on a reduction to the problem of counting items in a stream.

1 Introduction

Algorithms for dimensionality reduction usually aim to project an input set of d-dimensional vectors (database records) onto a $k \leq d-1$ dimensional affine subspace that minimizes the sum of squared distances to these vectors, under some constraints. Special cases include the Principle Component Analysis (PCA), Linear regression ($k = d-1$), Low-rank approximation (k-SVD), Latent Drichlet Analysis (LDA) and Non-negative matrix factorization (NNMF). Learning algorithms such as k-means clustering can then be applied on the low-dimensional data to obtain fast approximations with provable guarantees. To our knowledge, unlike SVD, there are no algorithms or coreset constructions with performance guarantees for computing the PCA of sparse $n \times n$ matrices in the streaming model, i.e. using memory that is poly-logarithmic in n. Much of the large scale high-dimensional data sets available today (e.g. image streams, text streams, etc.) are sparse. For example, consider the text case of Wikipedia. We can associate a matrix with Wikipedia, where the English words define the columns (approximately 1.4 million) and the individual documents define the rows (approximately 4.4 million documents). This large scale matrix is sparse because most English words do not appear in most documents. The size of this matrix is huge and no existing dimensionality reduction algorithm can compute its eigenvectors. To this point, running the state of the art SVD implementation from GenSim on the Wikipedia document-term matrix crashes the computer very quickly after applying its step of random projection on the first few thousand documents. This is because such dense vectors, each of length 1.4 million, use all of the computer's RAM capacity.

In this paper we present a dimensionality reduction algorithms that can handle very large scale sparse data sets such as Wikipedia and returns provably correct results. A long-open research question has been whether we can have a coreset for PCA that is both small in size and a subset of the original data. In this paper we answer this question affirmatively and provide an efficient construction. We also show that this algorithm provides a practical solution to a long-standing open practical problem: computing the PCA of large matrices such as those associated with Wikipedia.

2 Problem Formulation

Given a matrix A, a coreset C in this paper is defined as a weighted subset of rows of A such that the sum of squared distances from any given k-dimensional subspace to the rows of A is approximately the same as the sum of squared weighted distances to the rows in C. Formally,

For a compact set $S \in \mathbb{R}^d$ and a vector x in $\mathbb{R}^d$, we denote the Euclidean distance between x and its closest points in S by

$$\operatorname{dist}^2(x, S) := \min_{s \in S} \|x - s\|_2^2$$

For an $n \times d$ matrix A whose rows are $a_1, \ldots, a_n$, we define the sum of the squared distances from A to S by

$$\operatorname{dist}^2(A, S) := \sum_{i=1}^{n} \operatorname{dist}^2(a_i, S)$$

Definition 1 ((k, ε)-coreset)**.** *Given a $n \times d$ matrix A whose rows $a_1, \cdots, a_n$ are n points (vectors) in $\mathbb{R}^d$, an error parameter $\varepsilon \in (0, 1]$, and an integer $k \in [1, d-1] = \{1, \cdots, d-1\}$ that represents the desired dimensionality reduction, n (k, ε)-coreset for A is a weighted subset $C = \{w_i a_i \mid w_i > 0 \text{ and } i \in [n]\}$ of the rows of A, where $w = (w_1, \cdots, w_n) \in [0, \infty)^n$ is a non-negative weight vector, such that for every affine k-subspace S in $\mathbb{R}^d$ we have*

$$\left|\operatorname{dist}^2(A, S)) - \operatorname{dist}^2(C, S))\right| \leq \varepsilon \operatorname{dist}^2(A, S)). \tag{1}$$

That is, the sum of squared distances from the n points to S approximates the sum of squared weighted distances $\sum_{i=1}^{n} w_i^2 (\operatorname{dist}(a_i, S))^2$ to S. The approximation is up to a multiplicative factor of $1 \pm \varepsilon$. By choosing $w = (1, \cdots, 1)$ we obtain a trivial $(k, 0)$-coreset. However, in a more efficient coreset most of the weights will be zero and the corresponding rows in A can be discarded. The cardinality of the coreset is thus the sparsity of w, given by $|C| = \|w\|_0 := |\{w_i \neq 0 \mid i \in [n]\}|$.

If C is small, then the computation is efficient. Because C is a weighted subset of the rows of A, if A is sparse, then C is also sparse. A long-open research question has been whether we can have such a coreset that is both *of size* independent of the input dimension (n and d) and a *subset of the original input rows*.

2.1 Related Work

In [24] it was recently proved that an (k, ε) coreset of size $|C| = O(dk^3/\varepsilon^2)$ exists for every input matrix, and distances to the power of $z \geq 1$ where z is constant. The proof is based on a general framework for constructing different kinds of coresets, and is known as *sensitivity* [10, 17]. This coreset is efficient for tall matrices, since its cardinality is independent of n. However, it is useless for "fat" or square matrices (such as the Wikipedia matrix above), where d is in the order of n, which is the main motivation for our paper. In [5], the Frank-Wolfe algorithm was used to construct different types of coresets than ours, and for different problems. Our approach is based on a solution that we give to an open problem in [5], however we can see how it can be used to compute the coresets in [5] and vice versa. For the special case $z = 2$ (sum of squared distances), a coreset of size $O(k/\varepsilon^2)$ was suggested in [8] with a randomized version in [7] for a stream of n points that, unlike the standard approach of using merge-and-reduce trees, returns a coreset of size independent of n with a constant probability. These result minimizes the $\|\cdot\|_2$ error, while our result minimizes the Frobenius norm, which is always higher, and may be higher by a factor of d. After appropriate weighting, we can apply the uniform sampling of size $O(k/\varepsilon^2)$ to get a coreset with a small Frobenius error [14], as in our paper. However, in this case the probability of success is only constant. Since in the streaming case we compute roughly n coresets (formally, $O(n/m)$ coresets, where m is the size of the coreset) the probability that all these coresets constructions will succeed is close to zero (roughly $1/n$). Since the probability of failure in [14] reduces linearly with the size of the coreset, getting a constant probability of success in the streaming model for $O(n)$ coresets would require to take coresets of size that is no smaller than the input size.

There are many papers, especially in recent years, regarding data compression for computing the SVD of large matrices. None of these works addresses the fundamental problem of computing a sparse approximated PCA for a large matrix (in both rows and columns), such as Wikipedia. The reason is that current results use sketches which do no preserve the sparsity of the data (e.g. because of using random projections). Hence, neither the sketch nor the PCA computed on the sketch is sparse. On the other side, we define coreset as a small weighted subset of rows, which is thus sparse if the input is sparse. Moreover, the low rank approximation of a coreset is sparse, since each of its right singular vectors is a sum of a small set of sparse vectors. While there are coresets constructions as defined in this paper, all of them have cardinality of at least d points, which makes them impractical for large data matrices, where $d \geq n$. In what follows we describe these recent results in details.

The recent results in [8, 7] suggest coresets that are similar to our definition of coresets (i.e., weighted subsets), and do preserve sparsity. However, as mentioned above they minimize the 2-norm error and not the larger Frobesnius error, and maybe more important, they provide coresets for k-SVD (i.e., k-dimensional subspaces) and not for PCA (k-dimensional affine subspaces that might not intersect the origin). In addition [7] works with constant probability, while our algorithm is deterministic (works with probability 1).

Software. Popular software for computing SVD such as GenSim [21], redsvd [12] or the MATLAB sparse SVD function (`svds`) use sketches and crash for inputs of a few thousand of documents and a dimensionality reduction (approximation rank) $k < 100$ on a regular laptop, as expected from the analysis of their algorithms. This is why existing implementations (including Gensim) extract topics from large matrices (e.g. Wikipedia), based on low-rank approximation of only small subset of few thousands of selected words (matrix columns), and not the complete Wikipedia matrix.Even for $k = 3$, running the implementation of sparse SVD in Hadoop [23] took several days [13]. Next we give a broad overview of the very latest state of the dimensionality reduction methods, such as the Lanczoz algorithm [16] for large matrices, that such systems employ under the hood.

Coresets. Following a decade of research in [24] it was recently proved that an (ε, k)-coreset for low rank approximation of size $|C| = O(dk^3/\varepsilon^2)$ exists for every input matrix. The proof is based on a general framework for constructing different kinds of coresets, and is known as *sensitivity* [10, 17]. This coreset is efficient for tall matrices, since its cardinality is independent of n. However, it is useless for "fat" or square matrices (such as the Wikipedia matrix above), where d is in the order of n, which is the main motivation for our paper. In [5], the Frank-Wolfe algorithm was used to construct different types of coresets than ours, and for different problems. Our approach is based on a solution that we give to an open problem in [5].

Sketches. A *sketch* in the context of matrices is a set of vectors $u_1, \cdots, u_s$ in $\mathbb{R}^d$ such that the sum of squared distances $\sum_{i=1}^{n} (\mathrm{dist}(a_i, S))^2$ from the input n points to *every* k-dimensional subspace S in $\mathbb{R}^d$, can be approximated by $\sum_{i=1}^{n} (\mathrm{dist}(u_i, S))^2$ up to a multiplicative factor of $1 \pm \varepsilon$. Note that even if the input vectors $a_1, \cdots, a_n$ are sparse, the sketched vectors $u_1, \cdots, u_s$ in general are not sparse, unlike the case of coresets. A sketch of cardinality d can be constructed with no approximation error ($\varepsilon = 0$), by defining $u_1, \cdots, u_d$ to be the d rows of the matrix DV^T where $UDV^T = A$ is the SVD of A. It was proved in [11] that taking the first $O(k/\varepsilon)$ rows of DV^T yields such a sketch, i.e. of size independent of n and d.

The first sketch for sparse matrices was suggested in [6], but like more recent results, it assumes that the complete matrix fits in memory. Other sketching methods that usually do not support streaming include random projections [2, 1, 9] and randomly combined rows [20, 25, 22, 18].

The Lanczoz Algorithm. The Lanczoz method [19] and its variant [15] multiply a large matrix by a vector for a few iterations to get its largest eigenvector v_1. Then the computation is done recursively after projecting the matrix on the hyperplane that is orthogonal to v_1. However, v_1 is in general not sparse even A is sparse. Hence, when we project A on the orthogonal subspace to v_1, the resulting matrix is dense for the rest of the computations ($k > 1$). Indeed, our experimental results show that the MATLAB `svds` function which uses this method runs faster than the exact SVD, but crashes on large input, even for small k.

This paper builds on this extensive body of prior work in dimensionality reduction, and our approach uses coresets to solve the time and space challenges.

2.2 Key Contributions

Our main result is the first algorithm for computing an (k, ε)-coreset C of size independent of both n and d, for any given $n \times d$ input matrix. The algorithm takes as input a finite set of d-dimensional vectors, a desired approximation error ε, and an integer $k \geq 0$. It returns a weighted subset S (coreset) of k^2/ε^2 such vectors. This coreset S can be used to approximate the sum of squared distances from the matrix $A \in \mathbb{R}^{n \times d}$, whose rows are the n vectors seen so far, to any k-dimensional affine subspace in $\mathbb{R}^d$, up to a factor of $1 \pm \varepsilon$. For a (possibly unbounded) stream of such input vectors the coreset can be maintained at the cost of an additional factor of $\log^2 n$.

The polynomial dependency on d of the cardinality of previous coresets made them impractical for fat or square input matrices, such as Wikipedia, images in a sparse feature space representation, or adjacency matrix of a graph. If each row of in input matrix A has $O(\mathrm{nnz})$ non-zeroes entries, then the update time per insertion, the overall memory that is used by our algorithm, and the low rank approximation of the coreset S is $O(\mathrm{nnz} \cdot k^2/\varepsilon^2)$, i.e. independent of n and d.

We implemented our algorithm to obtain a low-rank approximation for the term-document matrix of Wikipedia with provable error bounds. Since our streaming algorithm is also "embarrassingly parallel" we run it on Amazon Cloud, and receive a significantly better running time and accuracy compared to existing heuristics (e.g. Hadoop/MapReduce) that yield non-sparse solutions.

The key contributions in this work are:

1. A new algorithm for dimensionality reduction of sparse data that uses a weighted subset of the data, and is independent of both the size and dimensionality of the data.
2. An efficient algorithm for computing such a reduction, with provable bounds on size and running time. (The project codebase will be open-sourced upon acceptance of this paper.)
3. A system that implements this dimensionality reduction algorithm and an application of the system to compute latent semantic analysis (LSA) of the entire English Wikipedia.

3 Technical Solution

Given a $n \times d$ matrix A, we propose a construction mechanism for a matrix C of size $|C| = O(k^2/\varepsilon^2)$ and claim that it is a (k, ε)-coreset for A. We use the following corollary for Definition 1 of a coreset, based on simple linear algebra that follows from the geometrical definitions (e.g. see [11]).

Property 1 (Coreset for sparse matrix)**.** *Let $A \in \mathbb{R}^{n \times d}$, $k \in [1, d-1]$ be an integer, and let $\varepsilon > 0$ be an error parameter. For a diagonal matrix $W \in \mathbb{R}^{n \times n}$, the matrix $C = WA$ is a (k, ε)-coreset for A if for every matrix $X \in \mathbb{R}^{d \times (d-k)}$ such that $X^T X = I$, we have*

$$\text{(i)} \quad \left| 1 - \frac{\|WAX\|}{\|AX\|} \right| \leq \varepsilon, \quad \text{and} \quad \text{(ii)} \quad \|A - WA\| < \varepsilon \operatorname{var}(A) \tag{2}$$

where $\operatorname{var}(A)$ is the sum of squared distances from the rows of A to their mean.

The goal of this paper is to prove that such a coreset (Definition 1) exists for any matrix A (Property 1) and can be computed efficiently. Formally,

Theorem 1. *For every input matrix $A \in \mathbb{R}^{n \times d}$, an error $\varepsilon \in (0, 1]$ and an integer $k \in [1, d-1]$:*

(a) there is a (k, ε)-coreset C of size $|C| = O(k^2/\varepsilon^2)$;

(b) such a coreset can be constructed in $O(k^2/\varepsilon^2)$ time.

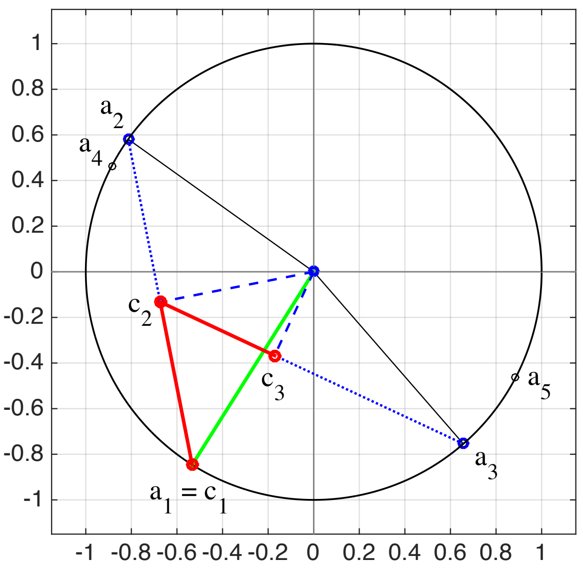
Theorem 1 is the formal statement for the main technical contribution of this paper. Sections 3–5 constitute a proof for Theorem 1.

To establish Theorem 1(a), we first state our two main results (Theorems 2 and 3) axiomatically, and show how they combine such that Property 1 holds. Thereafter we prove the these results in Sections 4 and 5, respectively. To prove Theorem 1(b) (efficient construction) we present an algorithm for computing a matrix C, and analyze the running time to show that the C can be constructed in $O(k^2/\varepsilon^2)$ iterations.

Algorithm 1 CORESET-SUMVECS(A, ε)

```
**Input:** $A$: $n$ input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$
**Input:** $\varepsilon \in (0,1)$: the approximation error
**Output:** $w \in [0,\infty)^n$: non-negative weights
$A \leftarrow A - \text{mean}(A)$
$A \leftarrow c\,A$ where $c$ is a constant s.t. $\text{var}(A) =$
$w \leftarrow (1, 0, \ldots, 0)$
$j \leftarrow 1,\ p \leftarrow A_j,\ J \leftarrow \{j\}$
$M_j = \left\{y^2 \mid y = A \cdot A_j^T\right\}$
**for** $i = 1, \ldots, n$ **do**
    $j \leftarrow \text{argmin}\,\{w_J \cdot M_J\}$
    $G \leftarrow W' \cdot A_J$ where $W'_{i,i} = \sqrt{w_i}$
    $\|c\| = \|G^T G)\|_F^2$
    $c \cdot p = \sum_{i=1}^{|J|} G\,p^T$
    $\|c - p\| = \sqrt{1 + \|c\|^2 - c \cdot p}$
    $\text{comp}_p(v) = 1/\|c-p\| - (c \cdot p)\,/\|c-p\|$
    $\|c - c'\| = \|c - p\| - \text{comp}_p(v)$
    $\alpha = \|c - c'\|/\|c - p\|$
    $w \leftarrow w(1 - |\alpha|)$
    $w_j \leftarrow w_j + \alpha$
    $w \leftarrow w/\sum_{i=1}^n w_i$
    $M_j \leftarrow \left\{y^2 \mid y = A \cdot A_j^T\right\}$
    $J \leftarrow J \cup \{j\}$
    **if** $\|c\|^2 \leq \varepsilon$ **then**
        **break**
    **end if**
**end for**
**return** $w$
```

(a) Coreset for sum of vectors algorithm



(b) Illustration showing first 3 steps of the computation

Let $A \in \mathbb{R}^{n \times d}$ be a matrix of rank d, and let $U\Sigma V^T = A$ denote its full SVD. Let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix. Let $k \in [1, d-1]$ be an integer. For every $i \in [n]$ let

$$v_i = \left(U_{i,1}, \cdots, U_{i,k}, \frac{U_{i,k+1:d}\Sigma_{k+1:d,k+1:d}}{\|\Sigma_{k+1:d,k+1:d}\|}, 1\right). \tag{3}$$

Then the following two results hold:

Theorem 2 (Coreset for sum of vectors)**.** *For every set of of n vectors $v_1, \cdots, v_n$ in $\mathbb{R}^d$ and every $\varepsilon \in (0,1)$, a weight vector $w \in (0,\infty)^n$ of sparsity $\|w\|_0 \leq 1/\varepsilon^2$ can be computed deterministically in $O(nd/\varepsilon)$ time such that*

$$\left\| \sum_{i=1}^n v_i - \sum_{i=1}^n w_i v_i \right\| \leq \varepsilon \sum_{i=1}^n \|v_i\|^2. \tag{4}$$

Section 4 establishes a proof for Theorem 2.

Theorem 3 (Coreset for Low rank approximation)**.** *For every $X \in \mathbb{R}^{d \times (d-k)}$ such that $X^T X = I$,*

$$\left| 1 - \frac{\|WAX\|^2}{\|AX\|^2} \right| \leq 5 \left\| \sum_{i=1}^n v_i v_i^T - W_{i,i} v_i v_i^T \right\|. \tag{5}$$

Section 5 establishes a proof for Theorem 3.

3.1 Proof of Theorem 1

Proof of Theorem 1(a). Replacing v_i with $v_i v_i^T$ and ε by $\varepsilon/(5d)$ in Theorem 2 yields

$$\left\| \sum_i v_i v_i^T - W_{i,i} v_i v_i^T \right\| \leq (\varepsilon/5d) \sum_{i=1}^n \|v_i v_i^T\|^2.$$

Combining this inequality with (4) gives

$$\left| 1 - \frac{\|WAX\|^2}{\|AX\|^2} \right| \leq 5 \left\| \sum_{i=1}^n v_i v_i^T - W_{i,i} v_i v_i^T \right\| \leq (\varepsilon/5d) \sum_{i=1}^n \|v_i v_i^T\|^2.$$

Thus the left-most term is bounded by the right-most term, which proves (2). This also means that $C = WA$ is a coreset for k-SVD, i.e., (non-affine) k-dimensional subspaces. To support PCA (affine subspaces) the coreset $C = WA$ needs to satisfy the expression in the last line of Property 1 regarding its mean. This holds using the last entry (one) in the definition of v_i (3), which implies that the sum of the rows is preserved as in equation (4). Therefore Property 1 holds for $C = WA$, which proves Theorem 1(a).

Claim Theorem 1(b) follows from simple analysis of Algorithm 2 that implements this construction. □

4 Coreset for Sum of Vectors $(k = 0)$

In order to prove the general result Theorem 1(a), that is the existence of a (k, ε)-coreset for any $k \in [1, d-1]$, we first establish the special case for $k = 0$. In this section, we prove Theorem 2 by providing an algorithm for constructing a small weighted subset of points that constitutes a general approximation for the sum of vectors.

To this end, we first introduce an intermediate result that shows that given n points on the unit ball with weight distribution z, there exists a small subset of points whose weighted mean is approximately the same as the weighted mean of the original points.

Let D^n denote the union over every vector $z \in [0, 1]^n$ that represent a distribution, i.e., $\sum_i z_i = 1$. Our first technical result is that for any finite set of unit vectors $a_1, \ldots, a_n$ in $\mathbb{R}^d$, any distribution $z \in D^n$, and every $\varepsilon \in (0, 1]$, we can compute a sparse weight vector $w \in D^n$ of sparsity (non-zeroes entries) $\|w\|_0 \leq 1/\varepsilon^2$.

Lemma 1. *Let $z \in D^n$ be a distribution over n unit vectors $a_1, \cdots, a_n$ in $\mathbb{R}^d$. For $\varepsilon \in (0, 1)$, a sparse weight vector $w \in D^n$ of sparsity $s \leq 1/\varepsilon^2$ can be computed in $O(nd/\varepsilon^2)$ time such that*

$$\left\| \sum_{i=1}^{n} z_i \cdot a_i - \sum_{i=2}^{n} w_i \, a_i \right\|_2 \leq \varepsilon. \tag{6}$$

Proof of Lemma 1. Please see Supplementary Material, Section A. □

We prove Theorem 2 by providing a computation of such a sparse weight vector w. The intuition for this computation is as follows. Given n input points $a_1,\ldots,a_n$ in $\mathbb{R}^d$, with weighted mean $\sum_i z_i \, a_i = \mathbf{0}$, we project all the points on the unit sphere. Pick an arbitrary starting point $a_1 = c_1$. At each step find the farthest point a_{j+1} from c_j, and compute c_{j+1} by projecting the origin onto the line segment $[c_j, a_{j+1}]$. Repeat this for $j = 1,\ldots,N$ iterations, where $N = 1/\varepsilon^2$. We prove that $\|c_i\|^2 = 1/i$, thus if we iterate $1/\epsilon^2$ times, this norm will be $\|c_{1/\epsilon^2}\| = \epsilon^2$. The resulting points c_i are a weighted linear combination of a small subset of the input points. The output weight vector $w \in D^n$ satisfies $c_N = \sum_{i=1}^{n} w_i \, a_i$, and this weighted subset forms the coreset.

Fig. 1a contains the pseudocode for Algorithm 1. Fig. 1b illustrates the first steps of the main computation (lines 9–26). Please see Supplementary Material, Section C for a complete line-by-line analysis of Algorithm 1.

Proof of Theorem 2. The proof of Theorem 2 follows by applying Lemma 1 after normalization of the input points and then post-processing the output. □

5 Coreset for Low Rank Approximation $(k > 0)$

In Section 4 we presented a new coreset construction for approximating the sum of vectors, showing that given n points on the unit ball there exists a small weighted subset of points that is a coreset for those points. In this section we describe the reduction of Algorithm 1 for $k = 0$ to an efficient algorithm for any low rank approximation with $k \in [1, d-1]$.

Conceptually, we achieve this reduction in two steps. The first step is to show that Algorithm 1 can be reduced to an inefficient computation for low rank approximation for matrices. To this end, we first prove Theorem 3, thus completing the existence clause Theorem 1(a).

Algorithm 2 CORESET-LOWRANK(A, k, ε)

```
 1: Input: $A$: A sparse $n \times d$ matrix
 2: Input: $k \in \mathbb{Z}_{>0}$: the approximation rank
 3: Input: $\varepsilon \in \left(0, \frac{1}{2}\right)$: the approximation error
 4: Output: $w \in [0, \infty)^n$: non-negative weights
 5: Compute $U\Sigma V^T = A$, the SVD of $A$
 6: $R \leftarrow \Sigma_{k+1:d,k+1:d}$
 7: $P \leftarrow$ matrix whose $i$-th row $\forall i \in [n]$ is
 8:     $P_i = (U_{i,1:k}, U_{i,k+1:d} \cdot \frac{R}{\|R\|_F})$
 9: $X \leftarrow$ matrix whose $i$-th row $\forall i \in [n]$ is
10:     $X_i = P_i / \|P_i\|_F$
11: $w \leftarrow (1, 0, \ldots, 0)$
```

(a) 1/2: Initialization

```
12: for $i = 1, \ldots, \lceil k^2/\varepsilon^2 \rceil$ do
13:     $j \leftarrow \text{argmin}_{i=1,\ldots,n}\{wXX_i\}$
14:     $a = \sum_{i=1}^{n} w_i (X_i^T X_j)^2$
15:     $b = \frac{1 - \|PX_j\|_F^2 + \sum_{i=1}^{n} w_i \|PX_i\|_F^2}{\|P\|_F^2}$
16:     $c = \|wX\|_F^2$
17:     $\alpha = (1 - a + b) \,/\, (1 + c - 2a)$
18:     $w \leftarrow (1 - \alpha) I_j + \alpha w$
19: end for
20: return $w$
```

(b) 2/2: Computation

Proof of Theorem 3. Let $\varepsilon = \| \sum_{i=1}^{n} (1 - W_{i,i}^2) v_i v_i^T \|$. For every $i \in [n]$ let $t_i = 1 - W_{i,i}^2$. Set $X \in \mathbb{R}^{d \times (d-k)}$ such that $X^T X = I$. Without loss of generality we assume $V^T = I$, i.e. $A = U\Sigma$, otherwise we replace X by $V^T X$. It thus suffices to prove that $\left| \sum_i t_i \|A_{i,:} X\|^2 \right| \le 5\varepsilon \, \|AX\|^2$. Using the triangle inequality, we get

$$\left| \sum_i t_i \|A_{i,:} X\|^2 \right| \le \left| \sum_i t_i \|A_{i,:} X\|^2 - \sum_i t_i \|(A_{i,1:k}, \mathbf{0}) X\|^2 \right| \tag{7}$$

$$+ \left| \sum_i t_i \|(A_{i,1:k}, \mathbf{0}) X\|^2 \right|. \tag{8}$$

We complete the proof by deriving bounds on (7) and (8), thus proving (5). For the complete proof, please see Supplementary Material, Section B. □

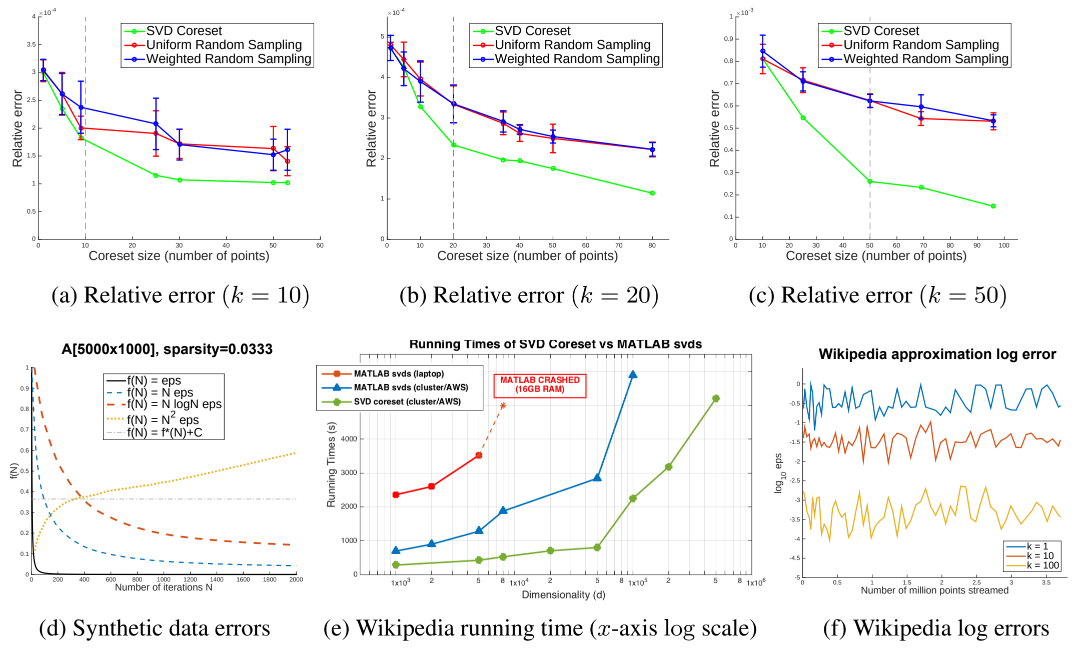
Together, Theorems 2 and 3 show that the error of the coreset is a $1 \pm \varepsilon$ approximation to the true weighted mean. By Theorem 3, we can now simply apply Algorithm 1 to the right hand side of (5) to compute the reduction. The intuition for this inefficient reduction is as follows. We first compute the outer product of each row vector x in the input matrix $A \in \mathbb{R}^{[n \times d]}$. Each such outer products $x^T x$ is a matrix in $\mathbb{R}^{d \times d}$. Next, we expand every such matrix into a vector, in $\mathbb{R}^{d^2}$ by concatenating its entries. Finally, we combine each such vector back to be a vector in the matrix $P \in \mathbb{R}^{n \times d^2}$. At this point the reduction is complete, however it is clear that this matrix expansion is inefficient.

The second step of the reduction is to transform the slow computation of running Algorithm 1 on the expanded matrix $P \in \mathbb{R}^{n \times d^2}$ into an equivalent and provably fast computation on the original set of points $A \in \mathbb{R}^d$. To this end we make use of the fact that each row of P is a sparse vector in $\mathbb{R}^d$ to implicitly run the computation in the original row space $\mathbb{R}^d$. We present Algorithm 2 and prove that it returns the weight vector $w = (w_1, \cdots, w_n)$ of a (k, ε)-coreset for low-rank approximation of the input point set P, and that this coreset is small, namely, only $O(k^2/\varepsilon^2)$ of the weights (entries) in w are non-zeros. Fig. 5 contains the pseudocode for Algorithm 2. Please see Supplementary Material, Section D for a complete line-by-line analysis of Algorithm 2.

6 Evaluation and Experimental Results

The coreset construction algorithm described in Section 5 was implemented in MATLAB. We make use of the redsvd package [12] to improve performance, but it is not required to run the system. We evaluate our system on two types of data: synthetic data generated with carefully controlled parameters, and real data from the English Wikipedia under the "bag of words" (BOW) model. Synthetic data provides ground-truth to evaluate the quality, efficiency, and scalability of our system, while the Wikipedia data provides us with a grand challenge for latent semantic analysis computation.

For our synthetic data experiments, we used a moderate size sparse input of (5000×1000) to evaluate the relationship between the error ε and the number of iterations of the algorithm N. We then compare our coreset against uniform sampling and weighted random sampling using the squared norms of U ($A = U\Sigma V^T$) as the weights. Finally, we evaluate the efficiency of our algorithm by



(a) Relative error ($k = 10$)   (b) Relative error ($k = 20$)   (c) Relative error ($k = 50$)

(d) Synthetic data errors   (e) Wikipedia running time (x-axis log scale)   (f) Wikipedia log errors

Figure 1: Experimental results for synthetic data (Fig. 1a–1d) and Wikipedia (Fig. 1e–Fig. 1f).

comparing the running time against the MATLAB `svds` function and against the most recent state of the art dimensionality reduction algorithm [7]. Figure 1a–1d show the exerimental results. Please see Supplementary Material, Section E for a complete description of the experiments.

6.1 Latent Semantic Analysis of Wikipedia

For our large-scale grand challenge experiment, we apply our algorithm for computing Latent Semantic Analysis (LSA) on the entire English Wikipedia. The size of the data is $n = 3.69$M (documents) with a dimensionality $d = 7.96$M (words). We specify a nominal error of $\varepsilon = 0.5$, which is a theoretical upper bound for $N = 2k/\varepsilon$ iterations, and show that the coreset error remains bounded. Figure 1f shows the log approximation error, i.e. sum of squared distances of the coreset to the subspace for increasing approximation rank $k = 1,\ 10,\ 100$. We see that the log error is proportional to k, and as the number of streamed points increases into the millions, coreset error remains bounded by k. Figure 1e shows the running time of our algorithm compared against `svds` for increasing dimensionality d and a fixed input size $n = 3.69$M (number of documents).

Finally, we show that our coreset can be used to create a topic model of 100 topics for the entire English Wikipedia. We construct the coreset of size $N = 1000$ words. Then to generate the topics, we compute a projection of the coreset onto a subspace of rank $k = 100$. Please see Supplementary Material, Section F for more details, including an example of the topics obtained in our experiments.

7 Conclusion

We present a new approach for dimensionality reduction using coresets. Our solution is general and can be used to project spaces of dimension d to subspaces of dimension $k < d$. The key feature of our algorithm is that it computes coresets that are small in size and subsets of the original data. We benchmark our algorithm for quality, efficiency, and scalability using synthetic data. We then apply our algorithm for computing LSA on the entire Wikipedia – a computation task hitherto not possible with state of the art algorithms. We see this work as a theoretical foundation and practical toolbox for a range of dimensionality reduction problems, and we believe that our algorithms will be used to construct many other coresets in the future. Our project codebase will be open-sourced upon acceptance of this paper, for reproducing the results and the benefit of the community.

References

[1] D. Achlioptas and F. Mcsherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9, 2007.

[2] S. Arora, E. Hazan, and S. Kale. A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 272–279. Springer, 2006.

[3] J. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[4] C. Carathéodory. Über den variabilitätsbereich der fourierschen konstanten von positiven harmonischen funktionen. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 32(1):193–217, 1911.

[5] K. L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

[6] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2013.

[7] M. B. Cohen, C. Musco, and J. W. Pachocki. Online row sampling. *CoRR*, abs/1604.05448, 2016.

[8] M. B. Cohen, J. Nelson, and D. P. Woodruff. Optimal approximate matrix product in terms of stable rank. *arXiv preprint arXiv:1507.02268*, 2015.

[9] P. Drineas and A. Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111(8):385–389, 2011.

[10] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proc. 41th Ann. ACM Symp. on Theory of Computing (STOC)*, 2010. Manuscript available at arXiv.org.

[11] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

[12] Google. redsvd. https://code.google.com/archive/p/redsvd/, 2011.

[13] N. P. Halko. *Randomized methods for computing low-rank approximations of matrices*. PhD thesis, University of Colorado, 2012.

[14] M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339. ACM, 1994.

[15] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

[16] C. Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[17] M. Langberg and L. J. Schulman. Universal ε approximators for integrals. *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[18] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[19] C. C. Paige. Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

[20] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM, 1998.

[21] R. Ruvrek, P. Sojka, et al. Gensimstatistical semantics in python. 2011.

[22] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

[23] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

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

[25] S. S. Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.

Supplementary Material

A Proof of Lemma 1

Lemma 1. *Let $z \in D^n$ be a distribution over n unit vectors $a_1, \cdots, a_n$ in $\mathbb{R}^d$. For $\varepsilon \in (0,1)$, a sparse weight vector $w \in D^n$ of sparsity $s \leq 1/\varepsilon^2$ can be computed in $O(nd/\varepsilon^2)$ time such that*

$$\left\| \sum_{i=1}^{n} z_i \cdot a_i - \sum_{i=2}^{n} w_i \, a_i \right\|_2 \leq \varepsilon. \tag{9}$$

We note that the Caratheodory Theorem [4] proves Lemma 1 for the special case $\varepsilon = 0$ using only $d+1$ points. Our approach and algorithm can thus be considered as an ε-approximation for the Caratheodory Theorem, to get coresets of size independent of d. Note that our Frank-Wolfe-style algorithm might run more than $d+1$ or n iterations without getting zero error, since the same point may be selected in several iterations. Computing in each iteration the closest point to the origin that is *spanned* by all the points selected in the previous iterations, would guarantee coresets of size at most $d+1$, and fewer iterations. Of course, the computation time of each iteration will also be much slower. '

Proof. We assume that $\sum_i z_i a_i = \mathbf{0}$, otherwise we subtract $\sum_j z_j a_j$ from each input vector a_i. We also assume $\varepsilon < 1$, otherwise the claim is trivial for $w = \mathbf{0}$. Let $w \in D^n$ such that $\|w\|_0 = 1$, and denote the current mean approximation by $c = \sum_i w_i a_i$. Hence, $\|c\|_2 = \|a_i\| = 1$.

The following iterative algorithm updates c in the end of each iteration until $\|c\|_2 < \varepsilon$. In the beginning of the Nth iteration the squared distance from c to the mean (origin) is

$$\|c\|_2^2 \in [\varepsilon, \frac{1}{N}]. \tag{10}$$

The average distance to c is thus

$$\sum_i z_i \|a_i - c\|_2^2 = \sum_i z_i \|a_i\|_2^2 + 2c^T \sum_i z_i a_i + \sum_i z_i \|c\|_2^2 = 1 + \|c\|_2^2 \geq 1 + \varepsilon \,,$$

where the sum here and in the rest of the proof are over $[n]$. Hence there must be a $j \in [n]$ such that

$$\|q_j - c\|_2^2 \geq 1 + \varepsilon. \tag{11}$$

Let r be the point on the segment between a_j and c at a distance $\rho := 1/\|a_j - c\|_2$ from a_j. Since $\|a_j - r\|_2 = \rho = \rho\|a_j - \mathbf{0}\|_2$, and $\|a_j - \mathbf{0}\|_2 = 1 = \rho\|a_j - c\|_2$, and $\angle(\mathbf{0}, a_j, c) = \angle(c, a_j, \mathbf{0})$, the triangle whose vertices are a_j, r and $\mathbf{0}$ is similar to the triangle whose vertices are a_j, $\mathbf{0}$, and c with a scaling factor of ρ. Therefore,

$$\|r - \mathbf{0}\|_2 = \rho \cdot \|\mathbf{0} - c\|_2 = \frac{\|c\|_2}{\|q_j - c\|_2}. \tag{12}$$

From (11) and (12), by letting c' be the closest point to $\mathbf{0}$ on the segment between a_j and c, we obtain

$$\|c'\|_2^2 \leq \|r\|_2^2 = \frac{\|c\|_2^2}{\|a_j - c\|_2^2} \leq \frac{\|c\|_2^2}{1+\varepsilon}.$$

Combining this with (10) yields

$$\|c'\|_2^2 \leq \frac{\frac{1}{N}}{1+\varepsilon} \leq \frac{\frac{1}{N}}{1+\frac{1}{N}} = \frac{1}{N+1}.$$

Since c' is a convex combination of a_j and c, there is $\alpha \in [0,1]$, such that $c' = \alpha a_j + (1-\alpha)c$. Therefore,

$$c' = \alpha a_j + (1-\alpha)\sum_i w_i a_i$$

and thus we have $c' = \sum_i w'_i a_i$, where $w' = (1-\alpha)w + \alpha e_j$, and $e_j \in D^n$ is the jth standard vector. Hence, $\|w'\|_0 = N+1$. If $\|c'\|_2^2 < \varepsilon$ the algorithm returns c'. Otherwise

$$\|c'\|_2^2 \in [\varepsilon, \frac{1}{N+1}] \tag{13}$$

We can repeat the procedure in (10) with c' instead of c and $N+1$ instead of N. By (29) $N+1 \leq 1/\varepsilon$ so the algorithm ends after $N \leq 1/\varepsilon$ iterations. After the last iteration we return the center $c' = \sum_{i=1}^n w'_i a_i$ so

$$\left\| \sum_i (z_i - w'_i) a_i \right\|_2^2 = \|c'\|_2^2 \leq \frac{1}{N+1} \leq \varepsilon.$$

□

B Proof of Theorem 3

Theorem 3 (Coreset for Low rank approximation)**.** *For every $X \in \mathbb{R}^{d\times(d-k)}$ such that $X^TX = I$,*

$$\left| 1 - \frac{\|WAX\|^2}{\|AX\|^2} \right| \leq 5 \left\| \sum_{i=1}^n v_i v_i^T - W_{i,i} v_i v_i^T \right\|. \tag{14}$$

Proof of Theorem 3. Let $\varepsilon = \| \sum_{i=1}^n (1 - W_{i,i}^2) v_i v_i^T \|$. For every $i \in [n]$ let $t_i = 1 - W_{i,i}^2$. Set $X \in \mathbb{R}^{d\times(d-k)}$ such that $X^TX = I$. Without loss of generality we assume $V^T = I$, i.e. $A = U\Sigma$, otherwise we replace X by V^TX. It thus suffices to prove that

$$\left| \sum_i t_i \|A_{i,:}X\|^2 \right| \leq 5\varepsilon \, \|AX\|^2. \tag{15}$$

Using the triangle inequality, we get

$$\left| \sum_i t_i \|A_{i,:}X\|^2 \right| \leq \left| \sum_i t_i \|A_{i,:}X\|^2 - \sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2 \right| \tag{16}$$

$$+ \left| \sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2 \right|. \tag{17}$$

We complete the proof by deriving bounds on (16) and (17).

Bound on (16)**:** It was proven in [1] that for every pair of k-subspaces S_1, S_2 in $\mathbb{R}^d$ there is $u \geq 0$ and a $(k-1)$-subspace $T \subseteq S_1$ such that the distance from every point $p \in S_1$ to S_2 equals to its distance to T multiplied by u. By letting S_1 denote the k-subspace that is spanned by the first k standard vectors of $\mathbb{R}^d$, letting S_2 denote the k-subspace that is orthogonal to each column of X, and $y \in \mathbb{R}^k$ be a unit vector that is orthogonal to T, we obtain that for every row vector $p \in \mathbb{R}^k$,

$$\|(p, \mathbf{0})X\|^2 = u^2 (py)^2. \tag{18}$$

After defining $x = \Sigma_{1:k,1:k} y / \|\Sigma_{1:k,1:k} y\|$, (16) is bounded by

$$\begin{aligned}
&\sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2 = \sum_i t_i \cdot u^2 \|A_{i,1:k} y\|^2 \\
&= u^2 \sum_i t_i \|A_{i,1:k} y\|^2 \\
&= u^2 \sum_i t_i \|U_{i,1:k} \Sigma_{1:k,1:k} y\|^2 \\
&= u^2 \|\Sigma_{1:k,1:k} y\|^2 \sum_i t_i \|(U_{i,1:k}) x\|^2.
\end{aligned} \tag{19}$$

The left side of (19) is bounded by substituting $p = \Sigma_{j,1:k}$ in (18) for $j \in [k]$, as

$$\begin{aligned}
u^2\|\Sigma_{1:k,1:k}y\|^2 &= \sum_{j=1}^{k} u^2(\Sigma_{j,1:k}y)^2 = \sum_{j=1}^{k} \|(\Sigma_{j,1:k}, \mathbf{0})X\|^2 \\
&= \sum_{j=1}^{k} \sigma_j^2 \|X_{j,:}\|^2 \leq \sum_{j=1}^{d} \sigma_d^2 \|X_{j,:}\|^2 \\
&= \|\Sigma X\|^2 = \|U\Sigma X\|^2 = \|AX\|^2. \qquad (20)
\end{aligned}$$

The right hand side of (19) is bounded by

$$\left|\sum_i t_i \|(U_{i,1:k})x\|^2\right| = \left|\sum_i t_i (U_{i,1:k})^T U_{i,1:k} \cdot xx^T\right| = \left|xx^T \cdot \sum_i t_i (U_{i,1:k})^T U_{i,1:k}\right|$$

$$\leq \|xx^T\| \cdot \|\sum_i t_i (U_{i,1:k})^T U_{i,1:k}\| \qquad (21)$$

$$\leq \|\sum_i t_i (v_{i,1:k})^T v_{i,1:k}\| \leq \|\sum_i t_i v_i^T v_i\| = \varepsilon \qquad (22)$$

where (21) is by the Cauchy-Schwartz inequality and the fact that $\|xx^T\| = \|x\|^2 = 1$, and in (22) we used the assumption $A_{i,j} = U_{i,j}\sigma_j = v_{i,j}$ for every $j \in [k]$.

Plugging (20) and (22) in (19) bounds (16) as

$$|\sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2| \leq \varepsilon \|AX\|^2. \qquad (23)$$

Bound on (17)**:** For every $i \in [n]$ we have

$$\begin{aligned}
&\|A_{i,:}X\|^2 - \|(A_{i,1:k}, \mathbf{0})X\|^2 \\
&= 2(A_{i,1:k}, \mathbf{0})XX^T(\mathbf{0}, A_{i,k+1:d})^T + \|(\mathbf{0}, A_{i,k+1:d})X\|^2 \\
&= 2A_{i,1:k}X_{1:k,:}(X_{k+1:d,:})^T(A_{i,k+1:d})^T + \|(\mathbf{0}, A_{i,k+1:d})X\|^2 \\
&= 2\sum_{j=1}^{k} A_{i,j}X_{j,:}(X_{k+1:d,:})^T(A_{i,k+1:d})^T + \|(\mathbf{0}, A_{i,k+1:d})X\|^2 \\
&= \sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d,:})^T \cdot \|\sigma_{k+1:d}\| v_{i,j}(v_{i,k+1:d})^T + \\
&\|\sigma_{k+1:d}\|^2 \|(\mathbf{0}, v_{i,k+1:d})X\|^2. \qquad (24)
\end{aligned}$$

Summing this over $i \in [n]$ with multiplicative weight t_i and using the triangle inequality, will bound (17) by

$$\begin{aligned}
&\left|\sum_i t_i \|A_{i,:}X\|^2 - \sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2\right| \\
&\leq \Big|\sum_i t_i \sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d,:})^T \qquad (25) \\
&\quad \cdot \|\sigma_{k+1:d}\| v_{i,j}(v_{i,k+1:d})^T\Big| \\
&\quad + \left|\sum_i t_i \|\sigma_{k+1:d}\|^2 \|(\mathbf{0}, v_{i,k+1:d})X\|^2\right|. \qquad (26)
\end{aligned}$$

The right hand side of (25) is bounded by

$$\begin{aligned}
&\left|\sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d})^T \cdot \|\sigma_{k+1:d}\| \sum_i t_i v_{i,j}(v_{i,k+1:d})^T\right| \\
&\le \sum_{j=1}^{k} 2\sigma_j \|X_{j,:}X_{k+1:d}\| \cdot \|\sigma_{k+1:d}\| \|\sum_i t_i v_{i,j} v_{i,k+1:d}\| && (27) \\
&\le \sum_{j=1}^{k} (\varepsilon\sigma_j^2 \|X_{j,:}\|^2 + \frac{\|\sigma_{k+1:d}\|^2}{\varepsilon} \|\sum_i t_i v_{i,j} v_{i,k+1:d}\|^2) && (28) \\
&\le 2\varepsilon \|AX\|^2, && (29)
\end{aligned}$$

where (27) is by the Cauchy-Schwartz inequality, (28) is by the inequality $2ab \le a^2 + b^2$. In (29) we used the fact that $\sum_i t_i(v_{i,1:k})^T v_{i,k+1:d}$ is a block in the matrix $\sum_i t_i v_i v_i^T$, and

$$\begin{aligned}
&\|\sigma_{k+1:d}\|^2 \le \|AX\|^2 \quad \text{and} \sum_{j=1}^{k} \sigma_j^2 \|X_{j,:}\|^2 \\
&= \|\Sigma_{1:k,1:k} X_{1:k,:}\|^2 \le \|\Sigma X\|^2 \le \|AX\|^2.
\end{aligned} \tag{30}$$

Next, we bound (26). Let $Y \in \mathbb{R}^{d\times k}$ such that $Y^TY = I$ and $Y^TX = \mathbf{0}$. Hence, the columns of Y span the k-subspace that is orthogonal to each of the $(d-k)$ columns of X. By using the Pythagorean Theorem and then the triangle inequality,

$$\begin{aligned}
&\|\sigma_{k+1:d}\|^2 |\sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})X\|^2| && (31) \\
=&\|\sigma_{k+1:d}\|^2 |\sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})\|^2 \\
&- \sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})Y\|^2| \\
&\le \|\sigma_{k+1:d}\|^2 |\sum_i t_i \|v_{i,k+1:d}\|^2| && (32) \\
&+ \|\sigma_{k+1:d}\|^2 |\sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})Y\|^2|. && (33)
\end{aligned}$$

For bounding (33), observe that Y corresponds to a $(d-k)$ subspace, and $(\mathbf{0}, v_{i,k+1:d})$ is contained in the $(d-k)$ subspace that is spanned by the last $(d-k)$ standard vectors. Using same observations as above (18), there is a unit vector $y \in \mathbb{R}^{d-k}$ such that for every $i \in [n]$ $\|(\mathbf{0}, v_{i,k+1:d})Y\|^2 = \|(v_{i,k+1:d})y\|^2$. Summing this over t_i yields,

$$\begin{aligned}
&|\sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})Y\|^2| = |\sum_i t_i \|v_{i,k+1:d} y\|^2| \\
&= |\sum_i t_i \sum_{j=k+1}^{d} v_{i,j}^2 y_{j-k}^2| = |\sum_{j=k+1}^{d} y_{j-k}^2 \sum_i t_i v_{i,j}^2|.
\end{aligned}$$

Replacing (33) in (31) by the last inequality yields

$$\begin{aligned}
&\|\sigma_{k+1:d}\|^2 |\sum_i t_i \|(\mathbf{0}, v_{i,k+1:d})X\|^2| \\
&\le \|\sigma_{k+1:d}\|^2 (|\sum_i t_i v_{i,d+1}^2| + \sum_{j=k+1}^{d} y_{j-k}^2 \|\sum_i t_i v_i v_i^T\|) && (34) \\
&\le \|\sigma_{k+1:d}\|^2 (\varepsilon + \varepsilon \sum_{j=k+1}^{d} y_{j-k}^2) \le 2\varepsilon \|AX\|^2, && (35)
\end{aligned}$$

Algorithm 1 CORESET-SUMVECS(A, ε)

```
1: Input: $A$: $n$ input points $a_1, \dots, a_n$ in $\mathbb{R}^d$
2: Input: $\varepsilon \in (0,1)$: the approximation error
3: Output: $w \in [0,\infty)^n$: non-negative weights
4: $A \leftarrow A - \text{mean}(A)$
5: $A \leftarrow c\,A$ where $c$ is a constant s.t. $\text{var}(A) = 1$
6: $w \leftarrow (1, 0, \dots, 0)$
7: $j \leftarrow 1,\ p \leftarrow A_j,\ J \leftarrow \{j\}$
8: $M_j = \left\{y^2 \mid y = A \cdot A_j^T\right\}$
9: for $i = 1, \dots, n$ do
10:     $j \leftarrow \text{argmin}\,\{w_J \cdot M_J\}$
11:     $G \leftarrow W' \cdot A_J$ where $W'_{i,i} = \sqrt{w_i}$
12:     $\|c\| = \|G^T G)\|_F^2$
13:     $c \cdot p = \sum_{i=1}^{|J|} G\,p^T$
14:     $\|c - p\| = \sqrt{1 + \|c\|^2 - c \cdot p}$
15:     $\text{comp}_p(v) = 1/\|c - p\| - (c \cdot p)\,/\|c - p\|$
16:     $\|c - c'\| = \|c - p\| - \text{comp}_p(v)$
17:     $\alpha = \|c - c'\|/\|c - p\|$
18:     $w \leftarrow w(1 - |\alpha|)$
19:     $w_j \leftarrow w_j + \alpha$
20:     $w \leftarrow w/\sum_{i=1}^n w_i$
21:     $M_j \leftarrow \left\{y^2 \mid y = A \cdot A_j^T\right\}$
22:     $J \leftarrow J \cup \{j\}$
23:     if $\|c\|^2 \le \varepsilon$ then
24:         break
25:     end if
26: end for
27: return $w$
```

where (34) follows since $\sum_i t_i v_{i,j}^2$ is an entry in the matrix $\sum_i t_i v_i v_i^T$, in (35) we used (30) and the fact that $\|y\|^2 = 1$. Plugging (29) in (25) and (35) in(20) gives the desired bound on (17) as

$$|\sum_i t_i \|A_{i,:} X\|^2 - \sum_i t_i \|(A_{i,1:k}, \mathbf{0}) X\|^2| \le 4\varepsilon \|AX\|^2.$$

Finally, using (23) in (16) and the last inequality in (17), proves the desired bound of (15).

□

C Analysis of Algorithm 1

Algorithm 1 contains the full listing of the construction algorithm for the coreset for sum of vectors.

Input: A: n input points $a_1, \dots, a_n$ in $\mathbb{R}^d$; $\varepsilon > 0$: the nominal approximation error.

Output: a non-negative vector $w \in [0,\infty)^n$ of only $O(1/\varepsilon^2)$ non-zeros entries which are the non-negative weights of the corresponding points selected for the coreset.

Analysis: The first step is to translate and scale the input points such that the mean is zero and the variance is 1 (lines 4–5). After initialization (lines 6–8), we begin the main iterative steps of the algorithm. First we find the index j of the farthest point from the initial point a_1. The next point added to the coreset is denoted by $p = a_j$. Next we compute $\|c - p\|$, the distance from the current point p to the previous center c. In order to do this we compute $G = W' \cdot A_J$ where J is the set of all previously added indices j, starting with the first point, and W' is defined in line 11. Note that G also gives us the error of the current iteration, $\varepsilon = \text{trace}(G\,G^T)$ (line 23). Next we find the point c' on the line from c to p that is closest to the origin, and find the distance between the current center c and the new center c' (lines 12–16). Finally, the ratio of distances between the current center,

Algorithm 2 CORESET-LOWRANK(A, k, ε)

```
 1: Input: $A$: A sparse $n \times d$ matrix
 2: Input: $k \in \mathbb{Z}_{>0}$: the approximation rank
 3: Input: $\varepsilon \in \left(0, \frac{1}{2}\right)$: the approximation error
 4: Output: $w \in [0, \infty)^n$: non-negative weights
 5: Compute $U\Sigma V^T = A$, the SVD of $A$
 6: $R \leftarrow \Sigma_{k+1:d,k+1:d}$
 7: $P \leftarrow$ matrix whose $i$-th row $\forall i \in [n]$ is
 8:     $P_i = (U_{i,1:k}, U_{i,k+1:d} \cdot \frac{R}{\|R\|_F})$
 9: $X \leftarrow$ matrix whose $i$-th row $\forall i \in [n]$ is
10:     $X_i = P_i / \|P_i\|_F$
11: $w \leftarrow (1, 0, \ldots, 0)$
12: for $i = 1, \ldots, \lceil k^2/\varepsilon^2 \rceil$ do
13:     $j \leftarrow \text{argmin}_{i=1,\ldots,n}\{wXX_i\}$
14:     $a = \sum_{i=1}^n w_i (X_i^T X_j)^2$
15:     $b = \dfrac{1 - \|PX_j\|_F^2 + \sum_{i=1}^n w_i \|PX_i\|_F^2}{\|P\|_F^2}$
16:     $c = \|wX\|_F^2$
17:     $\alpha = (1 - a + b) / (1 + c - 2a)$
18:     $w \leftarrow (1 - \alpha) I_j + \alpha w$
19: end for
20: return $w$
```

farthest point, and new center give us a value for α, the amount by which we update the coreset weights (lines 17–20).

The algorithm then updates the recorded indices J, update the lookup table M of previously computed row inner products for subsequent iterations, and repeat lines 10–26 until the loop terminates. The terminating conditions depend on the system specification – we may wish to bound the error, or the number of iterations. Moreover, if the update value α is below a specified threshold, we may also terminate the loop if such threshold is lower than a desired level of accuracy.

D Analysis of Algorithm 2

Algorithm 2 contains the full listing of the construction algorithm for the coreset for low rank approximation.

Input: A: n input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$; $k \geq 1$: the approximation rank; $\varepsilon > 0$: the nominal approximation error.

Output: a non-negative vector $w \in [0, \infty)^n$ of only $O(1/\varepsilon^2)$ non-zeros entries which are the non-negative weights of the corresponding points selected for the coreset.

Analysis: Algorithm 2 starts by computing the k-SVD of input matrix A (line 5). This is possible because we use the streaming model, so that the input arrives in small blocks. For each block we perform the computation to create its coreset. By merging the resulting coresets we preserve sparsity and can aggregate the coreset for A. Lines 7–8 use the k-SVD of this small input block to restructure the input matrix A into a combination of the columns of A corresponding to its k largest eigenvalues and the remaining columns of D, the singular values of A.

After initialization, we begin the main iterative steps of the algorithm. Note that lines 12–19 of Algorithm 2 are heavily optimized but functionally equivalent to lines 9–27 of Algorithm 1 – the end result in both cases is a computation of α at each iteration of the for loop, and an update to the vector of weights w. First we find the index j of the farthest point from the initial point a_1 (Line 13). The next point is implicitly added to the coreset is by updating w, and in turn affects the next farthest point as the computation wXX_i is performed iteratively. The variables a, b, c implicitly compute the distance from the current point p to the previous center q, the error of the current iteration ε, the point on the line from the p to q that is closest to the origin, and the distance between the current

Algorithm 3 MatrixProductApprox(A, k, ε)

```
Input:  A matrix $A \in \mathbb{R}^{n \times d}$, and an error parameter $\varepsilon > 0$.
Output: A vector $w \in [0, \infty)^n$ of $O(k/\varepsilon^2)$ non-zeros entries.
1  $X_u \leftarrow kI$
2  $X_l \leftarrow -kI$
3  $\delta_u \leftarrow \varepsilon + 2\varepsilon^2$
4  $\delta_l \leftarrow \varepsilon - 2\varepsilon^2$
5  Set $w \leftarrow (0, \cdots, 0)$
6  Set $Z$ to be the $d \times d$ zero matrix.
7  for $m \leftarrow 1, 2, \ldots$ to $k/\varepsilon^2$ do
8      Set
           $M_u \leftarrow ((X_u + \delta_u A^T A) - Z)^{-1}.$
9      Set
           $M_l \leftarrow (Z - (X_l + \delta_l A^T A))^{-1}.$
10     for $i = 1, 2, \ldots$ to $n$ do
11         Set $a_i \leftarrow$ a $d \times 1$ column vector which is the $i$th row of $A$
12         Set
               $\beta_l(i) \leftarrow \frac{a_i^T M_l A^T A M_l a_i}{\delta_l \mathrm{tr}(A M_l A^T A M_l A^T)} - a_i^T M_l a_i$
13         Set
               $\beta_u(i) \leftarrow \frac{a_i^T M_u A^T A M_u a_i}{\delta_u \mathrm{tr}(A M_u A^T A M_u A^T)} + a_i^T M_u a_i$
14     Compute $j \in [n]$ that maximizes $\beta_l(j) - \beta_u(j)$
15     Set $w_j \leftarrow \frac{1}{\beta_u(j)}$
16     Set $Z \leftarrow Z + w_j^2 a_j a_j^T$
17 return $w = (w_1, \cdots, w_n)$
```

Figure 2: Matrix product approximation algorithm [7]

center q and the new center q'. Finally, line 17 updates α and line 18 updates w using the new value of α.

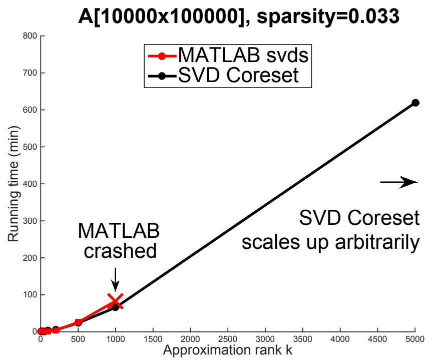
The algorithm terminates after k^2/ε^2 iterations, and we omit the explicit computation of ε since it is implied in the guarantees proven in the following section. As in Algorithm 1, the terminating conditions depend on the system specifications. We may wish to bound the error, or the number of iterations, or the update value α.

E Experimental Results – Synthetic Data

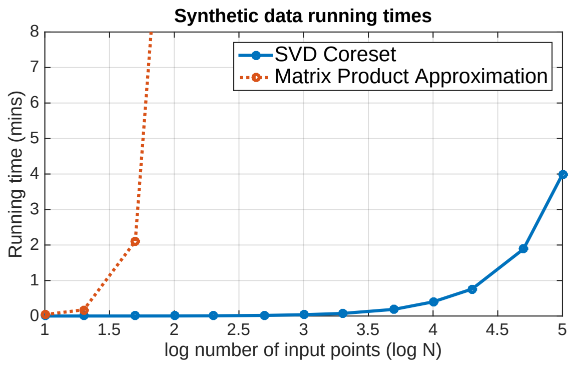
Synthetic data provides us with a ground-truth to objectively evaluate the quality, efficiency, and scalability of our system.

Approximation error. We carried out experiments on a moderate size sparse input of (5000×1000) to evaluate the relationship between the error ε and the number of iterations of the algorithm N. for a hyperplane coreset (i.e. $k = d$–1). Fig. 1d shows how the characteristic function of the approximation error $f(N)$ behaves with respect to increasing number of iterations N (normalized to $N = n$). Note that three of the plotted functions $f(N)$ converge as N increases, while the last one ramps up and then increases linearly. From this we conclude that ε decreases at a true rate somewhere between the rates of increase of $f(N) = N \log N$ and $f(N) = N^2$. The true characteristic $f^*(N) + C$ indicates the theoretical breakpoint between increasing and decreasing error.

We then compare our coreset against uniform sampling and weighted random sampling, using the squared norms of U ($A = U \Sigma V^T$) as the weights. Tests were carried out on a small subset of Wikipedia ($n = 1000,\ d = 257$K) to ensure representative data structure. Figure 1a–1c shows the results. As expected, approximation error decreases with coreset size, as well as the subspace rank. (Note that since our algorithm is deterministic, there is zero variance in the approximation error.)



(a) Relative error $(k = 10)$



(b) Relative error $(k = 20)$

Figure 3: Fig. 3a shows the runtimes of our coreset compared against MATLAB svds. Fig. 3b shows the runtimes of our coreset compared against the algorithm in [7].

Running time. We evaluate the efficiency of our algorithm by comparing the running time (coreset construction) against the built-in MATLAB `svds` function and against the most recent state of the art dimensionality reduction algorithm [7].

Algorithm 2 contains the pseudocode for our implementation of the algorithm presented in [7]. Fig. 3a shows the runtimes of our coreset compared against MATLAB svds. Fig. 3b shows the running time of our algorithm compared against Algorithm 3 run on synthetic data for the same set of input parameters. We used a fixed dimensionality $d = 1000$, approximation rank $k = 100$, sparsity 10^{-6} and evaluated construction time for increasing input size N. The results are plotted as a function of the log of the input size to show the order of magnitute difference in performance.

Besides the fact that our algorithm minimizes the Frobenius norm and support PCA, an important advantage of our technique compared to existing coreset constructions is that it is much numerically stable and faster in practice. For example, the result of [8] is based on the technique of [3]. This technique needs to compute many inverse of matrices during the computation, which makes it not only less stable but also very inefficient. Indeed, we implemented the coreset construction of [8] and the running time comparison to our algorithm for the same coreset size can be found in Fig. 3b. In conclusion, our algorithm is faster, numerically stable, and can be computed on practically unbounded size input data.

F Experimental Results – Latent Semantic Analysis of Wikipedia

For these experiments we used three types of machines:

1. Regular desktop computer with quad-core Intel Xeon E5640 CPU @2.67GHz, 6GB RAM (low spec).
2. Modern laptop with quad-core Intel i7-4500U CPU @1.8GHz, 16GB RAM (medium spec).
3. High-performance computing clusters on Amazon Web Services (AWS) as well as local clusters, e.g. an EC2 c3.8xlarge machine with 32-core Intel Xeon E5-2680v2 vCPU @2.8Ghz, 60GB RAM (high spec).

We compute the coreset using a buffer stream of size $N/2$, parallelized across 64 nodes on Amazon Web Services (AWS) clusters. The 64 individual coresets are then unified into a single coreset. Figure 1e shows the running time of our algorithm compared against `svds` for increasing dimensionality d and a fixed input size $n = 3.69$M (number of documents). Note that this is a log-scale plot of dimensionality against running time, so the differences in performance represent orders of magnitude. The desktop computer with 6GB RAM crashed for $d = 2000$ and was omitted from the plot. The same algorithm running on the cluster (blue plot) outperformed the laptop (red plot), which also quickly ran out of memory. Comparing `svds` computation on AWS against our coreset (green plot) highlights the difference in performance for identical computer architectures. As the dimensionality d increases, any algorithm dependent on d will eventually crash, given a large enough input.

We show that our coreset can be used to create a topic model of $k = 100$ topics for the entire English Wikipedia, with a fixed memory requirement and coreset size of just $N = 1000$ words. We compute the projection of the coresets on a subspace of rank k to generate the topics. Table 1 shows a selection of 10 of the most highly weighted words from 4 of the computed topics. The total running time, including coreset construction, merging and topic extraction was 140.66 min.

A cursory glance at the words suggests that the "themes" of these topics are (1) urban planning, (2) economy and finance, (3) road safety, (4) entertainment. This serves as a qualitative proof of concept that our system can produce meaningful results topics on very large datasets. We view this result optimistically, as proof of concept that our system can be used to compute a topic model of the English language. A more objective analysis would involve using a corpus of tagged documents as a ground truth, projecting the corresponding vectors onto our topics, and comparing the classification error against topics computed by other systems. This is the subject of our ongoing work.

Topic 1	Topic 2	Topic 3	Topic 4
US	credit	drivers	comedy
highway	risk	distracted	nominated
bridge	plan	phone	actress
road	union	driver	awards
river	interest	text	television
traffic	rating	car	episode
downtown	earnings	brain	musical
bus	capital	accidents	writing
harbor	liquidity	visual	tv
street	asset	crash	directing
...	...	...	...

Table 1: Example of the highest-weighted words from 4 topics of the $k = 100$ topic model of Wikipedia computed by our algorithm