Learning Big (Image) Data via Coresets for Dictionaries

Dan Feldman, Micha Feigin, and Nir Sochen

Abstract. Signal and image processing have seen in the last few years an explosion of interest in a new form of signal/image characterization via the concept of sparsity with respect to a dictionary. An active field of research is dictionary learning: the representation of a given large set of vectors (e.g. signals or images) as linear combinations of only few vectors (patterns). To further reduce the size of the representation, the combination are usually sparse, i.e, each signal is a linear combination of only small number of patterns.
This paper suggests a new computational approach to the problem of dictionary learning, known in computational geometry as *coresets*. Coreset is a small smart non-uniform sample from the input signals, such that the quality of any given dictionary with respect to the input can be approximated via the coreset. In particular, the optimal dictionary for the input can be approximated by learning the coreset. Since the coreset is small, the learning is faster. Moreover, using merge-and-reduce, the coreset can be constructed for streaming signals that do not fit in memory and can also be computed in parallel.
We apply our coresets for dictionary learning of images using the K-SVD algorithm and bound their size and approximation error analytically. Our simulations demonstrate gain factor of up to 60 in computational time with the same, and even better, performance. We also demonstrate our ability to perform computations on larger patches and high-definition images, where the traditional approach breaks down.

Key words: Sparsity, dictionary learning, K−SVD, coresets

1 Introduction

One of the major problems in image processing is image characterization. By image characterization we mean a system that gets a two-dimensional function or, in the discrete case, a matrix, and provides a probability measure as to whether or not this function/matrix is an image. We are still far from achieving this ultimate goal, yet few breakthroughs where recorded since the inception of image processing as a branch of scientific research. In the past, many characterization used the decay rate of the coefficients of certain transformations. That led to a characterization in a linear space of functions. In the last decade, a new approach that involves redundant representations and sparsity seems promising. In this framework, a signal is represented again as a superposition of signals. But unlike the representation with a basis of a linear space, the number of basic

signals (a.k.a. atoms) in this new approach exceeds the dimension of the signal such that a given signal may have many different representations. Uniqueness is achieved only for a subset of signals which can be represented with a limited number of atoms, called sparse signals. For this class of signals the sparsest representation is unique. This approach shifts the focus of attention from the general law of decay of coefficients to the outliers of such behavior, namely the large coefficients of such an expansion. The class of sparse signals does not form a linear space which reflects the non-linearity of the set of images. At the same time, we still use linear techniques which helps a lot in practice.

Similar sparsity approaches have been used as well for problems such as image and texture classification, image compression and image denoising.

The sparsity approach has appealing features for image processing, but it suffers from a few problems. First it is clear that sparsity is a notion which is attached to a given dictionary. Clearly, there is no one universal dictionary that can represent any image in a sparse way. This calls upon the need to construct dictionaries for each class of images or for each application. Constructing a dictionary for a large number of images from the same class/application goes under the name dictionary learning and is an active field of research.

Because of the prohibitive computational time and space complexity, as well as numerical instabilities in computing with a large number of large sized matrices, sparsity techniques are applied to small images only. In fact, 8×8 to 16×16 is the most common sizes in image processing. It means that these are patches of images rather than images themselves. Larger patches would also reduce their number for the training process.

Moreover, one may wish to construct dictionaries for the same class of images. Implicitly using the approximate self-similarity nature of images it is customary to use the patches of an image as a class of similar patches and to construct a dictionary per image. Here, again, the curse of limited space and time interfere and high definition images (megapixel resolution common in modern cameras) have a huge number of patches of such a small size which makes the dictionary learning task computationally prohibitive.

This paper brings the spell of coresets to cure the curse of space and time limitations. Informally, a coreset C for a set of elements Y is a compressed representation of Y that well approximates the original data in some problem-dependent sense. The given problem is then solved on the much smaller coreset C and the resulting solution is applicable to the original set Y. This is done by using C to give approximate answers for queries about Y.

Corset techniques were first introduced in the computational geometry field, and in the recent years have been used to solve some well known open problems in computer science and machine learning. The subject became more mature theoretically in the last few years. Corsets present a new approach to optimization in general and have huge success especially in tasks which use prohibitively large computation time and/or memory space. In particular, coresets suggest ways to use existing serial algorithms for distributed (parallel) computing, and provide solutions under the the streaming model, where the space (memory) for solving

the problem at hand is significantly smaller than its input size (and suffices for storing the coreset, but not the complete input).

Coresets for learning dictionaries are the main topic of this paper, and we demonstrate our ideas on the K−SVD method introduce by Aharon et al [4]. The K-SVD is a greedy algorithm designed to solve the following optimization problem. Given a positive values T_0, K and a matrix $Y \in \mathbb{R}^{d\times n}$ of atoms (column vectors) y, we want to find a dictionary $D \in \mathbb{R}^{d\times K}$ and a sparse coefficient matrix $X \in \mathbb{R}^{d\times k}$ that minimizes

$$\arg\min_{D,X} \|Y - DX\|_F \text{ s.t. } \forall i, \|x_i\|_0 \leq T_0. \tag{1}$$

Where the atoms y are the columns of the matrix Y, the vector x_i is the ith column of X, $\|\cdot\|_F$ is the Frobenius norm (the sum of squared entries in the matrix) and $\|x_i\|_0 \leq T_0$ is the sparsity of x_i, i.e, x_i contains at most T_0 non-zeros.

The algorithm solves alternate optimization problems, alternating between finding a better D (using the SVD algorithm) while preserving the required sparsity of X and find the best sparse X given a dictionary D (alternating matching pursuit is used to compute X, although alternative methods are suggested as well).

Formally, a coreset for the problem defined in (1) is a matrix C such that

$$\|Y - DX\|_F \sim \|Y - CX\|_F$$

for every dictionary $D \in \mathbb{R}^{d\times k}$ and the matrix X that minimizes its approximation cost $\|Y - DX\|_F$. Here, the symbol $\sim$ denotes a multiplicative factor of $1 \pm \varepsilon$. For example, it is easy to verify that Y is an ε-coreset of itself. However, an ε-coreset C is efficient if its number of columns is $c \ll n$ and the optimization problem can be solved more efficiently on the coreset, that is much smaller, without sacrificing too much accuracy.

Our coreset size depends solely on the required accuracy and problem parameters and not on the size of the input data Y. In fact, due to stability issues and sensitivity to initial conditions with greedy algorithms, our method provides a better solution at a lower running time. This is due to a smaller learning set and the ability to run the algorithm with multiple initial conditions due to the lower running times. Note though that only we provide a method to find the optimal D. We later compute X normally in one iteration of the original algorithm.

2 Our contribution

We first prove that for every input matrix Y of n atoms there is a small coresets for learning dictionaries. The construction is based on non-uniform random sampling of the n atoms, based on an approximate optimal solution D_0. The size of the coreset is independent on n, but only on the desired approximation error ε of the coreset, the size dk of the desired dictionary, and the probability δ that the construction will fail; see Theorem 1 for details.

Since it is not clear how to compute such an approximation D_0, in practice we compute a different initial dictionary D_0 and prove that the coreset admits an additive error of ε multiplied by the cost of D_0; see Corollary 2.

Since the approximation error depends on the quality of D_0, we suggest two options to compute D_0: 1) Use a single iteration of the K-SVD algorithm as an initial guess for a dictionary, and 2) Compute an approximation to a related problem called projective clustering.

In fact, in our experiments we use the most simple version of projective clustering which is the mean D_0 of the n atoms, and still get significantly improved results; see Section 5. In these experiments, we measure the quality, size and running time of our coresets, compared to both uniform random sampling and the full input data.

Our results also implies streaming and parallel version of K-SVD by computing the coreset in a well-known merge-and-reduce technique and then running the existing (off-line) K-SVD algorithm on the resulting coreset; see Section 5 for more details and applications.

3 Dictionaries and projective clustering

The main coreset techniques that we use in this paper usually applied in the context of *projective clustering* from computational geometry. Projective clustering is a constrained version of the problem that K-SVD designed to solve, but has several approximation algorithms with bounded error and running times, see [18, 38] and references therein. In this section we compare the problem of dictionary learning to projective clustering for this reason, and also because our algorithm and approximation error is based on an approximation to the projective clustering problem.

(α, β)-approximations. As explained in the previous section, the K-SVD algorithm is designed to solve the optimization problem in (1): given Y and T_0, compute

$$\arg\min_{D,X} \|Y - DX\|_F \text{ s.t } \forall i, \|x_i\|_0 \leq T_0.$$

The aim of the coreset construction is to select the atoms in Y with the higher importance, in a sense that will be defined formally in the proof of Theorem 1. However, it depends on the optimal solution of (1) whose computation is the main reason that we construct the coreset in the first place. To find a leeway from this chicken-and-egg situation, rough approximation D_0 for the optimal solution is computed. In computer science, such approximation is called α-approximation, or more generally (α, β)-approximation. In the context of our problem, α-approximation is a dictionary D_0 that minimizes (1) up to a multiplicative factor of $\alpha > 0$. For (α, β)-approximation, we also allow the dictionary to have βk columns for some $\beta > 1$, rather then k, and the cost is again larger by a factor of α compared to the optimal dictionary *with exactly k columns.* The size and quality of the coreset depends on the values of α and β. Roughly

speaking, if we use α-approximation D_0 for constructing the coreset, its additive error will be a factor of only $\varepsilon\alpha$ due to a sample of size quadratic in $1/\varepsilon$ whose distribution is defined by D_0.

Unfortunately, we couldn't find in the literature any provable (α, β)-approximations for the problem in (1). On the positive side, in the recent years several fast approximations were suggested for the projective clustering problem, which is a constrained version of the problem in (1), as we explain below. Using an approximation for projective clustering allows us to have bound on the additive error that is introduced by our coreset; see Corollary 2. In fact, for simplicity, in all our experiments we use the simplest version of projective clustering, which is the mean D_0 of the columns of Y. Still, due to the sampling process and the intuition that natural images tend to contain patches with small variances rather than random noise (a fact that we observed in all our experiments), the approximation error of the corresponding small sample was small. See Section 8 for discussion.

Projective clustering as a special case of dictionary learning. As noted in [6], for the case $T_0 = 1$ and the additional constraint that X is a binary matrix, the problem reduced to the K-means problems: compute a set D of K points that minimizes the sum of squared distances from each point (column) in Y to its nearest point in D.

For the case $T_0 = d$, the problem reduces to the low-rank approximation problem (also known as PCA) where we wish to approximate Y by a K-dimensional subspace which is spanned by the K columns of D.

For the case $T_0 = 1$ without the binary constraint on X, the problem is to compute a set of K lines, each passes through the origin, such that the sum of squared distances from each point in Y to its nearest line is minimizes. In this case the ith column of the matrix D corresponds to an arbitrary point on the ith line, for $i = 1, \cdots, K$. This is a variant of the problem known as K-line mean that is defined similarly, but without the constraint that each line must pass through the origin; see [17].

In a similar way, for the case $T_0 = 2$ the problem reduces to compute a set of $k = \binom{K}{2}$ two dimensional subspaces (i.e, planes that intersect the origin) which minimizes the sum of squared distances to the points of Y, with the constraint that each subspace is spanned by two vectors from a set D of K vector. That is, the set of $k = \binom{K}{2}$ subspaces is defined by the matrix $D \in \mathbb{R}^{d \times k}$.

In general, for $T_0, K \geq 1$, the problem in (1) is to compute a set of $k = \binom{K}{T_0}$ subspaces, each of dimension T_0, which minimizes the sum of squared distances to the points of Y, with the constraint that each subspace is spanned by T_0 vectors from a set D of K vectors.

While we couldn't find the dictionary problem in the literature of computational geometry, the related projective clustering problem has a long line of research, including several breakthroughs in the recent years in the context of coreset; see [38] and [18] for a survey. In this problem, we are given a set Y of n points and two integers $j, k \geq 1$, and wish to compute a set of k subspaces of $\mathbb{R}^d$, each of dimension j, such that the sum of squared distances from each point

in Y to its nearest subspace is minimized. In some papers, instead of subspaces, we wish to compute affine subspaces (i.e, translated subspaces that do not necessarily intersect the origin). Other distance functions and cost functions (such as sum of non-squared or maximum distances) were also considered during the recent years; see [18] for a survey.

Recently a $(1+\varepsilon)$ approximation for projective clustering that can be computed in $O(n)$ time were suggested in [38] for every constant j, k and ε, using the framework of [18]. For the case $k = 1$, the projective clustering problem reduces to the low rank approximation problem, for $j = 1$ it reduces to the k-line mean problem, and for the case $j = 0$ we obtain the k-mean problem. These three cases of projective clustering, yields the problem in (1) for $T_0 = d$, $T_0 = 1$ and $T_0 = 0$ respectively, as explained in the beginning of this section. In the general case, the problem in (1) is a constrained version of projective clustering of n points by $k = \binom{K}{T_0}$ subspaces each of dimension $j = s$. The constraint is that the subspaces will be spanned by a small set D of k vectors.

For projective clustering, computing the projection of each input point on its nearest subspace is trivial in $O(djk)$ time, by iterating through each of the k subspaces. On the general dictionary learning problem, however, there are $\binom{K}{T_0}$ candidates, and it is not clear how to compute the closest distance to such a set of subspaces efficiently. That is, even for a *given* dictionary D and a point $y \in y$ there are no polynomial time algorithm for computing the closest subspace in polynomial time. Nevertheless, a lot of heuristics have been suggested over the years for approximating distance to subspaces (called pursuit algorithms, see a detailed description of these methods in [4]), for approximating points by subspaces in general (see [16] and references therein), and for the k-dictionary problem in particular (see references in [4]).

4 Coresets for dictionaries

Approximation algorithms in computational geometry and machine learning often make use of random sampling [9, 32], feature extraction [13, 8] and ε-samples [26]. Coresets can be viewed as a general concept that includes all of the above and more. The idea behind coresets is to replace the set of original elements over which we want to solve a given problem with a new and much smaller (weighted) set that possesses the same solution to the problem at hand. This, theoretically, requires knowing the solution to the given problem so that we can construct a set with the same solution. It is generally possible though to replace the original problem with a much simpler one that gives enough information to construct the coreset without actually solving the original problem. The solution algorithm can then be applied to this new set. The size of the coreset depends only on the problem parameters and required approximation error and not on the input size. This allows to drastically reduce computational time and space complexity without changing the actual algorithm. See a comprehensive (but not so updated) survey on this topic by Agarwal, Har-Peled and Varadarajan [2]. Note though that it is not clear that there is any commonly agreed-upon

definition of a coreset, despite several inconsistent attempts to do so [24, 3, 10, 23, 12, 19].

In our context, i.e the K-SVD algorithm for a sparse dictionary construction, the input is a $d \times n$ matrix Y who's columns represent n points in $\mathbb{R}^d$ and integers $k, T_0 \geq 1$. We want to construct a $d \times k$ matrix D who's columns represent k points in $\mathbb{R}^d$. The matrix D is called a *dictionary* and we want to be able to represent each point in Y as a sparse sum of points in D. Typically k is much larger than d and n is much larger than k.

To this end, for every column y in Y and a candidate dictionary $D \in \mathbb{R}^{d \times k}$ we define

$$\text{err}(y, D) = \min_{x \in \mathbb{R}^k, \|x\|_0 \leq T_0} \|Dx - y\|_2^2$$

as the distance between y and its closest subspace over the set of $\binom{k}{T_0}$ subspaces that are spanned by at most T_0 columns of D. The sum of squared distances over the columns of Y is the cost of D:

$$\text{cost}(Y, D) = \sum_{y \in Y} \text{err}(y, D) = \min_X \|Y - DX\|^2,$$

where the minimum is over every matrix X whose columns are T_0-sparse.

Let $\mathcal{D}$ be a set of $d \times k$ matrices that represents the set of all possible dictionaries. A coreset scheme `Coreset` for a class of queries $\mathcal{D}$ is an algorithm that gets as input a $d \times n$ matrix Y, and a parameter $\varepsilon > 0$, and outputs a $d \times c$ matrix $C = \texttt{Coreset}(P, \varepsilon)$ such that for every $D \in \mathcal{D}$:

$$(1 - \varepsilon) \cdot \text{cost}(P, D) \leq \text{cost}(C, D) \leq (1 + \varepsilon) \cdot \text{cost}(P, D).$$

The matrix C is called a *coreset*. Typically, one expects c to be much smaller than n.

By the definition of coreset, the dictionary D^* that minimizes the cost $\text{cost}(P, D)$ over the collection $\mathcal{D}$ of dictionaries, with any additional set of constraints, can be approximated by the coreset C.

5 Why Coresets?

Dealing with NP-hard problems. For a lot of "hard" (for example, NP-hard) problems which have no efficient solutions, there exist corresponding small coresets that can be computed very efficiently. The projective clustering problem is NP-hard even for the case $j = 0$ (k-means) where k is part of the input (not a constant). Still, small coresets for k-means (of size independent of both n and d) can be constructed in linear time in all the parameters [18]. Practical heuristics are then applied on the coreset instead of the original set of points in order to get smaller running time [1, 22]. More generally, variants of the projective clustering problems that are NP-hard still have coresets that can be constructed in polynomial time.

Running heuristics on the coresets instead of the original input allows us to obtain faster running times and deal with larger dataset. The connections in Section 3 between projective clustering and dictionaries yield the natural question of whether we can also use coresets for learning dictionaries.

Similarly, the optimization problem (1) that K-SVD is designed to solve is NP-hard for most values of T_0 and K. No tractable algorithm is known that provably fits a good dictionary, even under idealized conditions similar to those in compressed sensing. Still, as in the related projective clustering problem, we may be able to compute coresets and provide bounds on their approximation error, as presented in this paper. The main reason is that, as in projective clustering, getting rough approximation for the optimal solution is much easier than actually solving the problem. However, for reducing the size of the data, rather then solve the optimization problem, such a rough approximation might suffice.

In fact, our experimental results show that sometimes we actually get improved approximation quality while running on the coreset compared to the original input. This is possible since K-SVD is a heuristic (not an optimal solution) that might get trapped in "bad" local minima. It might be that in the compression process a noisy data that cause such local minima is removed from the input.

Constrained Optimization. A coreset for a set $\mathcal{D}$ of dictionaries is, by definition, a coreset for a subset of $\mathcal{D}$. Therefore, a coreset for a problem can be used for solving the problems under non-trivial or field-specific constrainted solutions by applying (not necessarily efficient) algorithms that deal with these constraints on the coreset.

For example, constrained non-negative variant version of the K-SVD was suggested by Aharon, Elad and Bruckstein [6], where the atoms must be positive. Other versions of K-SVD have additional constraint on the L_1 norm of X (that can be handled via LASSO [36])). Our coreset is for K-SVD is suitable for these versions as well.

Streaming. There is a general reduction called "merge-and-reduce, that shows that a small coreset scheme to a given problem suffices to solve the corresponding problem on a streaming input [7, 25]. In the streaming model the n input bits of data arrived one by one, and we allowed to use memory (space) that is only sub-linear in n, usually $O(\log n)$ bits. For example, a movie that consists of Gigabytes of data that is broadcast via the Internet into a mobile phone that can store only few mega bytes.

The key idea is to construct and save in memory a coreset for every block of streaming bits. When we have two coresets in memory – we construct a single coreset for the pair of coresets. This recursive process yields a binary tree of height $O(\log n)$, where we need to store in memory a single coreset for each level on the tree. See Fig. 1.

Parallel/Distributed computations. Using the same ideas from the streaming model, a (non-parallel) coreset construction can be transformed into a parallel

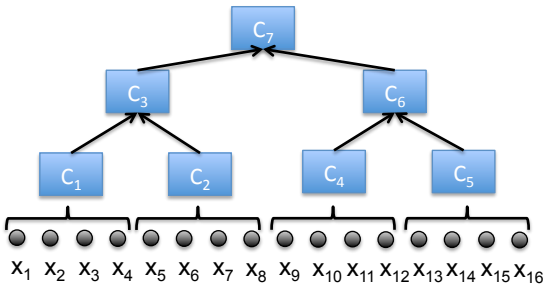


Fig. 1. Tree construction for generating coresets in parallel or from data streams. Black arrows indicate "merge-and-compress" operations. The (intermediate) coresets $C_1, \ldots, C_7$ are enumerated in the order in which they would be generated in the streaming case. In the parallel case, C_1, C_2, C_4 and C_5 would be constructed in parallel, followed by parallel construction of C_3 and C_6, finally resulting in C_7.

one; See Fig. 1. We partition the data into sets, and compute coreset for each set, independetly, on different computers. We then compute (in parallel) a coreset for every pair of such coresets. Continue in this manner, yields a process that takes $O(\log n)$ iteration of parallel computation.

Graphical Processing Units (GPUs) . More than 90% of new desktop and notebook computers have integrated GPUs, which is a specialized microprocessor that accelerates graphics rendering from the CPU. Because most of these computations involve matrix and vector operations, engineers and scientists have increasingly studied the use of GPUs for non-graphical calculations. However, in order to be efficient, algorithm that uses GPU must use mainly independent and very simple computations that can be done in parallel.

The above connection between coresets and parallel computations make them a natural tool for GPU computing. Since algorithms for computing coresets for a problem are usually much simpler than algorithms for solving the corresponding problem (as is the case of this paper), it may be useful to compute coresets via GPUs and then solve the problem on the small coreset using the CPU. For example, the recent version of Matlab (2010b) does not support SVD computation using the GPU but do support GPU functions that suffice for computing coresets for the SVD problem (say, using the algorithms in [18, 11]).

Privacy. Intuitively, coreset implies that a small dataset can be exposed instead of the larger original dataset, while preserving its information with respect to some family of queries. Coresets with formal proofs that guarantee that no information is leaking from them about individuals were introduced in [15].

6 Coreset for a single k-dictionary query

Algorithm 1 `Coreset`(Y, D_0, c)

Input: a $d \times n$ matrix Y, an integer $c \geq 1$, and a matrix D_0 (of arbitrary size).
Output: a weighted $c \times d$ matrix C that satisfies Theorem 1.

Pick a non-uniform random sample $S = \{s_1, s_2, \cdots, s_c\}$ of c i.i.d. columns from Y, where $y \in Y$ is chosen with probability proportional to $\text{err}(y, D_0)$. That is, for every $s \in S$ and $y \in Y$, the probability that $s = y$ is

$$\text{pr}(y) = \frac{\text{err}(y, D_0)}{\sum_{y \in Y} \text{err}(y, D_0)} = \frac{\text{err}(y, D_0)}{\text{cost}(Y, D_0)}.$$

Return the weighted matrix C whose columns are the vectors of S (in some arbitrary order), where each $y \in C$ is weighted (multiplied) by

$$w(y) = \frac{1}{c \cdot \text{pr}(y)}. \tag{2}$$

The following lemma can be easily proved using Chernoff-Hoeffding's inequality.

Lemma 1. *Let Y be a $d \times n$ matrix, and $\delta, \varepsilon > 0$ be. Let C be the output of the algorithm* `Coreset` *with input parameters Y, D_0 and*

$$c \geq \frac{10 \ln(1/\delta)}{\varepsilon^2}.$$

Let D be a fixed $d \times k$ dictionary. Then, with probability at least $1 - \delta$,

$$|cost(Y, D) - cost(C, D)| \leq \varepsilon cost(Y, D_0) \cdot \max_{y \in Y} \frac{\text{err}(y, D)}{\text{err}(y, D_0)}.$$

Proof. We first prove that $E[\text{cost}(C, D)] = \text{cost}(Y, D)$. Let s_i be the ith vector of S, as defined in the algorithm `Coreset`, for $1 \leq i \leq c$, and put $y \in C$. Since the c vectors of S are sampled independently, we have

$$E[\text{cost}(C, D)] = E\left[\sum_{y' \in C} w(y')\text{err}(y', D)\right] = c \cdot E\left[w(y)\text{err}(y, D)\right].$$

By (2), we have

$$\begin{aligned} E[w(y)\text{err}(y, D)] &= \sum_{y \in Y} \text{pr}(y) \cdot w(y)\text{err}(y, D) \\ &= \sum_{y \in Y} \text{pr}(y) \cdot \frac{1}{c \cdot \text{pr}(y)} \cdot \text{err}(y, D) \\ &= \sum_{y \in Y} \frac{\text{err}(y, D)}{c} = \frac{\text{cost}(Y, D)}{c}. \end{aligned}$$

Combining the last two equations yields

$$E[\text{cost}(C,D)] = \text{cost}(Y,D). \tag{3}$$

By the Chernoff-Hoeffding's inequality, for independent random variables $x_1,\ldots,x_c$ we have, with probability at least $1-\delta$,

$$\left| E\left[\sum_{i=1}^{c} x_i\right] - \sum_{i=1}^{c} x_i \right| \leq c\varepsilon \max_{1\leq i\leq c} x_i.$$

By substituting $x_i = w(y_i)\text{err}(y_i,D)$ where y_i is the ith column of C, we get that, with probability at least $1-\delta$,

$$\begin{aligned}|E[\text{cost}(C,D)] - \text{cost}(C,D)| &= \left| E\left[\sum_{y\in C} w(y)\text{err}(y,D)\right] - \sum_{y\in C} w(y)\text{err}(y,D)\right| \\ &\leq c\varepsilon \max_{y\in Y} w(y)\text{err}(y,D) \\ &= \varepsilon \text{cost}(Y,D_0)\max_{y\in Y}\frac{\text{err}(y,D)}{\text{err}(y,D_0)}.\end{aligned}$$

Assume that the last inequality indeed holds (which happens with probability at least $1-\delta$). Together with (3), we get

$$|\text{cost}(Y,D) - \text{cost}(C,D)| \leq \varepsilon \text{cost}(Y,D_0)\max_{y\in Y}\frac{\text{err}(y,D)}{\text{err}(y,D_0)}.$$

Corollary 1. *Let $\alpha \geq 1$ be an integer and D_0 be a dictionary, such that for every $D \in \mathcal{D}$:*

(i) $cost(Y,D_0) \leq \alpha \cdot cost(Y,D)$.
(ii) $\text{err}(y,D) \leq \text{err}(y,D_0)$ *for every* $y \in Y$.

Put $\varepsilon,\delta > 0$. Let C be the weighted matrix that is returned by the algorithm `Coreset` *with input parameters $c \geq 10\alpha^2\ln(1/\delta)/\varepsilon^2$ and D_0. Then, for a fixed dictionary $D \in \mathcal{D}$ (which is independent of C), we have*

$$(1-\varepsilon)cost(Y,D) \leq cost(C,D) \leq (1+\varepsilon)cost(Y,D),$$

with probability at least $1-\delta$.

Proof. We have $\text{cost}(Y,D_0) \leq \alpha \cdot \text{cost}(Y,D)$ by property (i). Replacing ε with ε/α in Lemma 1 yields

$$\begin{aligned}|\text{cost}(Y,D) - \text{cost}(C,D)| &\leq (\varepsilon/b)\text{cost}(Y,D_0)\max_{y\in Y}\frac{\text{err}(y,D)}{\text{err}(y,D_0)} \\ &\leq \varepsilon \text{cost}(Y,D)\max_{y\in Y}\frac{\text{err}(y,D)}{\text{err}(y,D_0)} \leq \varepsilon \text{cost}(Y,D),\end{aligned}$$

where the last inequality follows from property (ii).

7 Coreset for all k-dictionary queries

In order to have an ε-coreset for a set $\mathcal{D}$ of more than one dictionary, there are still two problems that remain to be solved. Firstly, we need to compute D_0 that satisfies Properties (i) and (ii) of Corollary 1 with sufficiently small b. This will be handle in the next section for our specific applications. Secondly, the definition of ε-coreset demands that C will approximate $\text{cost}(C, D)$ *simultaneously* for *every* $D \in \mathcal{D}$. However, Corollary 1 holds, with probability at least $1 - \delta$, only for a fixed dictionary $D \in \mathcal{D}$, i.e, a single query. If the size of $\mathcal{D}$ is finite, we can replace δ with $\delta/|\mathcal{D}|$ in Corllary 1 and use the union bound to obtain an ε-coreset for Y of size

$$c = O\left(\frac{\ln(|\mathcal{D}|)\alpha^2 \ln(1/\delta)}{\varepsilon^2}\right). \tag{4}$$

However, in the applications of this paper the size of $\mathcal{D}$ is infinite. In this case, we use the result of [30] that is based on PAC-learning theory. Roughly speaking, the result states that to obtain an ε-coreset, it suffices to replace the term $\ln(|\mathcal{D}|)$ in (4) by some dimension v that represents the complexity of the set $\mathcal{D}$, in a VC-dimension type notion. Usually v is proportional to the number of parameters that are needed to represent a dictionary D of $\mathcal{D}$, which is, in the general case, the number dk of entries in the matrix D. We give the formal definition in the proof of the following main theorem.

Theorem 1. *Let Y be a $d \times n$ matrix, $\varepsilon, \delta > 0$. Let D_0 be a matrix as defined in Corollary 1 for some value of $b > 0$. Let C be the weighted matrix that is returned by the algorithm* `Coreset` *with input parameters $c \geq 10dkb^2 \ln(1/\delta)/\varepsilon^2$ and D_0. Then, with probability at least $1 - \delta$, C is an ε-coreset of Y.*

Proof. We denote the ith column of Y by y_i. Let $F = \{\text{err}(y_i, \cdot) \mid 1 \leq i \leq n\}$ be the set of functions for the atoms to a given dictionary. That is, a function $f_i : \mathbb{R}^{d \times k} \to [0, \infty)$ in F maps every $d \times k$ matrix $D \in \mathcal{D}$ to $\text{err}(y_i, D)$. The *sensitivity* (or importance) of a function f in a set F is defined in [18] as

$$s(f) := \max_{D \in \mathcal{D}} \frac{f(D)}{\sum_{f \in F} f(D)}$$

For our definition of F, we thus obtain for every $i = 1, \cdots, n$,

$$s(f_i) = \max_{D \in \mathbb{R}^{d \times k}} \frac{\text{err}(y_i, D)}{\text{cost}(Y, D)}. \tag{5}$$

Intuitively, if there is a dictionary D such that $\text{cost}(Y, D)$ is dominated by y_i then $s(f_i)$ is close to 1 and we should choose y_i to the coreset. Otherwise, if the contribution of y_i is neglected for every possible query dictionary D, then $s(f_i)$ is close to zero and we can ignore y_i. Our hope is that not all the points are very important, otherwise the coreset will have to be large. To this end, we need to prove that the *total sensitivity*, which is defined as

$$T := \sum_{i=1}^{n} s(i)$$

is small.

Indeed, using the definition of D_0 in Corollary 1, we have $\text{cost}(Y, D) \geq \text{cost}(Y, D_0)/\alpha$ and $\text{err}(y_i, D) \leq \text{err}(y_i, D_0)$. Plugging this in (5) yields

$$s(f_i) \leq \frac{\alpha \text{err}(y_i, D_0)}{\text{cost}(Y, D_0)}.$$

Hence,

$$T \leq \sum_{i=1}^{n} \frac{\alpha \text{err}(y_i, D_0)}{\text{cost}(Y, D_0)} = \alpha.$$

The size of a coreset depends on the total sensitivity T, and the pseudo dimension $\dim(F)$ of F, which represents its complexity. We first define the dimension of a set of subsets, and then of a set of functions.

Definition 1 (range space [30, 18]). *A* range space *is a pair* $(F, \textbf{ranges})$ *where* F *is a set, and* **ranges** *is a set of subsets of* F*. The* dimension *of the range space* $(F, \textbf{ranges})$ *is the smallest integer* d*, such that for every* $G \subseteq F$ *we have*

$$\left| \{G \cap \textbf{range} \mid \textbf{range} \in \textbf{ranges}\} \right| \leq |G|^d \ .$$

The dimension of a range space relates (but is not equivalent) to a term known as the VC-dimension of a range space.

Definition 2 (pseudo dimension [31]). *Let* F *be a finite set of* functions *from a set* $\mathcal{D}$ *to* $[0, \infty)$*. The* pseudo dimension $\dim(F)$ *of* F *is the dimension of the range space* $\big(F, \textbf{ranges}(F)\big)$*, where* $\textbf{ranges}(F)$ *is defined as follows. For every* $D \in \mathcal{D}$ *and* $r \geq 0$*, let* $\textbf{range}(F, D, r) = \{f \in F \mid f(D) \leq r\}$*. Let* $\textbf{ranges}(F) = \{\textbf{range}(F, D, r) \mid D \in \mathcal{D}, r \geq 0\}$.

One of the main result of [18] is that a non-uniform random sample of $10 \dim(F) \cdot T^2 \log(1/\delta)/\varepsilon^2$ from F according to the sensitivity of the functions in F yields an ε-coreset for F.

Theorem 2 ([18]). *Let* F *be a set of non-negative real functions from* $\mathcal{D}$*, with total sensitivity* T*. Let* S *be a random sample of* $|S| = 10T^2 \dim(F) \ln(1/\delta)/\varepsilon^2$ *function from* F *where* $q = f$ *with probability proportional to its sensitivity* $s(f)$ *for every* $f \in F$ *and* $q \in S$*. Then, with probability at least* $1 - \delta$*, for every* $D \in \mathcal{D}$ *we have that*

$$\left| \sum_{f \in F} f(D) - \sum_{f \in S} \frac{1}{s(f)|S|} \cdot f(D) \right| \leq \varepsilon \sum_{f \in F} f(D).$$

Using the above bound on T we obtain that the size of the sample is

$$\frac{10 \dim(F) \cdot T^2 \log(1/\delta)}{\varepsilon^2} = \frac{10dk\alpha^2 \log(1/\delta)}{\varepsilon^2}.$$

By the construction of C in Algorithm `Coreset`, this proves Theorem 1.

For constructing D_0 such that Corollary 1 would hold, we add the assumption that every dictionary D has D_0 as a subset of its columns. Hence, $\text{err}(Y, D) \leq \text{err}(Y, D_0)$ for every $y \in Y$. To bound α we need to approximate $\min \text{cost}(Y, D)$ which is hard. Instead, we suggest two leeways: the first is to replace ε with $\varepsilon^2\alpha$ in the previous theorem and define the error with respect to the ratio $\max \text{cost}(Y, D_0)/\text{cost}(y, D)$. Alternatively, we can simply restrict ourselves to dictionaries with large enough cost. We summarize this in the next corollary.

Corollary 2. *Let Y be a $d \times n$ matrix, $\varepsilon, \delta > 0$. Let $D_0 \in \mathbb{R}^d$ denote the mean of the columns of Y. Let C be the weighted matrix that is returned by the algorithm* `Coreset` *with input parameters $c \geq 10dk \ln(1/\delta)/\varepsilon^2$ and D_0. Then, with probability at least $1 - \delta$, for every $D \in \mathbb{R}^{d \times k}$ that contains the column D_0 we have*

$$|\text{cost}(Y, D) - \text{cost}(C, D)| \leq \varepsilon \text{cost}(Y, D_0).$$

In particular, C is an ε-coreset for the set of all such dictionaries D whose cost is $\text{cost}(Y, D) \geq \varepsilon \text{cost}(Y, D_0)$.

Another practical choice of D_0 in the last corollary is run the K-SVD heuristic on the original input data Y only for a single iteration, and choose D_0 to be the returned dictionary.

8 Coresets vs. uniform random sample

A natural and popular approach to reduce the input matrix Y is simply to pick a small uniform random sample of its n columns. Such a sample can be computed in sub-linear time in n, assuming random access to the input, which is the case when the data is a table in the RAM or hard-drive. On the contrary, our coreset is based on non-uniform random sampling and it takes at least one pass over the data to compute the desired distribution. In fact, the Matlab implementation of K-SVD that we use applies such initial random sample for handling large images.

Consider a matrix Y that represents an image of white noise. More precisely, the columns of Y are i.i.d. standard random Gaussian vectors. In this case, the sampling distribution of our coreset will be very similar to uniform distribution. Hence, uniform random sample yields compression of the data with similar quality but much faster than coreset.

On the other extreme side, consider a cartoon image or a drawing, where most of the pixels have the same color. In this case, the uniform random sample will probably contain samples only from the background and therefore will be useless. On the other side, our coreset will never sample a solid patch (with zero variance for the pixel values) and only take representative points from the actual drawing. Similarly, on a natural image that contains clusters of interesting areas, say, an image of small window in a dark room, the uniform random sample is likely to miss small clusters (the window), while the coreset is biased to sample such small regions of interest. We show this phenomena in our experiments in Section 9.

The above intuitive discussion has a simple formal explanation. Chernoff bounds implies that the sum of a "small" random sample (of size quadratic in $1/\varepsilon$) from a set of n numbers between 0 to 1 would yield an approximation to the sum of the n numbers up to additive error of εn, with high probability. Each one of the n numbers corresponds to the error $\|y - Dx\|^2$ from a specific atom y to its approximation by a query dictionary D. By scaling the input, our coreset reduces the error to εM where M is the variance of the n numbers. Therefore, if all the dictionaries admit bad approximations, such as the case for the above noisy image Y, then n and the variance M are similar. For more natural images or when the input is sparse, the variance is close to zero, M is independent of n, and the approximation error of the coreset is much smaller, as proved in Theorem 1.

Our experimental results shows that, as implied by the above discussion, coresets perform better compared to random sampling on natural images. This holds even for our naive choice of the 1-mean as the initial rough approximation for the optimal dictionary. We expect that more involved initial dictionaries such as K-means or other projective clustering of the columns of Y, would yield even better error bounds.

Size of sample. Recently, Vainsencher, Mannor and Bruckstein [37] developed generalization bounds on the quality and required uniform random sample size for learning dictionaries under several types of constraints. The notation and lines of research that are used in [37] seem to be different from our analysis that is based on a general framework for bounding coresets size [28, 18]. However, it seems that there is a strong connection between the underlying math.

In particular, our coreset size depends on total sensitivity and VC-dimension that are combined with Hoeffding inequality [18, 28], which have a strong connection to Lipschitz mapping (see [21, Section 2.1]), covering number (as shown in [35]), and Bernstein Inequality (which is a generalization of Hoeffding inequality), respectively, that are used in [37]. Since the analysis of coresets size is mainly based on a long line of research of PAC-learning regarding the required size of uniform random sample [30], we believe that the results of Vainsencher, Mannor and Bruckstein [37] may be used to improve or generalize the bound of the coresets in this paper or vice versa. The main observation for finding such connections is that non-uniform random sampling from a set can be described as a uniform random sampling for a set that contains multiple copies of each item.

The theoretical bounds on our coreset size and error are given in Theorem 1. The size of the coreset depends on the desired bounds on the probability of failure δ, the approximation error ε, and the size of the desired dictionary. The overall error depends on ε and the quality of the initial dictionary D_0, which can be approximated in $O(nd)$ time for the case of 1-mean.

Of course, the same output coreset corresponds to unbounded number of combination of ε, and δ. The worst case theoretical and general analysis also ignores nice structures that usually appear in practical inputs, and thus often are too pessimistic. Fortunately, in practice, we simply choose the size of the

coreset based on our memory or time constrained. The theory provides us the way (distribution) to sample the points, and the guarantee that the coreset size, in general, must be small and independent of the the input matrix Y or its size.

9 Experimental results

9.1 Hardware

We run the experiments on a standard personal modern Laptop, namely, IBM Lenovo W500 as provided by the manufacturer, without additional hardware. In particular, we use the CPU "Intel Core 2 Duo processor T9600 (2.80 GHz)" with 2GB memory. See manufacturer's website (http://www-307.ibm.com/pc/support/site.wss/document.do?lndocid=MIGR-71785) for exact hardware details.

Software. The operation system that we used is "Windows Vista Business" and the Matlab version is 2010b. For the $\mathrm{K}-\mathrm{SVD}$ and OMP algorithms, we use the implementation of Rubinstien that was generously uploaded on the Internet [34]. This implementation was used as a "black box" without changing a line of code. The time and space improvements are therefore only due to the replacing of the original input matrix Y with its coreset.

9.2 Synthetic data

As in previously reported works [27, 29, 5], we first try to construct coresets of synthetic data. In [5] it was shown how the $\mathrm{K}-\mathrm{SVD}$ algorithm approximated the original dictionary D^* that generated a synthetic data matrix Y. In the following experiments we replace Y by its (usually much smaller) coreset C, and compare the results of applying $\mathrm{K}-\mathrm{SVD}$ on C instead of Y. The construction of C is done using algorithm `Coreset` with D_0 and different values of c, as defined in the coreset construction. The construction of the generative dictionary D^* and the input matrix Y was based on the suggested experiments in [5]. As was suggested in [5] and implemented in the code for the K-SVD iterations, instead of taking the actual columns from Y to the coreset, we subtracted the mean of Y from every chosen column, and also added the mean column to our initial dictionary.

Generating the dictionary D^* and the matrix Y. A random (dictionary) matrix D^* of size $d \times k = 20 \times 50$ was generated with i.i.d. uniformly distributed entries. Each column was normalized to a unit norm. Then, a $20 \times n$ matrix Y was produced for different values of n. Each column y of Y was created using a linear combination D^*x of $\|x\|_0 = j = 3$ random and independent different columns of D^*, with uniformly distributed i.i.d. coefficients. White Gaussian noise with varying signal-to-noise ratio (SNR) $\sigma = 20$ was added to the resulting vector D^*x. That is, $Y = D^*X + N$ where N is a matrix that represents the

Gaussian noise in each entry, and every column x of X corresponds to a column vector y in Y as defined above.

We run the experiment with 11 different assignments for n, that were approximately doubled in every experiment: from $n = 585$ to $n = 500,000$. For every such value of n, 50 trials were conducted, when in every trial new dictionary D^* and matrices Y and X were constructed.

Applying $\mathrm{K-SVD}$ *on* Y. We run the $\mathrm{K-SVD}$ implementation of [34], where the maximum number of iterations was set to 40. The rest of parameters were the defaults of the implementation in [34]. We denote the output dictionary by D_Y.

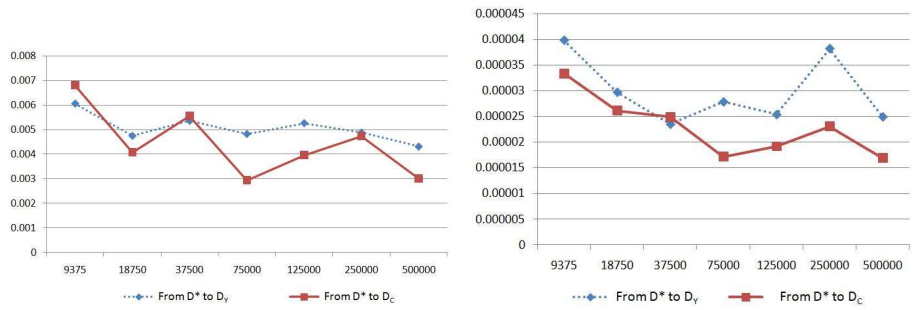
Generating the coreset C. We implemented and run the algorithm $\texttt{Coreset}(c, D_0)$ on the input matrix Y where the size of the coreset was set to $c = 5000$. The parameter D_0 was always set to be the column vector of d ones. This vector yields nearly the same results as taking the mean but could be computed without passing over the input matrix Y.

Applying $\mathrm{K-SVD}$ *on* C. We called to the $\mathrm{K-SVD}$ algorithm using the same parameters as the above call for Y, except for the maximum number of iterations. After setting the number of iterations to 40 for the input C (as in the runs on Y), we got results that are only slightly worse than on Y, but significantly faster (up to 100 times). We therefore decided to sacrifice time in order to get better results, and used 120 iterations on the $\mathrm{K-SVD}$ with the input C. We denote the output dictionary by D_C.

Approximating the sparse coefficients matrix. In order to approximate the entries of the matrix X, we used the OMP heuristic as defined in [33] and implemented in [34]. The objective of OMP is to minimize $\|Y - D_Y X_Y\|_F$ for the given dictionary D_Y and the input matrix Y, over every matrix X_Y whose columns are sparse ($\|x\|_0 = j = 3$ for every column $x \in X_Y$). This is done by minimizing $\|y - D_Y x\|_F$ for every column $y \in Y$ (one by one) over the set of j-sparse vectors x. Similarly, we computed X_C that suppose to minimize $\|Y - D_C X_C\|$ using the OMP heuristic, as done for Y and D_Y.

Measurement. To measure how close D_Y is to D^*, compared to the difference between D_C and D^*, we used the same error measurement $\mathrm{Distance}(D, D^*)$ that was used in the original $\mathrm{K-SVD}$ paper [5], and implemented in [34].

The computation of $\mathrm{Distance}(D, D^*)$ for two dictionaries D and D^* is done by sweeping through the columns of D^* and finding the closest column (in distance) in the computed dictionary D, measuring the distance via $1-|d_i^T \tilde{d}_i|$, where d_i is a column in D^* and $\tilde{d}_i$ is its corresponding element in the recovered dictionary D. The average distance is denoted by $\mathrm{Distance}(D, D^*)$. That is, $\mathrm{Distance}(D, D^*)$ is the sum of distances over every i, $1 \leq i \leq k$, divided by k.



(a) (b)

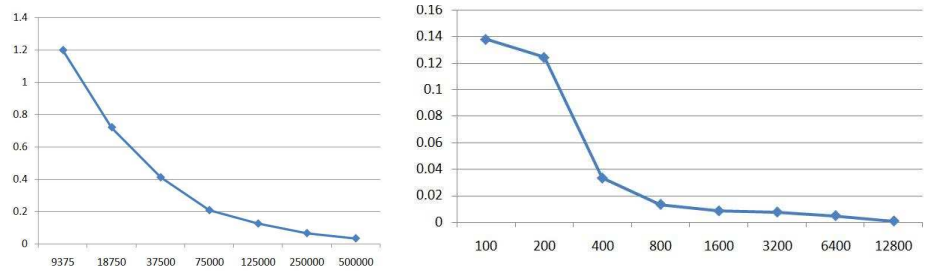
Fig. 2. Comparison of the differences between the dictionaries D_Y D_C and D^* over the number n of rows in the matrix Y. The dictionaries D_Y, D_C are respectively the dictionaries that were constructed using the original matrix Y, and its coreset C. The original generator dictionary of Y is denoted by D^*.

The Results. In Fig. 2(a) we compare the difference (the y-axis) between the dictionaries (the two lines) for different values of n (the x-axis). For example, the dotted line show the average value, for every assignment of n, of Distance(D_C, D^*) over the 50 trials , between the generation dictionary D^* and the dictionary that returned when running $\mathrm{K-SVD}$ with the input matrix Y. The variance over the sets of 50 experiments that corresponds to the average in Fig 2(a) is shown in Fig. 2(b).

The comparison between the running times appears in Fig 3. The x-axis shows the values of n as in Fig. 2, while the y-axis is the ratio between the running time of constructing D_Y, the dictionary of Y, and the running time of constructing D_C, the dictionary of C. The construction time for D_C is the sum of the time it took to construct the coreset C from Y, and the time for constructing D_C from C.

Discussion. In Fig. 2(a) we see that the coreset is usually good at least as the original set for reconstructing the generating dictionary D^*. By Theorem 1, the quality of the coreset C depends on its size c, but not of n. Indeed, the error in Fig. 2 seems to be independent of the value of n. In Fig. 2(b) we see that the results are also more stable on the coreset runs.

Since the size of the coreset is the same ($c = 5000$), the value of n is getting larger, and the running time of the $\mathrm{K-SVD}$ algorithm is linear in the rows of the input matrix (c or n), it is not surprising that the ratio between running times grows linearly with the value of n; see Fig. 3(a). For $n = 500K$ in Fig 3(a), the ratio between the running time is approximately 1:30 (0.032). For $n = 1M$



(a) (b)

Fig. 3. (a) The ratio between the running time of the K − SVD algorithm on the input matrix Y and its corresponding coreset C. (b) The error between the dictionary D_C for coresets C of different sizes.

this ratio is approximately 1:60. However due to time and memory constraints we didn't repeat the experiment for $n = 1M$ 50 times.

The role of the sample size c. By Theorem 1, the size c of the coreset C is polynomial in $1/\varepsilon$, where ε represents the desired quality of the coreset. In Fig. 3(b) we show results for additional set of experiments for a constant $n = 500K$ and different values of the coreset size c. The number of iterations is still 120, and the rest of the parameters remain the same as in the previous experiments. The y-axis is the log of the distance between the dictionaries (base 10) over 50 trials. Indeed, it seems that the error is reduced roughly linearly with the size of c.

(a) (b)

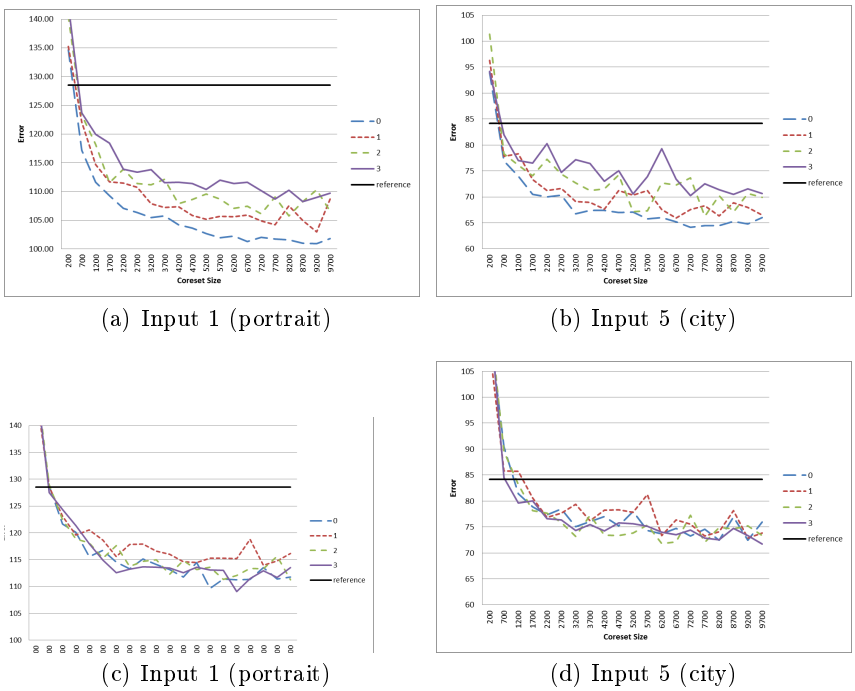
Fig. 4. (a) Noisy Image with SNR= 50. The resulting $PSNR$ is $14.15dB$. **(b)** Denoised image using [14] on the small coreset. The resulting $PSNR$ is ~ 30.9.

9.3 Coresets for High-Definition Images

In [14] it is explained how to apply image denoising using the algorihtm K−SVD. Fortunately, source code was also provided by Rubinstein [34]. We downloaded high-definition images from the movie "Inception"' that was recently released by Warner Bros; see web page "http://collider.com/new-inception-images-posters-christopher-nolan/34058/". We used only one of the images, whose size is $4752 \times 3168 = 15,054,336$ pixels; see Fig. 4. We added a Gaussian noise of $SNR = 50$ which yields a noisy image of $PSNR = 14.15$. Then, we partition the noisy image into 8×8 blocks as explained in [14], and convert the blocks into a matrix Y of approximately $n = 12M$ vectors of dimension $d = 8 \times 8 = 64$. We then hoped to apply the K−SVD as explained in [14] using the default parameters in [34]. However, we got "out of memory" error from Matlab already in the construction of Y. So, instead, we constructed a coreset C of Y in the streaming model using one pass over Y. In this model, coresets are constructed (using our algorithm `Coreset`) from subsets of columns of Y that are loaded one by one and deleted from memory. When there are too many coresets in memory, a coreset for the union of coresets is constructed and the original coresets are deleted. See details in [20]. After constructing such a coreset C of size $c = 10000$ for *all* the columns of Y, we apply the K−SVD on the coreset using sparsity $j = 10$, and $k = 256$ atoms, and 40 iterations. The $PSNR$ was increased, on average of 10

experiments, from 14.15 to 30.9, with variance of ~ 0.002, while the average time for constructing the dictionary was 69 seconds with variance of ~ 7.2

9.4 Comparison of initial dictionaries D_0



(a) Input 1 (portrait) (b) Input 5 (city)

(c) Input 1 (portrait) (d) Input 5 (city)

Fig. 5. Dictionary quality as a function of the number of vectors used to sample the coreset. The series labeled "0" is the constant vector. The other labels represent the number of significant singular vectors that were added to the dictionary. The black line is the reference quality for running on the entire set. Fig. (a) shows the results for the image appearing in Fig. 6(a), Fig. (b) shows the results for the image appearing in Fig. 6(m). Fig. (c) and (d) show a comparison to results when not inserting D_0 into the final dictionary.

We ran the following test on 5 different input images, ranging in size from 8 MP to 60MP, both color and gray scale. The images were broken down into patches of size 16×16, which gave a vector dimension of 256 for gray scale images and 756 for color images. The dictionary size was set to 100 vectors, and the sparsity was set to 5.

We start by looking at the effect of sampling the coreset based on the constant vector, the most significant singular vector, or several of the most significant singular vectors. As can be seen in Fig. 5, using the constant vector or the most

significant singular vector gives nearly the same results. In fact, these two vectors look nearly the same, where the deviation from constant for the most significant singular vector was on a scale of three orders of magnitude smaller than the average (DC).

As mentioned earlier, we can see here, as well as in the next results (Fig. 6), that learning the dictionary on the coreset, also consistently gives better results that learning it on the full set. Theoretically this does not suppose to happen for an ideal algorithm, but the KSVD algorithm is a greedy algorithm, so the assumption is that the coreset smooths the data, reducing sensitivity to local minima.

9.5 Comparison to uniform random sample

Fig. 6 shows a comparison of learning the dictionary on the coreset vs. learning the dictionary on a uniformly sampled random subset of the same size. These test clearly show that coresets are superior to random sampling, both in total run-time vs dictionary quality (coreset construction time + KSVD time) and sample size vs dictionary quality. More interestingly, we see that as expected, random sampling gives a dictionary that is worse than the dictionary learned on the whole dataset, while unexpectedly, coresets consistently give a dictionary that is better than the dictionary learned on the whole dataset. The fact that corestes are better than random sampling is easy to realize for cartoon images as in Fig. 6(j), as random sampling is going to mostly select constant vectors, while coresets will only select detail patches, but is also clearly seen for other types of images.

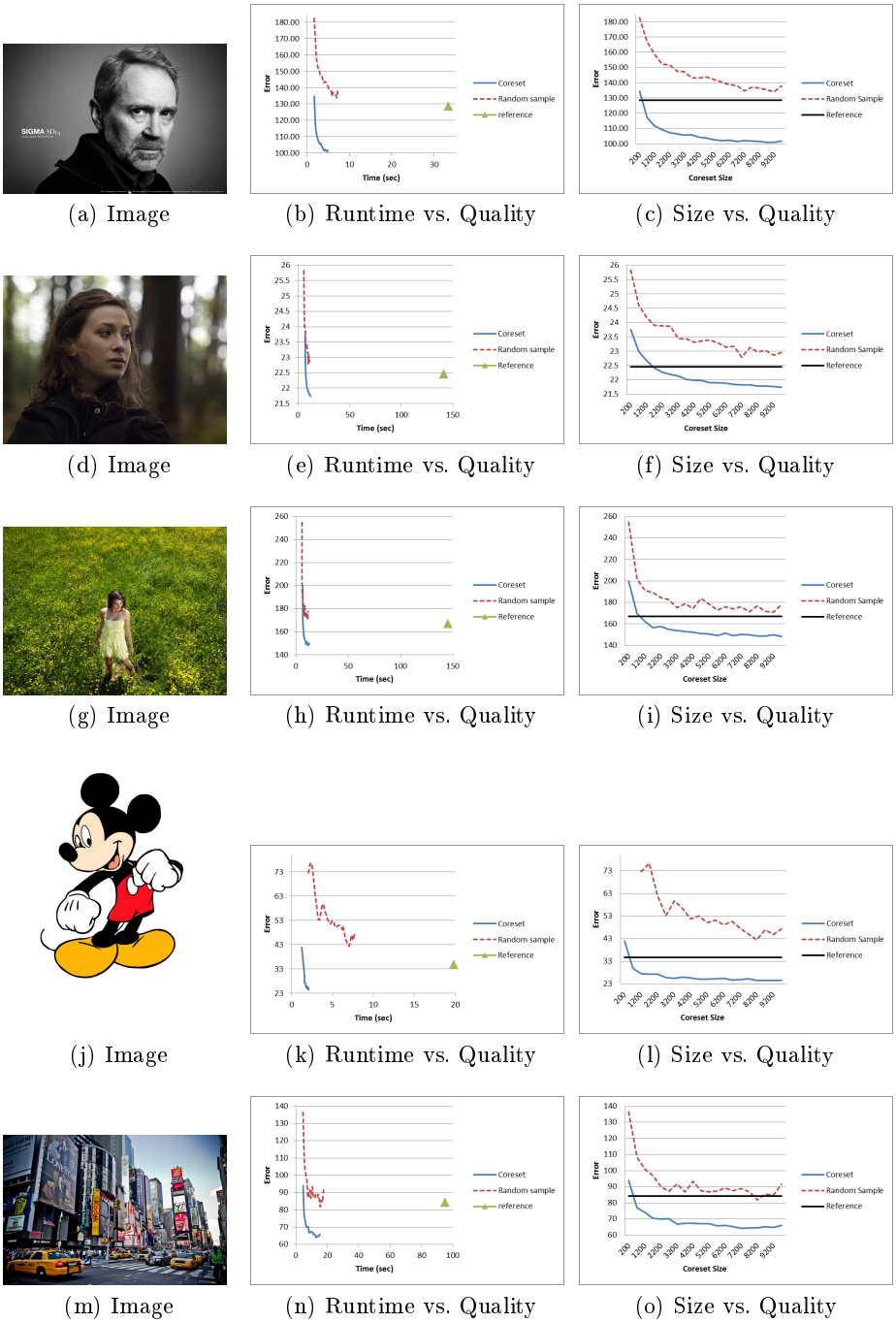
Another result that is less expected is that learning the dictionary on the coreset was more stable and faster than learning the dictionary on the random sample of the same size, offsetting for the longer time taken to construct the coreset.

Overall, we see accelerations ranging from $\times 15$ to over $\times 25$, and an improvement in dictionary quality (reconstruction error) that in most cases ranges from 10% to 40%.

10 Conclusions

The use of coresets opens up new possibilities for dictionary methods. Coreset use allows implementing dictionary methods on anything from HD images up to full length movies. In fact, we've been able to process and entire film using entire frames as feature vectors.

All this opens up the research to perform things such as scene analysis and optimal I-Frame selection or even replacing the entire I-Frame approach with dictionary based reference frames.



(a) Image (b) Runtime vs. Quality (c) Size vs. Quality

(d) Image (e) Runtime vs. Quality (f) Size vs. Quality

(g) Image (h) Runtime vs. Quality (i) Size vs. Quality

(j) Image (k) Runtime vs. Quality (l) Size vs. Quality

(m) Image (n) Runtime vs. Quality (o) Size vs. Quality

Fig. 6. Comparison of reconstruction error vs total run-time (second column) and sample size (third column) for learning a dictionary using a coreset vs. learning the dictionary using a random sample. The coreset was learned using the constant vector. All graphs also show the results for learning the dictionary on the full set (reference). First column is the image used for that test.

References

1. Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *J. Exp. Algorithmics*, 17(1):2.4:2.1–2.4:2.30, May 2012.
2. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.
3. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry - MSRI Publications*, 52:1–30, 2005.
4. M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.
5. M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.
6. M. Aharon, M. Elad, and A.M. Bruckstein. K-svd and its non-negative variant for dictionary design. In *Proceedings of the SPIE conference wavelets*, volume 5914, page 591411, 2005.
7. J.L. Bentley and J.B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation* 1. *Journal of Algorithms*, 1(4):301–358, 1980.
8. R. M. Cesar and L. F. Costa. *Shape Analysis and Classification*. CRC Press, Boca Raton, 2001.
9. M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *12th Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 642–651, 2001.
10. K. L. Clarkson. Subgradient and sampling algorithms for l_1-regression. In *16th Annu. ACM-SIAM Symp. on Discrete algorithms (SODA)*, pages 257–266, 2005.
11. A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. Sampling algorithms and coresets for ℓ_p-regression. In *Proc. 19th Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 932–941, 2008.
12. A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. *10th Int. Workshop on Randomization and Computation (RANDOM)*, pages 292–303, 2006.
13. I. L. Dryden and K. V. Mardia. *Statistical Shape Analysis*. John Wiley and Sons, San Diego, 1998.
14. M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. Image Processing*, 15(12):3736–3745, December 2006.
15. D. Feldman, A. Fiat, H. Kaplan, and K. Nissim. Private coresets. In *Proc. 41st Annu. ACM Symp. on Theory of Computing (STOC)*, pages 361–370, 2009.
16. D. Feldman, A. Fiat, D. Segev, and M. Sharir. Bi-criteria linear-time approximations for generalized k-mean/median/center. In *23rd ACM Symp. on Computational Geometry (SOCG)*, pages 19–26, 2007.
17. D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proc. 47th IEEE Annu. Symp. on Foundations of Computer Science (FOCS)*, pages 315–324, 2006.
18. D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proc. 41th Ann. ACM Symp. on Theory of Computing (STOC), full version in http : //arxiv.org/abs/*1106.1379, 2010.

19. D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *23rd ACM Symp. on Computational Geometry (SoCG)*, pages 11–18, 2007.
20. D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proc. 23rd ACM Symp. on Computational Geometry (SoCG)*, pages 11–18, 2007.
21. D. Feldman and L.J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1343–1354. SIAM, 2012.
22. G. Frahling and C. Sohler. A fast k-means implementation using coresets. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 135–143. ACM, 2006.
23. S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.
24. S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *36th Annu. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.
25. S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Annu. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.
26. D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992.
27. K. Kreutz-Delgado, J.F. Murray, B.D. Rao, K. Engan, T.W. Lee, and T.J. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2):349–396, 2003.
28. M. Langberg and L. J. Schulman. Universal ε approximators for integrals. *proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.
29. S. Lesage, R. Gribonval, F. Bimbot, and L. Benaroya. Learning unions of orthonormal bases with thresholded singular value decomposition. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 5. IEEE, 2005.
30. Y. Li, P. M. Long, and A. Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences (JCSS)*, 62, 2001.
31. Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. In *Symp. on Discrete Algorithms*, pages 309–318, 2000.
32. K. Mulmuley. *Computational Geometry, an Introduction through Randomized Algorithms*. Prentice Hall, 1993.
33. YC Pati, R. Rezaiifar, and PS Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 2002.
34. Ron Rubinstein. Ksvd-box v13. Technical report.
35. J. Shawe-Taylor, P.L. Bartlett, R.C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *Information Theory, IEEE Transactions on*, 44(5):1926–1940, 1998.
36. B. Shen, W. Hu, Y. Zhang, and Y.J. Zhang. Image inpainting via sparse representation. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 697–700. IEEE, 2009.
37. D. Vainsencher, S. Mannor, and A.M. Bruckstein. The sample complexity of dictionary learning. *Arxiv preprint arXiv:1011.5395*, 2010.

38. Kasturi Varadarajan and Xin Xiao. A near-linear algorithm for projective clustering integer points. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1329–1342. SIAM, 2012.