Improved Practical Matrix Sketching with Guarantees

Amey Desai, Mina Ghashami, and Jeff M. Phillips

Abstract—Matrices have become essential data representations for many large-scale problems in data analytics, and hence matrix sketching is a critical task. Although much research has focused on improving the error/size tradeoff under various sketching paradigms, the many forms of error bounds make these approaches hard to compare in theory and in practice. This paper attempts to categorize and compare the most known methods under row-wise streaming updates with provable guarantees, and then to tweak some of these methods to gain practical improvements while retaining guarantees.
For instance, we observe that a simple heuristic iSVD, with no guarantees, tends to outperform all known approaches in terms of size/error trade-off. We modify the best performing method with guarantees, FREQUENTDIRECTIONS, under the size/error trade-off to match the performance of iSVD and retain its guarantees. We also demonstrate some adversarial datasets where iSVD performs quite poorly. In comparing techniques in the time/error trade-off, techniques based on hashing or sampling tend to perform better. In this setting we modify the most studied sampling regime to retain error guarantee but obtain dramatic improvements in the time/error trade-off.
Finally, we provide easy replication of our studies on APT, a new testbed which makes available not only code and datasets, but also a computing platform with fixed environmental settings.

Index Terms—Low-rank matrix approximation, Streaming algroithm, Frequent Directions

1 Introduction

MATRIX sketching has become a central challenge [7], [28], [36], [46], [52] in large-scale data analysis as many large data sets including customer recommendations, image databases, social graphs, document feature vectors can be modeled as a matrix, and sketching is either a necessary first step in data reduction or has direct relationships to core techniques including PCA, LDA, and clustering. There are several variants of this problem, but in general the goal is to process an $n \times d$ matrix A to somehow represent a matrix B so that $\|A - B\|_{2,F}$ or (examining the covariance) $\|A^T A - B^T B\|_2$ is small. In both cases, the best rank-k approximation A_k can be computed using the singular value decomposition (svd); however this takes $O(nd \min(n, d))$ time and $O(nd)$ memory. This is prohibitive for modern applications which usually desire a small space streaming approach, or even an approach that works in parallel. For instance, diverse applications receive data in a potentially unbounded and time-varying stream and want to maintain some sketch B. Examples of these applications include data feeds from sensor networks [13], financial tickers [18], [62], on-line auctions [11], and network traffic [40], [57].

In recent years, extensive works have been done to improve theoretical bounds in the size of B. Random projections [7], [56] and hashing [19], [60] approximate A as a random linear combination of rows and/or columns of A. The column sampling methods [14], [27], [28], [30], [33], [48], [55] choose a set of columns (and/or rows) from A to represent B; the best bounds require multiple passes over the data. We refer readers to recent works [19], [39], [47], [61] for extensive discussion of various models and error bounds. Recently Liberty [46] introduced a new technique called as FREQUENTDIRECTIONS (abbreviated as FD) which is deterministic, achieves a spectral bound on covariance error $\|A^T A - B^T B\|_2$, and moreover greatly outperforms the random projection, hashing, and column sampling techniques in practice. In addition, there is a family of heuristic techniques [16], [41], [42], [45], [54] (which we refer to as iSVD, described relative to FD in Section 2), which are used in many practical settings, but are not known to have any error guarantees. Hence, this problem has seen immense progress in the last decade with a wide variety of algorithmic improvements in a variety of models [7], [19], [20], [23], [27], [30], [30], [33], [39], [46], [53], [56]; which we review thoroughly in Section 2 and assess comprehensively empirically in Section 5.

1.1 Notation and Problem Statement

We denote an $n \times d$ matrix A as a set of n rows as $[a_1; a_2; \ldots; a_n]$ where each a_i is a row of length d. Alternatively a matrix V can be written as a set of columns $[v_1, v_2, \ldots, v_d]$. We assume $d \ll n$. We will consider streaming algorithms where each element

Thanks to support by NSF CCF-1115677, CCF-1350888, IIS-1251019, ACI-1443046, and CNS-1514520.

of the stream is a row a_i of A. Some of the algorithms also work in a more general setting (e.g., allowing deletions or distributed streams). The squared Frobenius norm of a matrix A is defined $\|A\|_F^2 = \sum_{i=1} \|a_i\|^2$ where $\|a_i\|$ is Euclidean norm of row a_i, and it intuitively represents the total size of A. The spectral norm $\|A\|_2 = \max_{x:\|x\|=1} \|Ax\|$, and represents the maximum influence along any unit direction x. It follows that $\|A^TA - B^TB\|_2 = \max_{x:\|x\|=1} \left|\|Ax\|^2 - \|Bx\|^2\right|$.

Given a matrix A and a low-rank matrix X, let $\pi_X(A) = AX^\dagger X$ be a *projection* operation of A onto the rowspace spanned by X; that is if X is of rank r, then it projects to the r-dimensional subspace of points (e.g. rows) in X. Here $X^\dagger$ indicates taking the Moore-Penrose pseudoinverse of X. The singular value decomposition of A, written $\mathsf{svd}(A)$, produces three matrices $[U, S, V]$ so that $A = USV^T$. The matrix U is $n \times n$ and orthogonal. The matrix V is $d \times d$ and orthogonal; its columns $[v_1, v_2, \ldots, v_d]$ are the right singular vectors, describing directions of the most covariance in A^TA. The matrix S is $n \times d$ and is all 0s except for the diagonal entries $\{\sigma_1, \sigma_2, \ldots, \sigma_r\}$, the *singular values*, where $r \leq d$ is the rank of A. Note that $\sigma_j \geq \sigma_{j+1}$, $\|A\|_2 = \sigma_1$, and $\sigma_j = \|Av_j\|$ describes the norm along direction v_j.

1.1.1 Error measures

In this paper we focus on two classes of error measures between input matrix A and its $\ell \times d$ sketch B. The *covariance error* ensures for any unit vector $x \in \mathbb{R}^d$ that $\|Ax\|^2 - \|Bx\|^2$ is small. This can be mapped to the covariance matrix A^TA since $\max_{\|x\|=1} \|Ax\|^2 - \|Bx\|^2 = \|A^TA - B^TB\|_2$. To normalize the *covariance error*, we set

$$\mathsf{cov\text{-}err}(A, B) = \|A^TA - B^TB\|_2 / \|A\|_F^2$$

which is always greater than 0. [1] The *projection error* describes the error in the subspace captured by B without focusing on its scale. Here we compare the best rank k subspace described by B (as B_k) against the same for A (as A_k). We measure the error by comparing the tail (what remains after projecting A onto this subspace) as $\|A - \pi_{B_k}(A)\|_F^2$. Specifically we normalize the *projection error* so that it is comparable across datasets as

$$\mathsf{proj\text{-}err}(A, B) = \|A - \pi_{B_k}(A)\|_F^2 / \|A - A_k\|_F^2.$$

Note the denominator is equivalent to $\|A - \pi_{A_k}(A)\|^2$, so this ensures the projection error is at least 1. We set $k = 10$ as a default value for rank parameter in all the experiments.[2] Although there are several other error measures used in literature, in this paper we only focus on these bounds. The reason is two fold. First, these cover the two main abstract goals often associated with the matrix sketching: subspace approximation (via proj-err) and directional magnitude or covariance approximation (via cov-err). Second, we are able to translate (or re-derive) the worst case guarantees in terms of these bounds for most known classes of algorithms, at least those for which one can prove bounds. This allows us to see all approaches in common terms. Finally, we can actually evaluate these bounds empirically. This is not obviously true for other bounds (e.g. $\max_{x \in \mathbb{R}^d} \|Ax\|/\|Bx\|$).

1.2 Contributions

We survey and categorize the main approaches to matrix sketching in a row-wise update stream. We consider three categories: sampling, projection/hashing, and iterative, and show how all approaches fit simply into one of these three categories. For each family of algorithms, we map error bounds onto the cov-err and the proj-err measures when possible. We also provide an extensive set of experiments to compare these algorithms along size, error (projection and covariance), and runtime on real and synthetic data.

To make this study easily and readily *reproducible*, we implement all experiments on a new extension of Emulab [5] called Adaptable Profile-Driven Testbed or APT [3]. It allows one to check out a virtual machine with the same specs as we run our experiments, load our precise environments and code and datasets, and directly reproduce all experiments. We also consider new variants of these approaches which maintain error guarantees but significantly improving performance. We introduce several new variants of FD; one of them is α-FD that matches or exceeds the performance of a popular heuristic, iSVD. Before this new variant, iSVD is a top performer in space/error trade-off, but has no guarantees, and as we demonstrate on some adversarial datasets, can fail spectacularly. We also show how to efficiently implement and analyze *without-replacement* row sampling methods, and how this can empirically improve upon more traditional (and easier to analyze) with-replacement row sampling.

2 Matrix Sketching Algorithms

In this section, we review the main algorithms for sketching matrices. We divide them into 3 main categories: (1)

1. The normalization term $\|A\|_F^2$ is invariant to the desired rank k and unit vector x. Some methods have bounds on $\|Ax\|^2 - \|Bx\|^2$ that are relative to $\|A - A_k\|^2$ or $\|Ax\|^2$; as these introduce an extra parameter they are harder to measure empirically.

2. There are variations in bounds on this sort of error. Some measure spectral (e.g. $\|\cdot\|_2$) norm. Others provide a weaker error bound on $\|A - [\pi_B(A)]_k\|_F^2$, where the "best rank k approximation" denoted by $[\cdot]_k$, is taken after the projection. This is less useful since then for a very large rank B (might be rank 500) it is not clear which subspace best approximates A until this projection is performed. Additionally, some approaches create a set of (usually three) matrices (e.g., CUR) whose product approximates A_k. This provides a stronger constructive result which we do not consider in this paper.

sampling algorithms, these select a subset of rows from A to use as the sketch B; (2) projection algorithms, these project the n rows of A onto a random ℓ dimensional subspace to construct the sketch B, sometimes they use hashing; (3) iterative algorithms, these maintain B as a low-rank version of A updated as more rows are added.

There exist other forms of matrix approximation, for instance, using sparsification techniques [7], [12], [34], or allowing more general element-wise updates at the expense of larger sketch sizes [19], [20], [56]. We are interested in preserving the right singular vectors and other statistical properties on the rows of A.

2.1 Sampling Matrix Sketching

Sampling algorithms assign a probability p_i to each row a_i and then select ℓ rows from A into B using this probability. In B, each row has its squared norm rescaled to w_i as a function of p_i and $\|a_i\|$. One can achieve additive error bound using importance sampling with $p_i = \|a_i\|^2/\|A\|_F^2$ and $w_i = \|a_i\|^2/(\ell p_i) = \|A\|_F^2/\ell$, as analyzed by Drineas *et al.* [29] and [36]. These algorithms typically advocate sampling ℓ items independently (with replacement) using ℓ distinct reservoir samplers, taking $O(\ell)$ time per element. Another version [33] samples each row independently, and only retains ℓ rows in expectation. We discuss two improvements to this process in Section 3.2. Much of the related literature describes selecting columns instead of rows (called the *column subset selection problem*) [15]. This is just a transpose of the data and has no real difference from what is described here. There are also techniques [33] that select both columns and rows, but are orthogonal to our goals. This family of techniques has the advantage that the resulting sketch is *interpretable* in that each row of B corresponds to a data point in A, not just a linear combination of them.

2.1.1 Leverage Sampling

An insightful adaptation changes the probability p_i using *leverage scores* [32] or *simplex volume* [25], [26]. These techniques take into account more of the structure of the problem than simply the rows norm, and can achieve stronger relative error bounds. But they also require an extra parameter k as part of the algorithm, and for the most part require much more work to generate these modified p_i scores. We use Leverage Sampling [33] as a representative; it samples rows according to leverage scores (described below). Simplex volume calculations [25], [26] were too involved to be practical. There are also recent techniques to improve on the theoretical runtime for leverage sampling [31] by approximating the desired values p_i, but as the exact approaches do not demonstrate consistent tangible error improvements, we do not pursue this complicated theoretical runtime improvement.

To calculate the leverage scores, we first calculate the svd of A (the task we hoped to avoid). Let U_k be the matrix of the top k left singular vectors, and let $U_k(i)$ represent the ith row of that matrix. Then the *leverage score* for row i is $s_i = \|U_k(i)\|^2$, the fraction of squared norm of a_i along subspace U_k. Then set p_i proportional to s_i (e.g. $p_i = s_i/k$, noting that $\sum_i s_i = k$).

2.1.2 Deterministic Leverage Scores

Another option is to deterministically select rows with the highest s_i values instead of at random. It can be implemented with a simple priority queue of size ℓ. This has been applied to using the leverage scores by Papailiopoulos *et al.* [53], which again first requires calculating the svd of A. We refer to this algorithm as Deterministic Leverage Sampling.

2.2 Projection Matrix Sketching

These methods construct the sketch by taking random combination of n data points in A. A survey by Woodruff [61] (especially Section 2.1) gives an excellent account of this area. In the simplest version, each row $a_i \in A$ would map to a row $b_j \in B$ with element $s_{j,i}$ (jth row and ith column) of a projection matrix S, and each $s_{j,i}$ is a Gaussian random variable with 0 mean and $\sqrt{n/\ell}$ standard deviation. That is, $B = SA$, where S is $\ell \times n$. This follows from the celebrated Johnson-Lindenstrauss lemma [43] as first shown by Sarlos [56] and strengthened by Clarkson and Woodruff [19]. Gaussian random variables $s_{j,i}$ can be replaced with (appropriately scaled) $\{-1, 0, +1\}$ or $\{-1, +1\}$ random variables [6]. We call the version with scaled $\{-1, +1\}$ random variables as Random Projection.

2.2.1 Fast JLT

Using a sparse projection matrix S would improve the runtime, but these lose guarantees if the input matrix A is sparse and its non-zero elements do not align with non-zeros of S. This is circumvented by rotating the space with a Hadamard matrix [9], which can be applied more efficiently using FFT tricks, despite being dense. More precisely, we use three matrices: $P \in \mathbb{R}^{\ell \times n}$ that has entries $p_{i,j} = 0$ with probability $1 - q$, and otherwise $p_{i,j} \sim N(0, \ell/q)$ with probability $q = \min\{1, \Theta((\log^2 n)/d)\}$. The matrix $H \in \mathbb{R}^{n \times n}$ is random Hadamard (this requires n to be padded to a power of 2). The diagonal matrix D with entries $\{\pm 1\}$ on diagonal. Then the projection matrix is $S = PHD$, although algorithmically the matrices are applied implicitly. We refer to this algorithm as Fast JLT. Ultimately, the runtime is brought from $O(nd\ell)$ to $O(nd \log d + (d/\varepsilon^2) \log n)$. The second term in the runtime can be improved with more complicated constructions [10], [23] which we do not pursue here; we point the reader here [58] for a discussion of some of these extensions.

TABLE 1
Theoretical Bounds for Sampling Algorithms (O-notations are suppressed). The proj-err bounds are based on a slightly weaker $\|A - \pi_B(A)\|_F^2$ numerator (instead of $\|A - \pi_{B_k}(A)\|_F^2$). (†) See [24]; the Leverage Sampling bound assumes a constant lower bound on leverage scores. ($\star$) Maximum of this and $\{k, (k/\varepsilon)^{1/(1+\eta)}\}$ where leverage scores follow power-law with decay exponent $1+\eta$.

	ℓ	cov-err	ℓ	proj-err	runtime
Norm Sampling	d/ε^2	ε (†) [30]	k/ε^2	$1+\varepsilon \frac{\|A\|_F^2}{\|A-A_k\|_F^2}$ [30]	$\mathsf{nnz}(A)\cdot\ell$
Leverage Sampling	d/ε^2	ε (†)	$(k\log k)/\varepsilon^2$	$1+\varepsilon$ [48]	$\mathsf{svd}(A)+\mathsf{nnz}(A)\cdot\ell$
Deterministic Leverage	ℓ	-	$(k/\eta\varepsilon)^{1/\eta}(\star)$	$1+\varepsilon$ [53]	$\mathsf{svd}(A)+\mathsf{nnz}(A)\cdot\ell\log\ell$

TABLE 2
Theoretical Bounds for Projection Algorithms (via an ℓ_2 subspace embedding; see [24]), where ℓ is the number of rows maintained, and $\rho(A) = \frac{\|A\|_F^2}{\|A\|_2^2} \geq 1$ is the *numeric rank* of A. (O-notations are suppressed).

	ℓ	cov-err	ℓ	proj-err	runtime
Random Projection	d/ε^2 [56]	$\varepsilon/\rho(A)$	d/ε^2 [56]	$1+\varepsilon$	$\mathsf{nnz}(A)\cdot\ell$
Fast JLT	d/ε^2 [56]	$\varepsilon/\rho(A)$	d/ε^2 [56]	$1+\varepsilon$	$nd\log d + (d/\varepsilon^2)\log n$ [9]
Hashing	d^2/ε^2 [20], [51]	$\varepsilon/\rho(A)$	d^2/ε^2 [20], [51]	$1+\varepsilon$	$\mathsf{nnz}(A) + n\mathsf{poly}(d/\varepsilon)$
OSNAP	$d^{1+o(s/\varepsilon)}/\varepsilon^2$ [51]	$\varepsilon/\rho(A)$	$d^{1+o(s/\varepsilon)}/\varepsilon^2$ [51]	$1+\varepsilon$	$\mathsf{nnz}(A)\cdot s + n\mathsf{poly}(d/\varepsilon)$

TABLE 3
Theoretical Bounds for Iterative Algorithms.

	ℓ	cov-err	ℓ	proj-err	runtime
FD	$k+\frac{1}{\varepsilon}$	$\varepsilon\frac{\|A-A_k\|_F^2}{\|A\|_F^2}$ [38]	$\frac{k}{\varepsilon}$	$1+\varepsilon$ [39]	$nd\ell$
ISVD	ℓ	-	ℓ	-	$nd\ell^2$

2.2.2 Sparse Random Projections

Clarkson and Woodruff [20] analyzed a very sparse projection matrix S, conceived of earlier [23], [60]; it has exactly 1 non-zero element per column. To generate S, for each column choose a random value between 1 and ℓ to be the non-zero, and then choose a -1 or $+1$ for that location. Thus each row vector a_i can be processed in time proportional to its number of non-zeros; it is randomly added or subtracted from 1 row of B, as a count sketch [17] on rows instead of counts. We refer to this as Hashing. A slight modification by Nelson and Nguyen [51], called OSNAP, stacks s instances of the projection matrix S on top of each other. If HASHING used ℓ' rows, then OSNAP uses $\ell = s \cdot \ell'$ rows (we use $s = 4$).

2.3 Iterative Matrix Sketching

The main structure of these algorithms is presented in Algorithm 2.1, they maintain a sketch $B_{[i]}$ of $A_{[i]}$, the first i rows of A. The sketch $B_{[i-1]}$ always uses at most $\ell - 1$ rows. On seeing the ith row of A, it is appended to $B_{[i-1]}$ and forms the new matrix $B_{[i]}$, and if needed the sketch is reduced to use at most $\ell - 1$ rows again using some REDUCERANK procedure. Notationally, we use σ_j as the jth singular value in S, and σ'_j as the jth singular value in S'.

2.3.1 Iterative SVD

The simplest variant of this procedure is a heuristic rediscovered several times [16], [41], [42], [45], [54], with a few minor modifications; we refer to it as *iterative SVD* or iSVD. In iSVD, the ReduceRank(S) simply keeps ${\sigma'}_j = \sigma_j$ for $j < \ell$ and sets $\sigma'_\ell = 0$. This has no worst case guarantees (despite several claims).

Algorithm 2.1 (Generic) FD Algorithm

Input: $\ell, \alpha \in (0,1], A \in \mathbb{R}^{n\times d}$
$B_{[0]} \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell\times d}$
for $i \in [n]$ **do**
  Insert a_i into zero valued row of $B_{[i-1]}$; result is $B_{[i]}$
  if ($B_{[i]}$ has no zero valued rows) **then**
    $[U, S, V] \leftarrow \mathsf{svd}(B_{[i]})$
    $C_{[i]} = SV^T$ # Only needed for proof notation
    $S' \leftarrow$ REDUCERANK(S)
    $B_{[i]} \leftarrow S'V^T$
return $B = B_{[n]}$

2.3.2 Frequent Directions

Recently Liberty [46] proposed an algorithm called Frequent Directions (or FD), further analyzed by Ghashami and Phillips [39], and then together jointly with Woodruff [38]. The REDUCERANK step in FD, sets each ${\sigma'}_j = \sqrt{\sigma_j^2 - \delta_i}$ where $\delta_i = \sigma_\ell^2$. Liberty also presented a faster variant, FastFD, that instead sets $\delta_i = \sigma_{\ell/2}^2$ and updates new singular values to ${\sigma'}_j = \max\{0, \sqrt{\sigma_j^2 - \delta_i}\}$, hence ensuring at most half of the rows are all zeros after each such step. This reduces the runtime from $O(nd\ell^2)$ to $O(nd\ell)$ at expense of a sketch sometimes only using half of its rows.

3 New Sketching Algorithms

Here we describe our new variants on FD that perform better in practice and are backed with error guarantees.

In addition, we explain a couple of new matrix sketching techniques that makes subtle but tangible improvements to the other state-of-the-art algorithms mentioned above.

3.1 Variants On Frequent Directions

Since all our proposed algorithms on FREQUENTDIRECTIONS share the same structure, to avoid repeating the proof steps, we abstract out three properties that these algorithms follow and prove that *any* algorithm with these properties satisfy the desired error bounds. This slightly generalizes (allowing for $\alpha \neq 1$) a recent framework [38]. Some proofs are in the supplementary material.

Consider any algorithm that takes an input matrix $A \in \mathbb{R}^{n\times d}$, outputs a matrix $B \in \mathbb{R}^{\ell\times d}$ and follows three properties below for any unit vector $x \in \mathbb{R}^d$ (for some parameter $\alpha \in (0,1]$ and some value $\Delta > 0$):

- **Property 1**: $\|Ax\|^2 - \|Bx\|^2 \geq 0$
- **Property 2**: $\|Ax\|^2 - \|Bx\|^2 \leq \Delta$
- **Property 3**: $\|A\|_F^2 - \|B\|_F^2 \geq \alpha\ell\Delta$

Lemma 3.1. *Any B satisfying the above three properties satisfies $0 \leq \|A^TA - B^TB\|_2 \leq \frac{1}{\alpha\ell-k}\|A-A_k\|_F^2$, and $\|A - \pi_{B_k}(A)\|_F^2 \leq \frac{\alpha\ell}{\alpha\ell-k}\|A-A_k\|_F^2$, where $\pi_{B_k}(\cdot)$ represents the projection operator onto B_k, the top k singular vectors of B.*

Setting $\ell = (k+1/\varepsilon)/\alpha$ achieves $\|A^TA - B^TB\|_2 \leq \varepsilon\|A-A_k\|_F^2$, and setting $\ell = (k+k/\varepsilon)/\alpha$ achieves $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A-A_k\|_F^2$. FD maintains an $\ell \times d$ matrix B (i.e. using $O(\ell d)$ space), and it is shown [39] that there exists a value Δ such that FD satisfies three above-mentioned properties with $\alpha = 1$.

3.1.1 Parameterized FD

Parameterized FD uses Algorithm 3.1 to reduce the rank of the sketch. This method has an extra parameter $\alpha \in [0,1]$ that describes the fraction of singular values which will get affected in the REDUCERANK subroutine. Note that iSVD has $\alpha = 0$ and FD has $\alpha = 1$. The intuition is that the smaller singular values are more likely associated with noise terms and the larger ones with signals, so we should avoid altering the signal terms in the REDUCERANK step.

Algorithm 3.1 REDUCERANK-PFD(S, α)

$\delta_i \leftarrow \sigma_\ell^2$
return
$\mathsf{diag}(\sigma_1, \ldots, \sigma_{\ell(1-\alpha)}, \sqrt{\sigma_{\ell(1-\alpha)+1}^2 - \delta_i}, \ldots, \sqrt{\sigma_\ell^2 - \delta_i})$

Here, we show error bounds asymptotically matching FD for α-FD (for constant $\alpha > 0$), by showing the three Properties hold. We use $\Delta = \sum_{i=1}^n \delta_i$.

Lemma 3.2. *For any unit vector x and any $\alpha \in [0,1]$: $0 \leq \|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 \leq \delta_i$.*

Proof. The right hand side is shown by just expanding $\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2$ as $\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 = \sum_{j=1}^{\ell} \sigma_j^2\langle v_j, x\rangle^2 - \sum_{j=1}^{\ell} \sigma'^2_j\langle v_j, x\rangle^2 = \sum_{j=1}^{\ell}(\sigma_j^2 - \sigma'^2_j)\langle v_j, x\rangle^2 = \delta_i \sum_{j=(1-\alpha)\ell+1}^{\ell}\langle v_j, x\rangle^2 \leq \delta_i$. For the left side $\delta_i \sum_{j=(1-\alpha)\ell+1}^{\ell}\langle v_j, x\rangle^2 \geq 0$. □

Then summing over all steps of the algorithm (using $\|a_ix\|^2 = \|C_{[i]}x\|^2 - \|B_{[i-1]}x\|^2$) it follows (see Lemma 2.3 in [39]) that $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \sum_{i=1}^n \delta_i = \Delta$, proving Property 1 and Property 2 about α-FD for any $\alpha \in [0,1]$.

Lemma 3.3. *For any $\alpha \in (0,1]$, $\|A\|_F^2 - \|B\|_F^2 = \alpha\ell\Delta$, proving Property 3.*

Proof. We expand $\|C_{[i]}\|_F^2 = \sum_{j=1}^{\ell}\sigma_j^2$ as $\|C_{[i]}\|_F^2 = \sum_{j=1}^{(1-\alpha)\ell}\sigma_j^2 + \sum_{j=(1-\alpha)\ell+1}^{\ell}\sigma_j^2 = \sum_{j=1}^{(1-\alpha)\ell}\sigma'^2_j + \sum_{j=(1-\alpha)\ell+1}^{\ell}(\sigma'^2_j + \delta_i) = \|B_{[i]}\|_F^2 + \alpha\ell\delta_i$. By using $\|a_i\|^2 = \|C_{[i]}\|_F^2 - \|B_{[i-1]}\|_F^2 = (\|B_{[i]}\|_F^2 + \alpha\ell\delta_i) - \|B_{[i-1]}\|_F^2$, and summing over i we get $\|A\|_F^2 = \sum_{i=1}^n \|a_i\|^2 = \sum_{i=1}^n \|B_{[i]}\|_F^2 - \|B_{[i-1]}\|_F^2 + \alpha\ell\delta_i = \|B\|_F^2 + \alpha\ell\Delta$. Subtracting $\|B\|_F^2$ from both sides, completes the proof. □

Since α-FD satisfies the three properties, due to Lemma 3.1 it obtains the following results.

Theorem 3.1. *Given an input matrix $A \in \mathbb{R}^{n\times d}$, α-FD with parameter ℓ returns a sketch $B \in \mathbb{R}^{\ell\times d}$ that satisfies for all $k > \alpha\ell$, $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A - A_k\|_F^2/(\alpha\ell - k)$ and projection of A onto B_k, the top k rows of B, satisfies $\|A - \pi_{B_k}(A)\|_F^2 \leq \frac{\alpha\ell}{\alpha\ell-k}\|A-A_k\|_F^2$.*

Setting $\ell = (k+1/\varepsilon)/\alpha$ yields $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \varepsilon\|A-A_k\|_F^2$ and setting $\ell = (k+k/\varepsilon)/\alpha$ yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A-A_k\|_F^2$.

3.1.2 Fast Parameterized FD

Fast Parameterized FD (or Fast α-FD) improves the runtime performance of parameterized FD in the same way Fast FD improves the performance of FD. More specifically, in REDUCERANK we set δ_i as the $(\ell - \ell\alpha/2)$th squared singular value, i.e. $\delta_i = \sigma_t^2$ for $t = \ell - \ell\alpha/2$. Then we update the sketch by only changing the last $\alpha\ell$ singular values: we set $\sigma'^2_j = \max(\sigma_j^2 - \delta_i, 0)$. This sets at least $\alpha\ell/2$ singular values to 0 once every $\alpha\ell/2$ steps. Thus the algorithm takes total time $O(nd + n/(\alpha\ell/2)\cdot d\ell^2) = O(nd\ell/\alpha)$.

It is easy to see that Fast α-FD inherits the same worst case bounds as α-FD on cov-err and proj-err, if we use twice as many rows. That is, setting $\ell = 2(k+1/\varepsilon)/\alpha$ yields $\|A^TA - B^TB\|_2 \leq \varepsilon\|A-A_k\|_F^2$ and setting $\ell = 2(k+k/\varepsilon)/\alpha$ yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A - A_k\|_F^2$. In experiments we consider Fast 0.2-FD.

3.1.3 SpaceSaving Directions

FD is inspired by an algorithm by Misra and Gries (MG) [50] for the streaming frequent items problem. That is, given a stream $S = \langle s_1, s_2, \ldots, s_n\rangle$, where each item

TABLE 4
Theoretical Bounds for New Iterative Algorithms.

	ℓ	cov-err	ℓ	proj-err	runtime
Fast α-FD	$(k+1/\varepsilon)/\alpha$	$\varepsilon\|A-A_k\|_F^2/\|A\|_F^2$	$k/(\varepsilon\alpha)$	$1+\varepsilon$	$nd\ell/\alpha$
SpaceSaving Directions	$k+1/\varepsilon$	$\varepsilon\|A-A_k\|_F^2/\|A\|_F^2$	k/ε	$1+\varepsilon$	$nd\ell$
Compensative FD	$k+1/\varepsilon$	$\varepsilon\|A-A_k\|_F^2/\|A\|_F^2$	k/ε	$1+\varepsilon$	$nd\ell$

$s_i \in [u] = \{1, 2, \ldots, u\}$, and $f_j = |\{s_i \in S \mid s_i = j\}|$ denotes the frequency of item j in the stream S, the MG sketch uses $O(1/\varepsilon)$ space to construct an estimate $\hat{f}_j$ (for *all* $j \in [u]$) so that $0 \leq f_j - \hat{f}_j \leq \varepsilon n$. It keeps $\ell - 1 = 1/\varepsilon$ counters, each labeled by some $j \in [u]$: it increments a counter if the new item matches the associated label or for an empty counter, and it decrements all counters if there is no empty counter and none match the stream element. Intuitively, FD works similarly treating the singular vectors of B as labels and the squared singular values as counters.

Motivated by an empirical study [22] showing that the SpaceSaving algorithm [49] tends to outperform its analog Misra-Gries [50] in practice, we design an algorithm called SPACESAVING DIRECTIONS (abbreviated SSD) to try to extend these ideas to matrix sketching. It uses Algorithm 3.2 for REDUCERANK. Like the SS algorithm for frequent items, it assigns the counts for the second smallest counter (in this case squared singular value $\sigma_{\ell-1}^2$) to the direction of the smallest. Unlike the SS algorithm, we do not use $\sigma_{\ell-1}^2$ as the squared norm along each direction orthogonal to B, as that gives a consistent over-estimate.

Algorithm 3.2 REDUCERANK-SS(S)

$\delta_i \leftarrow \sigma_{\ell-1}^2$
return diag$(\sigma_1, \ldots, \sigma_{\ell-2}, 0, \sqrt{\sigma_\ell^2 + \delta_i})$.

We can also show similar error bounds for SSD; see supplementary material for proofs. A simple transformation of the output sketch $B \leftarrow \mathsf{SSD}(A)$ satisfies the three properties, although B itself does not.

Theorem 3.2. *After obtaining a matrix B from SSD on a matrix A with parameter ℓ, the following properties hold:*

- $\|A\|_F^2 = \|B\|_F^2$.
- *for any unit vector x and for $k < \ell/2 - 1/2$, we have* $|\|Ax\|^2 - \|Bx\|^2| \leq \|A - A_k\|_F^2/(\ell/2 - 1/2 - k)$.
- *for $k < \ell/2 - 1$ we have* $\|A - \pi_B^k(A)\|_F^2 \leq \|A - A_k\|_F^2(\ell - 1)/(\ell - 1 - 2k)$.

Setting $\ell = 2k + 2/\varepsilon + 1$ yields $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \varepsilon\|A - A_k\|_F^2$ and setting $\ell = 2k + 1 + 2k/\varepsilon$ yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A - A_k\|_F^2$.

3.1.4 Compensative Frequent Directions

Inspired by the isomorphic transformation [8] between the Misra-Gries [50] and the SpaceSaving sketch [49], which performed better in practice on the frequent items problem [22], we consider another variant of FD for matrix sketching. In the frequent items problem, this would return an identical result to SSD, but in the matrix setting it does not. We call this approach COMPENSATIVE FREQUENTDIRECTIONS (abbreviated CFD). In original FD, the computed sketch B underestimates the Frobenius norm of stream [39], in CFD we try to compensate for this. Specifically, we keep track of the total mass $\Delta = \sum_{i=1}^n \delta_i$ subtracted from squared singular values (this requires only an extra counter). Then we slightly modify the FD algorithm. In the final step where $B = S'V^T$, we modify S' to $\hat{S}$ by setting each singular value $\hat{\sigma}_j = \sqrt{\sigma'^2_j + \Delta}$, then we instead return $B = \hat{S}V^T$.

It now follows that for any $k \leq \ell$, including $k = 0$, that $\|A\|_F^2 = \|B\|_F^2$, that for any unit vector x we have $|\|Ax\|_F^2 - \|Bx\|_F^2| \leq \Delta \leq \|A - A_k\|_F^2/(\ell - k)$ for any $k < \ell$, and since V is unchanged that $\|A - \pi_B^k(A)\|_F^2 \leq \|A - A_k\|_F^2 \cdot \ell/(\ell - k)$. Also as in FD, setting $\ell = k + 1/\varepsilon$ yields $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \varepsilon\|A - A_k\|_F^2$ and setting $\ell = k/\varepsilon$ yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A - A_k\|_F^2$.

3.2 New Without Replacement Sampling Algorithms

TABLE 5
Theoretical Bounds for New Sampling Algorithms.

	ℓ	cov-err	ℓ	proj-err	runtime
Priority	d/ε^2	ε	ℓ	-	nnz$(A)\log\ell$
VarOpt	d/ε^2	ε	ℓ	-	nnz$(A)\log\ell$

As mentioned above, most sampling algorithms use *sampling with replacement* (SwR) of rows. This is likely because, in contrast to *sampling without replacement* (SwoR), it is easy to analyze and for weighted samples conceptually easy to compute. SwoR for unweighted data can easily be done with variants of reservoir sampling [59]; however, variants for weighted data have been much less resolved until recently [21], [35].

3.2.1 Priority Sampling

A simple technique [35] for SwoR on weighted elements first assigns each element i a random number $u_i \in \mathsf{Unif}(0, 1)$. This implies a priority $\rho_i = w_i/u_i$, based on its weight w_i (for matrix rows $w_i = \|a\|_i^2$). We then simply retain the ℓ rows with largest priorities, using a priority queue of size ℓ. Thus each step takes $O(\log \ell)$ time, but on randomly ordered data would take only $O(1)$ time in expectation since elements with $\rho_i \leq \tau$, where τ is the ℓth largest priority seen so far, are discarded. Retained

rows are given a squared norm $\hat{w}_i = \max(w_i, \tau)$. Rows with $w_i \geq \tau$ are always retained with original norm. Small weighted rows are kept proportional to their squared norms. The technique, Priority Sampling, is simple to implement, but requires a second pass on retained rows to assign final weights.

3.2.2 VarOpt Sampling

VarOpt (or Variance Optimal) sampling [21] is a modification of priority sampling that takes more care in selecting the threshold τ. In priority sampling, τ is generated so $\mathbf{E}[\sum_{a_i \in B} \hat{w}_i] = \|A\|_F^2$, but if τ is set more carefully, then we can achieve $\sum_{a_i \in B} \hat{w}_i = \|A\|_F^2$ deterministically. VarOpt selects each row with some probability $p_i = \min(1, w_i/\tau)$, with $\hat{w}_i = \max(w_i, \tau)$, and so exactly ℓ rows are selected.

The above implies that for a set L of ℓ rows maintained, there is a fixed threshold τ that creates the equality. We maintain this value τ as well as the t weights smaller than τ inductively in L. If we have seen at least $\ell + 1$ items in the stream, there must be at least one weight less than τ. On seeing a new item, we use the stored priorities $\rho_i = w_i/u_i$ for each item in L to either (a) discard the new item, or (b) keep it and drop another item from the reservoir. As the priorities increase, the threshold τ must always increase. It takes amortized constant time to discard a new item or $O(\log \ell)$ time to keep the new item, and does not require a final pass on L. We refer to it as VarOpt.

A similar algorithm using priority sampling was considered in a distributed streaming setting [37], which provided a high probability bound on cov-err. A constant probability of failure bound for $\ell = O(d/\varepsilon^2)$ and cov-err $\leq \varepsilon$, follows with minor modification from [24]. It is an open question to bound the projection error for these algorithms, but we conjecture the bounds will match those of Norm Sampling.

4 Experimental Setup

We used an OpenSUSE 12.3 machine with 32 cores of Intel(R) Core(TM) i7-4770S CPU(3.10 GHz) and 32GB of RAM. Randomized algorithms were run five times; we report the median error value.

Datasets. We compare performance of the algorithms on both synthetic and real datasets. In addition, we generate adversarial data to show that iSVD performs poorly under specific circumstances, this explains why there is no theoretical guarantee. Each data set is an $n \times d$ matrix A, and the n rows are processed one-by-one in a stream. Table 6 lists all datasets with information about their n, d, rank(A), numeric rank $\|A\|_F^2/\|A\|_2^2$, percentage of non-zeros (as nnz%, measuring sparsity), and excess kurtosis. We follow Fisher's distribution with baseline kurtosis (from normal distribution) is 0; positive excess kurtosis reflects heavier tails and negative excess kurtosis represents thinner tails. For Random Noisy, we generate the input $n \times d$ matrix A synthetically, mimicking the approach by Liberty [46]. We compose $A = SDU + F/\zeta$, where SDU is the m-dimensional signal (for $m < d$) and F/ζ is the (full) d-dimensional noise with ζ controlling the signal to noise ratio. Each entry $F_{i,j}$ of F is generated i.i.d. from a normal distribution $N(0, 1)$, and we set $\zeta = 10$. For the signal, $S \in \mathbb{R}^{n \times m}$ again we generate each $S_{i,j} \sim N(0, 1)$ i.i.d; D is diagonal with entries $D_{i,i} = 1 - (i-1)/d$ linearly decreasing; and $U \in \mathbb{R}^{m \times d}$ is just a random rotation. We use $n = 10000$, $d = 500$, and consider $m \in \{10, 20, 30, 50\}$ with $m = 30$ as default. In order to create Adversarial data, we constructed two orthogonal subspaces $S_1 = \mathbb{R}^{m_1}$ and $S_2 = \mathbb{R}^{m_2}$ ($m_1 = 400$ and $m_2 = 4$). Then we picked two separate sets of random vectors Y and Z and projected them on S_1 and S_2, respectively. Normalizing the projected vectors and concatenating them gives us the input matrix A. All vectors in $\pi_{S_1}(Y)$ appear in the stream before $\pi_{S_2}(Z)$; this represents a very sudden and orthogonal shift. As the theorems predict, FD and our proposed iterative algorithms adjust to this change and properly compensate for it. However, since $m_1 \geq \ell$, then iSVD cannot adjust and always discards all new rows in S_2 since they always represent the smallest singular value of $B_{[i]}$.

We consider 4 real-world datasets. ConnectUS is taken from the University of Florida Sparse Matrix collection [4]. ConnectUS represents a recommendation system. Each column is a user, and each row is a webpage, tagged 1 if favorable, 0 otherwise. It contains 171 users that share no webpage preferences with any other users. Birds [1] has each row represent an image of a bird, and each column a feature. PCA is a common first approach in analyzing this data, so we center the matrix. Spam [2] has each row represent a spam message, and each column some feature; it has dramatic and abrupt feature drift over the stream, but not as much as Adversarial. CIFAR-10 is a standard computer vision benchmark dataset for deep learning [44]. We will focus most of our experiments on three data sets Birds (dense, tall, large numeric rank), Spam (sparse, not tall, negative kurtosis, high numeric rank), and Random Noisy (dense, tall, synthetic). However, for some distinctions between algorithms require considering much larger datasets; for these we use CIFAR-10 (dense, not as tall, small numeric rank) and ConnectUS (sparse, tall, medium numeric rank). Finally, Adversarial and, perhaps surprisingly ConnectUS are used to show that using iSVD (which has no guarantees) does not always perform well.

5 Experimental Evaluation

We divide our experimental evaluation into four sections: The first three sections contain comparisons within algorithms of each group (sampling, projection, and iterative), while the fourth compares accuracy and run time of exemplar algorithm in each group against each other.

We measure error for all algorithms as we change the parameter ℓ (Sketch Size) determining the number

TABLE 6
Dataset Statistics.

DataSet	# datapoints	# attributes	rank	numeric rank	nnz%	excess kurtosis	$\|A\|_F^2/\|A-A_{10}\|_F^2$
Birds	11789	312	312	12.50	100	1.72	2.1658
Random Noisy	10000	500	500	14.93	100	0.95	9.1054
CIFAR-10	60000	3072	3072	1.19	99.75	1.34	13.4492
Connectus	394792	512	512	4.83	0.0055	17.60	2.0243
Spam	9324	499	499	3.25	0.07	3.79	2.0461
Adversarial	10000	500	500	1.69	100	5.80	6.4623

of rows in matrix B. We measure covariance error as err $= \|A^TA - B^TB\|_2/\|A\|_F^2$ (Covariance Error); this indicates for instance for FD, that err should be at most $1/\ell$, but could be dramatically less if $\|A-A_k\|_F^2$ is much less than $\|A\|_F^2$ for some not so large k. We consider proj-err $= \|A-\pi_{B_k}(A)\|_F^2/\|A-A_k\|_F^2$, always using $k=10$ (Projection Error); for FD we should have proj-err $\leq \ell/(\ell-10)$, and ≥ 1 in general. We also measure run-time as sketch size varies. Within each class, the algorithms are not dramatically different across sketch sizes. But across classes, they vary in other ways, and so in the global comparison, we will also show plots comparing runtime to cov-err or proj-err, which will help demonstrate and compare these trade-offs.

5.1 Sampling Algorithms

Fig.1 shows the covariance error, projection error, and runtime for the sampling algorithms as a function of sketch size, run on the Birds, Spam, and Random Noisy(30) datasets with sketch sizes from $\ell = 20$ to 100. We use parameter $k = 10$ for Leverage Sampling, the same k used to evaluate proj-err. First note that Deterministic Leverage performs quite differently than all other algorithms. The error rates can be drastically different: smaller on Random Noisy proj-err and Birds proj-err, while higher on Spam proj-err and all cov-err plots. The proven guarantees are only for matrices with Zipfian leverage score sequences and proj-err, and so when this does not hold it can perform worse. But when the conditions are right it outperforms the randomized algorithms since it deterministically chooses the best rows.

Otherwise, there is very small difference between the error performance of all randomized algorithms, within random variation. The small difference is perhaps surprising since Leverage Sampling has a stronger error guarantee, achieving a relative proj-err bound instead of an additive error of Norm Sampling, Priority Sampling and VarOpt Sampling which only use the row norms. Moreover Leverage Sampling and Deterministic Leverage Sampling are significantly slower than the other approaches since they require first computing the SVD and leverage scores. We note that if $\|A-A_k\|_F^2 > c\|A\|_F^2$ for a large enough constant c, then for that choice of k, the tail is effectively heavy, and thus not much is gained by the relative error bounds. Moreover, Leverage Sampling bounds are only stronger than Norm Sampling in a variant of proj-err where $[\pi_B(A)]_k$ (with best rank k applied *after* projection) instead of $\pi_{B_k}(A)$, and cov-err bounds are only known (see [24]) under some restrictions for Leverage Sampling, while unrestricted for the other randomized sampling algorithms.

5.2 Projection Algorithms

Fig.2 plots the covariance and projection error, as well as the runtime for various sketch sizes of 20 to 100 for the projection algorithms. All algorithms perform about the same in projection and covariance error up to the variation from randomness. Fast JLT performs a bit better than others in cov-err on Birds and Noisy Random, but we have chosen the best q parameter (sampling rate) by trial and error, so this may give an unfair advantage. But for runtime Hashing and OSNAP are significantly faster, especially as the sketch size grows. While Random Projections and Fast JLT appear to grow in time roughly linearly with sketch size, Hashing and OSNAP are basically constant.

5.3 Iterative Algorithms

Here we consider variants of FD. We first explore the α parameter in Parametrized FD, writing each version as α-FD. Then we compare against all of the other variants using exemplars from Parametrized FD.

In Fig.3 and 4, we explore the effect of the parameter α, and run variants with $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$, comparing against FD ($\alpha = 1$) and iSVD ($\alpha = 0$). Note that the guaranteed error gets worse for smaller α, so performance being equal, it is preferable to have larger α. Yet, we observe empirically on datasets Birds, Spam, and Random Noisy that FD is consistently the worst algorithm, and iSVD is fairly consistently the best, and as α decreases, the observed error improves. The difference can be quite dramatic; for instance in the Spam dataset, for $\ell = 20$, FD has err = 0.032 while iSVD and 0.2-FD have err = 0.008. Yet, as ℓ approaches 100, all algorithms seems to be approaching the same small error. In Fig.4, we explore the effect of α-FD on Random Noisy data by varying $m \in \{10, 20, 50\}$, and $m = 30$ in Fig.3. We observe that all algorithms get smaller error for smaller m (there are fewer "directions" to approximate), but that each α-FD variant reaches 0.005 err before $\ell = 100$, sooner for smaller α; eventually "snapping" to a smaller 0.002 err level. Next in Fig.5, we compare iSVD, FD, and 0.2-FD

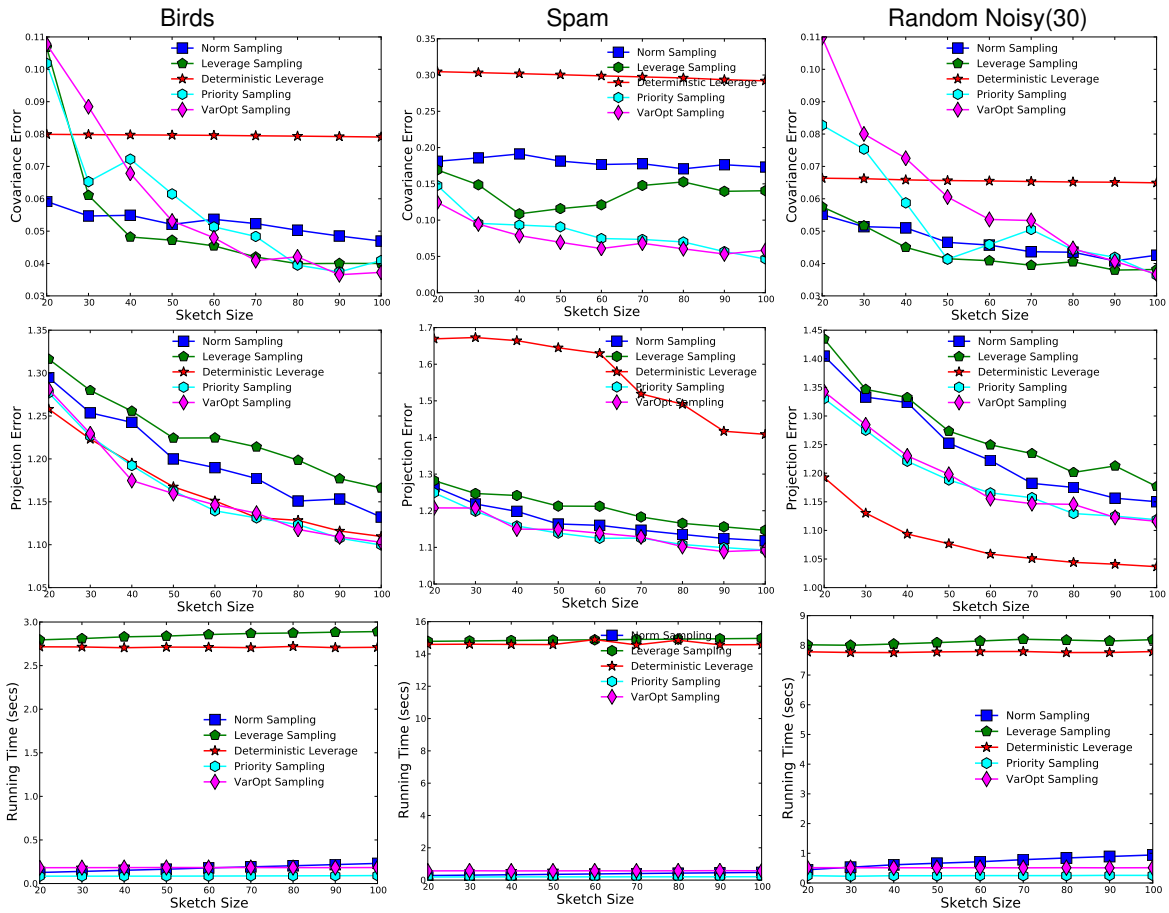

Fig. 1. Sampling algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

with two groups of variants: one based on SS streaming algorithm (CFD and SSD) and another based on Fast FD. We see that CFD and SSD typically perform slightly better than FD in cov-err and same or worse in proj-err, but not nearly as good as 0.2-FD and iSVD. Perhaps it is surprising that although SpaceSavings variants empirically improve upon MG variants for frequent items, 0.2-FD (based on MG) can largely outperform all the SS variants on matrix sketching. All variants achieve a very small error, but 0.2-FD, iSVD, and Fast 0.2-FD consistently matche or outperform others in both cov-err and proj-err while Fast FD incurs more error compared to other algorithms. We also observe that Fast FD and Fast 0.2-FD are significantly (sometimes 10 times) faster than FD, iSVD, and 0.2-FD. Fast FD takes less time, sometimes half as much compared to Fast 0.2-FD, however, given its much smaller error Fast 0.2-FD seems to have the best all-around performance.

5.3.1 Data adversarial to iSVD

Next, using the Adversarial construction we show that iSVD is not always better in practice. In Fig.6, in image on the left, we see that iSVD can perform much worse than other techniques. Although at $\ell = 20$, iSVD and FD roughly perform the same (with about err = 0.09), iSVD does not improve much as ℓ increases, obtaining only err = 0.08 for $\ell = 100$. On the other hand, FD (as well as CFD and SSD) decrease markedly and consistently to err = 0.02 for $\ell = 100$. The large-norm directions are the first 4 singular vectors (from the second part of the stream) and once these directions are recognized as having the largest singular vectors, they are no longer decremented in any of algorithms. Middle and right images of Fig.6 demonstrate the scalability of these approaches on a much larger real data set ConnectUS. As the derived bounds on covariance error based on sketch size do not depend on n, the number of rows in A, it is not surprising that the performance of most algorithms is unchanged. There are just a couple differences to point out. First, no algorithm in the middle and right figures of Fig.6 converges as close to 0 error as with the the smaller data sets (left figure of Fig.6); this is likely because with the much larger size, there is some variation that can not be captured even with $\ell = 100$ rows of a sketch. Second, iSVD performs noticeably worse than the other FD-based algorithms (although still significantly better than the leading randomized algorithms). This likely has to do with the sparsity of ConnectUS combined with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2539943, IEEE Transactions on Knowledge and Data Engineering

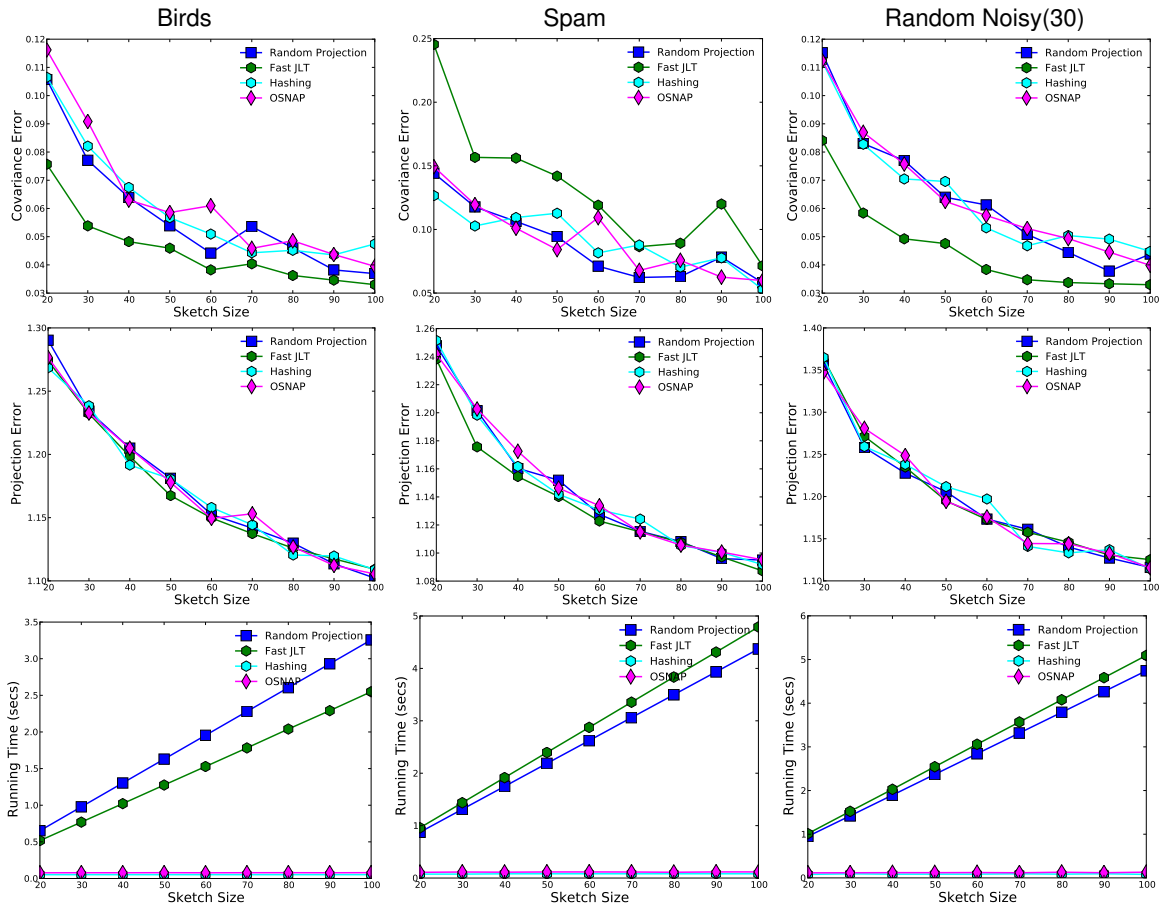

Fig. 2. Projection algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

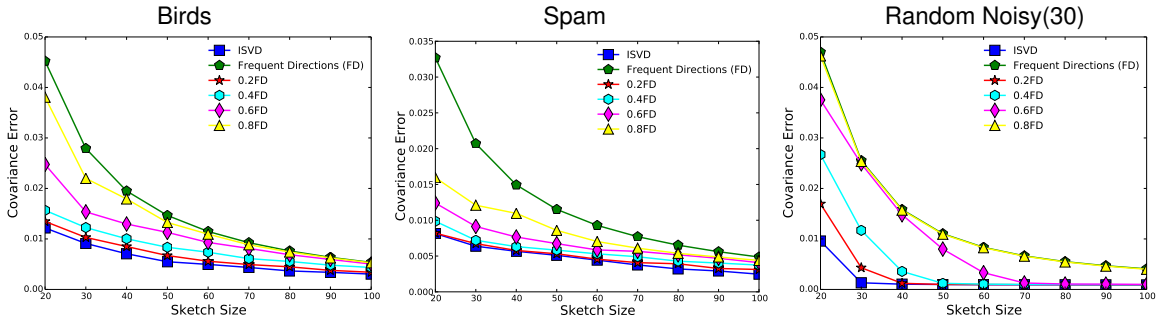

Fig. 3. Parametrized FD on Birds (left), and Spam (middle), Random Noisy(30) (right).

a data drift. After building up a sketch on the first part of the matrix, sparse rows are observed orthogonal to existing directions. The orthogonality, the same difficult property as in Adversarial, likely occurs here because the new rows have a small number of non-zero entrees, and all rows in the sketch have zeros in these locations; these correspond to the webpages marked by one of the unconnected users.

5.4 Global Comparison

Fig.7 shows the covariance error, projection error, as well as the runtime for various sketch sizes of $\ell = 20$ to 100 for the the leading algorithms from each category. We can observe that the iterative algorithms achieve much smaller errors, both covariance and projection, than all other algorithms, sometimes matched by Deterministic Leverage. However, they are also significantly slower (sometimes a factor of 20 or more) than other algorithms. The exception is Fast FD and Fast 0.2-FD, which are slower than the other algorithms, but not significantly so.

We also observe that for the most part, there is a negligible difference in the performance between the sampling algorithms and the projection algorithms, except for the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2539943, IEEE Transactions on Knowledge and Data Engineering

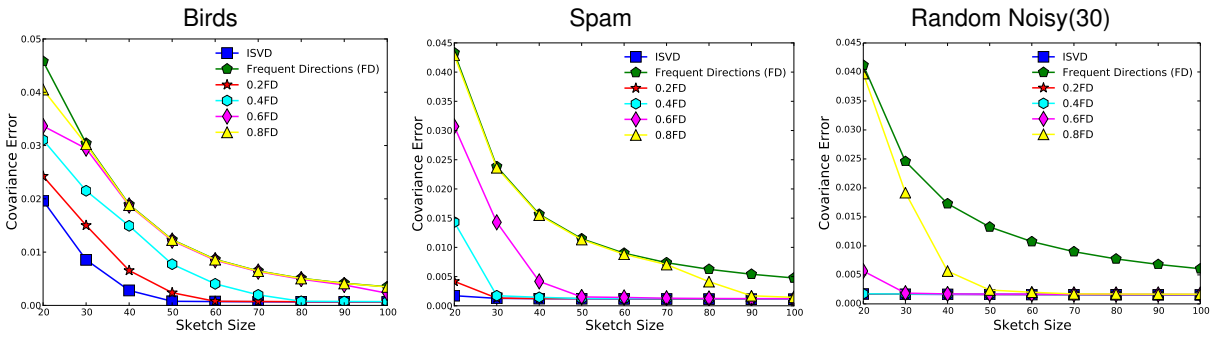

Fig. 4. Parametrized FD on Random Noisy for $m = 50$ (left), 20 (middle), 10 (right).

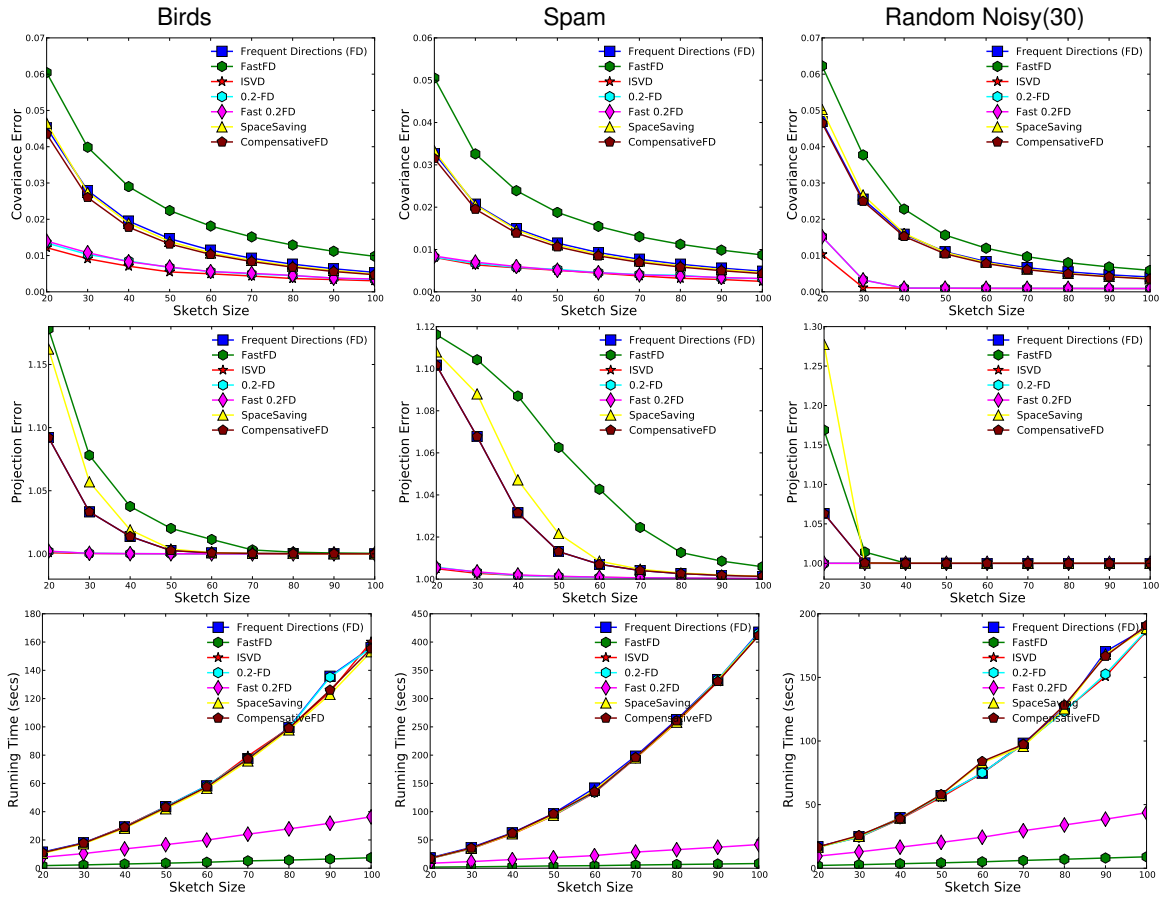

Fig. 5. Iterative algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

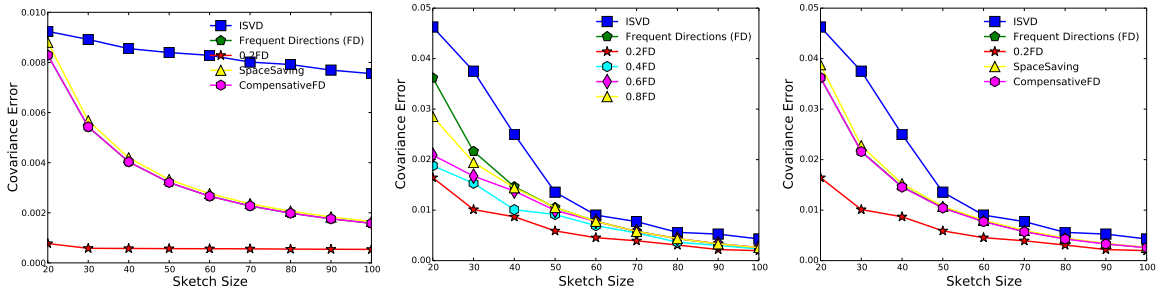

Fig. 6. Demonstrating dangers of iSVD on Adversarial data(left), Parameterized FD on ConnectUS dataset(middle); other iterative algorithms (right) on ConnectUS dataset.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2539943, IEEE Transactions on Knowledge and Data Engineering

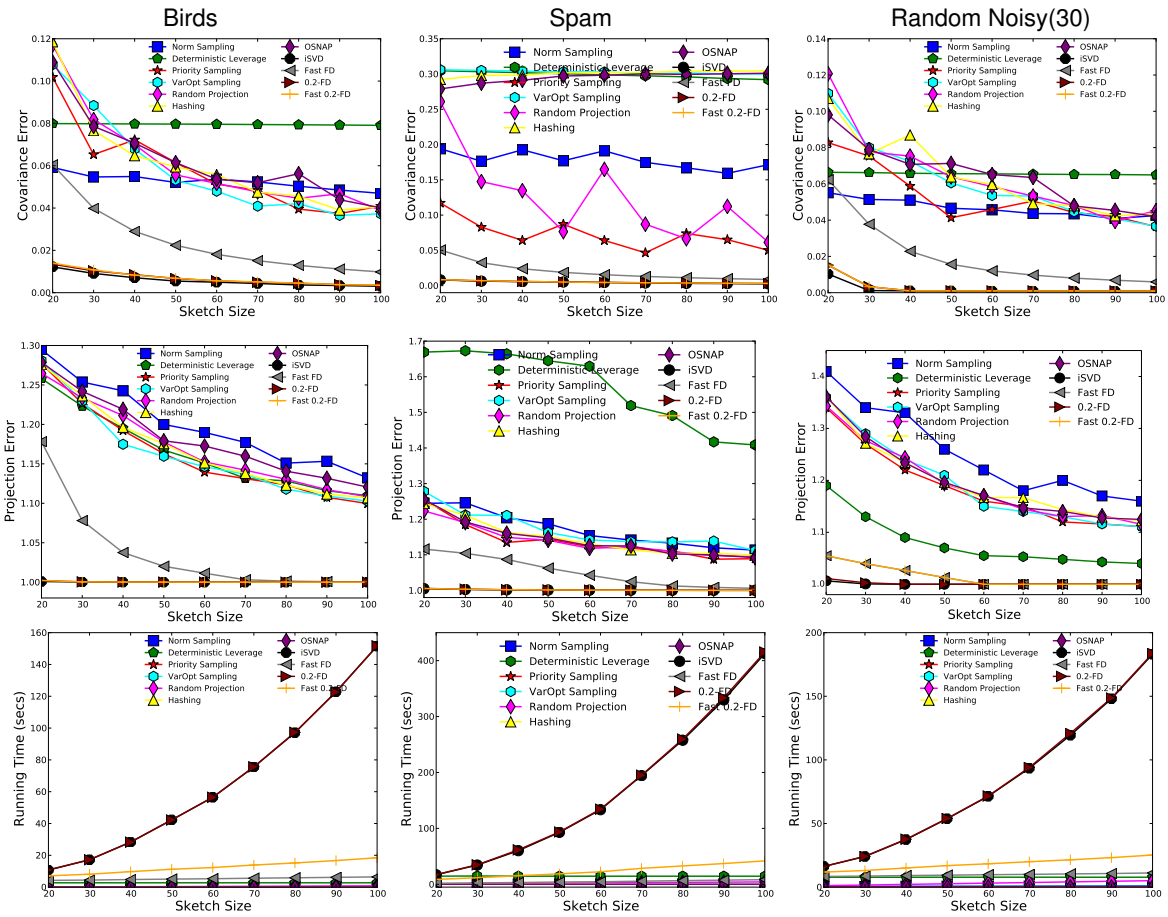

Fig. 7. Leading algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

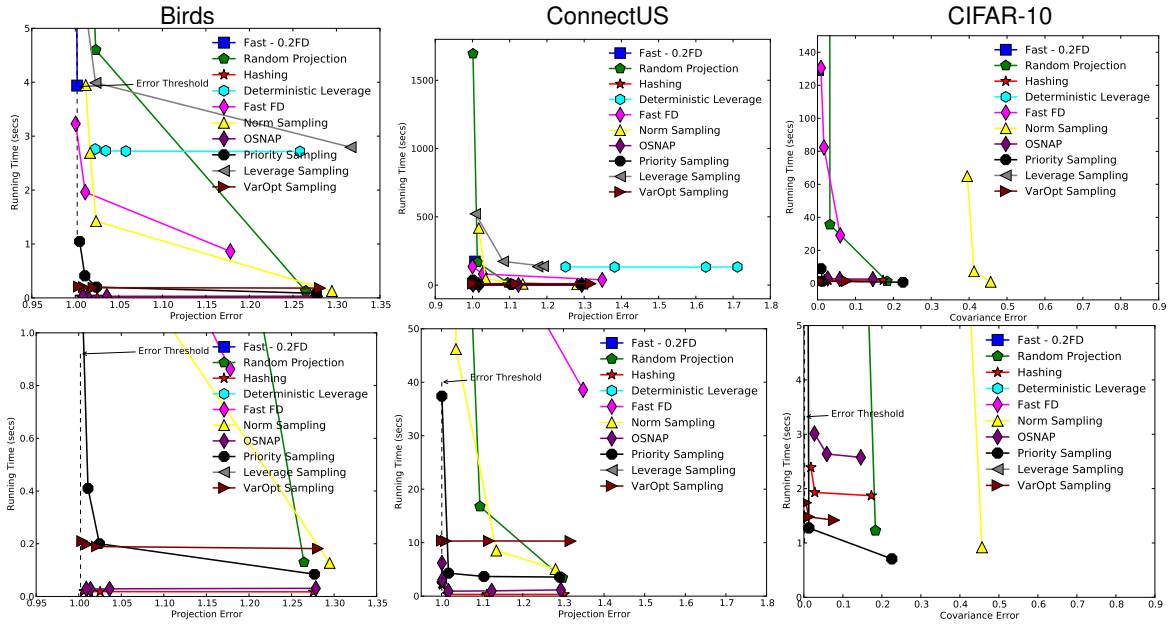

Fig. 8. Projection error versus time on Birds and ConnectUS as well as Covariance error versus time on CIFAR-10. The second line shows close-ups

Random Noisy dataset where Hashing and OSNAP result in worse projection error. However, if we allow a much large sketch size for faster runtime and small error, then these plots do not effectively demonstrate which algorithm

performs best. Thus in Fig.8 we run the leading algorithms on Birds as well as larger datasets, ConnectUS which is sparse and CIFAR-10 which is dense. We plot the error versus the runtime for various sketch sizes ranging up to $\ell = 10{,}000$. The top row of the plots shows most data points to give a holistic view, and the second row zooms in on the relevant portion. For some plots, we draw an *Error Threshold* vertical line corresponding to the error achieved by Fast 0.2-FD using $\ell = 20$. Since this error is typically very low, but in comparison to the sampling or projection algorithms Fast 0.2-FD is slow, this threshold is a useful target error rate for the other leading algorithms. We observe that Fast FD can sometimes match this error with slightly less time (see on Birds), but requires a larger sketch size of $\ell = 100$. Additionally VarOpt, Priority Sampling, Hashing, and OSNAP can often meet this threshold. Their runtimes can be roughly 100 to 200 times faster, but require sketch sizes on the order of $\ell = 10{,}000$ to match the error of Fast 0.2-FD with $\ell = 20$. Among these fast algorithms requiring large sketch sizes we observe that VarOpt scales better than Priority Sampling, and that these two perform best on CIFAR-10, the large dense dataset. They also noticeably outperform Norm Sampling both in runtime and error for the same sketch size. On the sparse dataset ConnectUS, algorithms Hashing and OSNAP seem to dominate Priority Sampling and VarOpt, and of those two Hashing performs slightly better. To put the space in perspective, on CIFAR-10 (n = $60{,}000$ rows, 1.4GB memory footprint), to approximately reach the *error threshold* Hashing needs $\ell = 10{,}000$ and 234MB in 2.4 seconds, VarOpt Sampling requires $\ell = 5{,}000$ and 117MB in 1.2 seconds, Fast FD requires $\ell = 100$ and 2.3MB in 130 seconds, and Fast 0.2-FD requires $\ell = 20$ and 0.48MB in 128 seconds. All of these will easily fit in memory of most modern machines. The smaller sketch by Fast 0.2-FD will allow expensive downstream applications (such as deep learning) to run much faster. Alternatively, the output from VarOpt Sampling (which maintains interpretability of original rows) could be fed into Fast 0.2-FD to get a compressed sketch in less time.

5.4.1 Reproducibility

Our results are publicly available at `http://aptlab.net/p/MatrixApx/MatrixApproxComparision`. This testbed facility, *APT* [3], allows researchers to perform experiments and keep them public for verification and validation.

References

[1] http://www.vision.caltech.edu/visipedia/CUB-200-2011.html.

[2] http://mlkd.csd.auth.gr/concept_drift.html.

[3] https://www.flux.utah.edu/project/apt.

[4] http://www.cise.ufl.edu/research/sparse/matrices.

[5] http://www.flux.utah.edu/project/emulab.

[6] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66:671–687, 2003.

[7] Dimitris Achlioptas and Frank McSherry. Fast computation of low rank matrix approximations. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[8] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *Proceedings of the 31st Symposium on Principles of Database Systems*, 2012.

[9] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of 38th ACM symposium on Theory of computing*, 2006.

[10] Nir Ailon and Edo Liberty. An almost optimal unrestricted fast Johnson-Lindenstrauss transform. In *Proceedings of 22nd ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[11] Arvind Arasu, Shivnath Babu, and Jennifer Widom. An abstract semantics and concrete language for continuous queries over streams and relations. 2002.

[12] Sanjeev Arora, Elad Hazan, and Satyen Kale. A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2006.

[13] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. *Lecture Notes in Computer Science*, pages 3–14, 2001.

[14] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near optimal column-based matrix reconstruction. In *Proceedings of 52nd Annual Symposium on Foundations of Computer Science*, 2011.

[15] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of 20th ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[16] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the 7th European Conference on Computer Vision*, 2002.

[17] Moses Charikan, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of International Colloquium on Automata, Languanges, and Programming*, 2002.

[18] Jianjun Chen, David J DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. *ACM SIGMOD Record*, 29:379–390, 2000.

[19] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 2009.

[20] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM symposium on Theory of computing*, 2013.

[21] Edith Cohen, Nick Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Stream sampling for variance-optimal estimation of subset sums. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[22] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. In *Proceedings of the 34th International Conference on Very Large Data Bases*, 2008.

[23] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse Johnson-Lindenstrauss transform. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, 2010.

[24] Amey Desai, Mina Ghashami, and Jeff M Phillips. Improved practical matrix sketching with guarantees. *arXiv preprint arXiv:1501.06561*, 2015.

[25] Amit Deshpande and Luis Radamacher. Efficient volume sampling for row/column subset selection. In *Proceedings of 51st IEEE Symposium on Foundations of Computer Science*, 2010.

[26] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithm*, 2006.

[27] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation,*

Randomization, and Combinatorial Optimization. Algorithms and Techniques. 2006.

[28] Petros Drineas and Ravi Kannan. Pass efficient algorithms for approximating large matrices. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[29] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36:132–157, 2006.

[30] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36:158–183, 2006.

[31] Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. Fast approximation of statistical leverage. *Journal of Machine Learning Research*, 13:3475–3506, 2012.

[32] Petros Drineas and Michael W. Mahoney. Effective resistances, statistical leverage, and applications to linear equation solving. In *arXiv:1005.3097*, 2010.

[33] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30:844–881, 2008.

[34] Petros Drineas and Anastasios Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111:385–389, 2011.

[35] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM*, 54:32, 2007.

[36] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51:1025–1041, 2004.

[37] Mina Ghashami, Feifei Li, and Jeff M. Phillips. Continuous matrix approximation on distributed data. In *Proceedings of the 40th International Conference on Very Large Data Bases*, 2014.

[38] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *arXiv preprint arXiv:1501.01711*, 2015.

[39] Mina Ghashami and Jeff M Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of 25th ACM-SIAM Symposium on Discrete Algorithms*, 2014.

[40] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Quicksand: Quick summary and analysis of network data. Technical report, DIMACS Technical Report, 2001.

[41] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHUP, 2012.

[42] Peter Hall, David Marshall, and Ralph Martin. Incremental eigenanalysis for classification. In *Proceedings of the British Machine Vision Conference*, 1998.

[43] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26:189–206, 1984.

[44] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Technical Report*, 2009.

[45] A. Levey and Michael Lindenbaum. Sequential Karhunen-Loeve basis extraction and its application to images. *IEEE Transactions on Image Processing*, 9:1371–1374, 2000.

[46] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

[47] Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning, NOW Publishers*, 3(2), 2011.

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

[49] Ahmed Metwally, Divyakant Agrawal, and Amr El. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 31:1095–1133, 2006.

[50] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2:143–152, 1982.

[51] Jelani Nelson and Huy L. Nguyen. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings of 54th IEEE Symposium on Foundations of Computer Science*, 2013.

[52] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, 1998.

[53] Dimitris Papapailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. In *Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2014.

[54] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77:125–141, 2008.

[55] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM*, 54:21, 2007.

[56] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[57] Mark Sullivan and Andrew Heybey. A system for managing large databases of network traffic. In *Proceedings of USENIX Annual Technical Conference*, 1998.

[58] Suresh Venkatasubramanian and Qiushi Wang. The Johnson-Lindenstrauss transform: An empirical study. In *Proceedings of ALENEX Workshop on Algorithms Engineering and Experimentation*, 2011.

[59] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11:37–57, 1985.

[60] Killian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of 26th Interantional Conference on Machine Learning*, 2009.

[61] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10:1–157, 2014.

[62] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

Amey Desai received a M.Sc. degree in Computer Science from University of Utah, and currently works as a software engineer at Urban Engines in California.

Mina Ghashami received a M.Sc. degree in Software Engineering at Sharif University of Technology and is currently completing a Ph.D. in Computer Science at University of Utah.

Jeff M. Phillips received a BS in Computer Science and BA in Math from Rice University. He earned a Ph.D. from Duke University in Computer Science. He was a CI Postdoctoral Fellow at the University of Utah. Currently he is an Assistant Professor in the School of Computing at University of Utah.