ON FREQUENT DIRECTIONS, A STREAMING MATRIX SKETCHING ALGORITHM

by

Mina Ghashami

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

December 2017

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Mina Ghashami**

has been approved by the following supervisory committee members:

Jeffrey M. Phillips,	Chair(s)	**30 May 2017** Date Approved
Edo Liberty,	Member	**10 June 2017** Date Approved
Vivek Srikumar,	Member	**30 May 2017** Date Approved
Suresh Venkatasubramanian,	Member	**30 May 2017** Date Approved
Hari Sundar,	Member	**30 May 2017** Date Approved

by **Ross T. Whitaker**, Chair/Dean of

the Department/College/School of **Computing**

and by **David B. Kieda**, Dean of The Graduate School.

ABSTRACT

Matrices are essential data representations for many large-scale problems in data analytics; for example, in text analysis under the bag-of-words model, a large corpus of documents is often represented as a matrix. Many data analytic tasks rely on obtaining a summary (a.k.a sketch) of the data matrix. Using this summary in place of the original data matrix saves on space usage and run-time of machine learning algorithms. Therefore, sketching a matrix is often a necessary first step in data reduction, and sometimes has direct relationships to core techniques including PCA, LDA, and clustering.

In this dissertation, we study the problem of matrix sketching over data streams. We first describe a deterministic matrix sketching algorithm called FrequentDirections. The algorithm is presented an arbitrary input matrix $A \in \mathbb{R}^{n \times d}$ one row at a time. It performs $O(d\ell)$ operations per row and maintains a sketch matrix $B \in \mathbb{R}^{\ell \times d}$ such that for any $k < \ell$,

$$\|A^T A - B^T B\|_2 \leq \|A - A_k\|_F^2 / (\ell - k) \quad \text{and} \quad \|A - \pi_{B_k}(A)\|_F^2 \leq \big(1 + \frac{k}{\ell - k}\big) \|A - A_k\|_F^2 \,.$$

Here, A_k stands for the minimizer of $\|A - A_k\|_F$ over all rank k matrices (similarly B_k), and $\pi_{B_k}(A)$ is the rank k matrix resulting from projecting A on the row span of B_k. We show both of these bounds are the best possible for the space allowed, the sketch is mergeable, and hence trivially parallelizable. We propose several variants of FrequentDirections that improve its error-size tradeoff, and nearly matches the simple heuristic *Iterative SVD* method in practice.

We then describe SparseFrequentDirections for sketching sparse matrices. It resembles the original algorithm in many ways including having the same optimal asymptotic guarantees with respect to the space-accuracy tradeoff in the streaming setting, but unlike FrequentDirections which runs in $O(nd\ell)$ time, SparseFrequentDirections runs in $\tilde{O}\left(\text{nnz}(A)\ell + n\ell^2\right)$ time.

We then extend our methods to *distributed streaming model*, where there are m distributed sites each observing a distinct stream of data, and which has a communication channel with a coordinator. The goal is to track an ε-approximation (for $\varepsilon \in (0,1)$) to the norm of the matrix along any direction. We present novel algorithms to address this problem. All our methods satisfy an additive error bound that for any unit vector x, $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2$ holds.

To all who provide educational opportunities for children across the globe.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION AND SYMBOLS

$[d]$	Set $\{1, \cdots, d\}$
$\|x\|$	Euclidean norm of vector x, i.e. $\|x\| = (\sum_i x_i^2)^{1/2}$
$\|A\|_2$	Spectral (Operator) norm of a matrix A
$\|A\|_F$	Frobenius norm of a matrix A
$\mathrm{SVD}(A)$	Singular value decomposition of matrix A
$\mathrm{rank}(A)$	Numerical rank of matrix A
$\pi_X(r)$	projection operation of row vector r onto the subspace spanned by matrix X
I_ℓ	Denotes $\ell \times \ell$ Identity matrix
$X^\dagger$	Moore-Penrose pseudoinverse of X
ε	Error parameter between zero and one
δ	Failure parameter between zero and one
$\mathrm{nnz}(A)$	Number of non-zero entries of A
$\mathrm{SVD}(A)$	Singular Value Decomposition of A

CHAPTER 1

INTRODUCTION

Large data matrices are found in numerous domains, such as scientific computing, multimedia applications, networking systems, server and user logs, and many others [86, 87, 91, 106]. Since such data are huge in size and often generated continuously, it is important to process them in streaming fashion and maintain an approximating summary or a sketch. In this chapter, we formally define the *Matrix Sketching* problem, and review different existing sketching paradigms.

1.1 Matrix Sketching Problem

A standard task in scientific computing is to compute for a given $n \times d$ matrix A, a smaller lower rank matrix B that approximates A well. Such matrix B is called a *sketch* of A. Often B is smaller than A in one dimension, e.g., B can be $\ell \times d$ for $\ell \ll n$, and has a small rank k such that $k \leq \ell, \text{rank}(A)$.

It is well-known that the best rank-k approximation (under Frobenius or 2 norm) A_k can be computed using the singular value decomposition (SVD); however this takes $O(nd \min(n,d))$ time and $O(nd)$ memory. The SVD(A) produces three matrices U, S, and V where U and V are orthonormal, of size $n \times n$ and $d \times d$, respectively, and S is $n \times d$ but only has non-zero elements on its diagonal $\{\sigma_1, \ldots, \sigma_d\}$. Let U_k, S_k, and V_k be the first k columns of each matrix, then $A = USV^T$ and $A_k = U_k S_k V_k^T$. Note that although A_k requires $O(nd)$ space, the set of matrices $\{U_k, S_k, V_k\}$ requires only a total of $O((n+d)k)$ space (or $O(nk)$ if the matrix is tall). Moreover, even the set $\{U, S, V\}$ really only takes $O(nd + d^2)$ space since we can drop the last $n - d$ columns of U and the last $n - d$ rows of S without changing the result.

Computing SVD 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 [21], financial tickers [30, 129], on-line auctions [13], network traffic [72, 114], and telecom call records [39]. Computing such sketches efficiently is an important building block in modern algorithms [7, 48, 60, 92, 105] as sketching is either a necessary first step in data reduction or has direct relationships to core techniques including PCA, LDA, and clustering.

In the streaming version, the goal is to compute something that replicates the effect of A_k using less space and only seeing each row once.

1.2 Data Streaming Models

Data streaming model considers computation on a large possibly infinite dataset $A = (a_1, \cdots, a_n, \cdots)$ where data items arrive one at a time in arbitrary order. Thus some or all of the input data is not available for random access from disk or memory, but rather data items arrive one by one. Data streams are often denoted as an ordered sequence of updates that must be accessed in order and can be read only once or a few number of times. Based on the type of updates, there exist two common data streaming models:

cash-register model: In this model, each update is of the form (i, c), so that a_i is incremented by some positive integer c. Throughout this dissertation, we consider the special case of $c = 1$ and denote it as *row-update model*, as it allows insertion of rows only.

turnstile model: This model is more general and allows for updates of form (i, c) where c can be a negative number.

Data streaming model also enforces that only a small amount of memory is available at any given time. This small space constraint is critical when the full dataset cannot fit in memory or disk. Typically, the amount of space required is traded off with the accuracy of the computation on A. Usually the computation results in some summary or sketch of A, and this trade-off determines how accurate one can be with the available space resources.

1.2.1 Distributed Streaming Model

In this model, there are multiple distributed sites and a single coordinator which has a two-way communication channel to all sites. Each site observes a disjoint stream of data and together they attempt to monitor a function at the coordinator site. Due to its wide applications in practice [37], a flurry of work has been done under this setting. This model is more general than the *data streaming model* [16] that maintains a function at a single site with small space. It is also different from *communication model* [127] in which data are (already) stored at multiple sites and the goal is to do a *one-time* computation of a target function. The key resources to optimize in distributed streaming model is not just the space needed at the coordinator or each site, but the communication between the sites and coordinator.

This model appears in many applications in distributed databases, wireless sensor networks, cloud computing, etc.[101] where data sources are distributed over a network and collecting all data together at a central location is not a viable option. In many such environments queries must be answered continuously, based on the total data that have arrived so far. Since data are continuously changing in these applications, query results can also change with time. So the challenge is to minimize the communication between sites and the coordinator while maintaining accuracy of results at all times.

1.3 Sketching Paradigms

There exist several approaches to sketching an input matrix $A \in \mathbb{R}^{n \times d}$:

Sparsifying the Matrix: This approach retains a small number of non-zero elements of the matrix [6, 7, 14, 56]. These algorithms typically are assumed to know the $n \times d$ dimensions of A, and are thus not directly applicable in our model.

Random Projection: This approach randomly combines rows of matrix A [93, 105, 113, 117]. An efficient variant [4], computes the sketch B as $B = RA$ where R is an $\ell \times n$ matrix such that each element $R_{i,j} \in \{-1/\sqrt{\ell}, 1/\sqrt{\ell}\}$ is chosen uniformly. This is easily computed in a streaming fashion, while requiring at most $O(\ell d)$ space and $O(\ell d)$ operations per row update. Sparser constructions of random projection matrices are known to exist [40, 82].

Hashing: This approach [33] uses an extra sign-hash function to replicate the count-sketch [28] with matrix rows. Specifically, the sketch B is initialized as the $\ell \times d$ all zeros matrix, then each row a_i of A is added to $h(i)$-th row of $B_{h(i)} \leftarrow B_{h(i)} +$

$s(i)a_i$, where $h : [n] \to [\ell]$ and $s : [n] \leftarrow \{-1, 1\}$ are perfect hash functions. There is no harm in assuming such functions exist since complete randomness is naïvely possible without dominating either space or running time. This method is often used in practice by the machine learning community and is referred to as "feature hashing" [121].

Sampling: This sketching approach is to find a small subset of matrix rows (and/or columns) that approximate the entire matrix. This problem is known as the *Column Subset Selection Problem* and has been thoroughly investigated [22, 23, 46, 48, 55, 61]. Recent results offer algorithms with almost matching lower bounds [22, 31, 46]. A simple streaming solution to the *Column Subset Selection Problem* is obtained by sampling rows from the input matrix with probability proportional to their squared ℓ_2 norm. Specifically, each row B_j takes the value $A_i/\sqrt{\ell p_i}$ iid with probability $p_i = \|A_i\|^2/\|A\|_F^2$. The space it requires is $O(\ell d)$ in the worst case but it can be much lower if the chosen rows are sparse. Since the value of $\|A\|_F$ is not a priori known, the streaming algorithm is implemented by ℓ independent reservoir samplers, each sampling a single row according to the distribution. The update running time is therefore $O(d)$ per row in A. Despite this algorithm's apparent simplicity, providing tight bounds for its error performance required over a decade of research [10, 48, 55, 61, 103, 112, 119]. Such advanced algorithms utilize the leverage scores of the rows [54] and not their squared ℓ_2 norms.

Iterative Sketching: This sketching approach maintains a sketch in memory and receives rows of the input matrix one by one, process each row and iteratively update the sketch. Examples of these method include *incremental PCA* [26, 74, 76, 90, 111]. These approaches attempt to maintain the PCA of a dataset A (using the SVD and a constant amount of additional bookkeeping space) as each row of A arrives in a stream. In particular, after $i-1$ rows they consider maintaining A_k^i, and on a new row a_i compute $\text{SVD}([A_k^i; a_i]) = U^i S^i (V^i)^T$ and, then only retain its top rank k approximation as $A_k^{i+1} = U_k^i S_k^i (V_k^i)^T$.

1.4 Error Bounds and Measurement

The accuracy of a sketch matrix B can be measured in several ways. In most common ways, one constructs an $n \times d$, rank k matrix $\hat{A}$ from B and sometimes also using A, and measures approximation error between A and $\hat{A}$. If $\hat{A}$ is derived entirely from B, the result is called a *construction* result, and provides the stronger *construction* bound, that $\|A - \hat{A}\|_\xi \leq (1+\varepsilon)\|A - A_k\|_\xi$ for $\xi = \{2, F\}$ and for some parameter $\varepsilon \in (0,1)$.

Unless A is sparse, then storing $\hat{A}$ explicitly may require $\Omega(nd)$ space, so that is why various representations of $\hat{A}$ are used in its place. This can include decompositions similar to the SVD, e.g., a CUR decomposition [48, 54, 94] where $\hat{A} = CUR$ and where U is small and dense, and C and R are sparse and skinny, or others [33] where the middle matrix is still diagonal. The sparsity is often preserved by constructing the wrapper matrices (e.g., C and R) from the original columns or rows of A. There is an obvious $\Omega(n + d)$ space lower bound for any construction result in order to preserve the column and the row space.

In cases with space independent of n, either $\hat{A}$ implicitly depends on A, or it requires another pass over the data. In the former case, $\hat{A}$ can be defined as $\hat{A} = \pi_B^k(A)$, it takes B_k which is the best rank-k approximation to B, and projects A onto it. In the latter case $\hat{A}$ is defined as $\hat{A} = \Pi_B^k(A)$, this definition projects A onto B, and *then* takes the best rank-k approximation of the result. Note that $\pi_B^k(A)$ is better than $\Pi_B^k(A)$, since it knows the rank-k subspace to project onto without re-examining A.

Both cases provide the weaker *projection* bound, as it cannot actually represent $\hat{A}$ without making another pass over A to do the projection. When B or B_k is composed of a set of ℓ rows (and perhaps B_k is only k rows) then the total size is only $O(d\ell)$ (allotting constant space for each entry), so it does not depend on n. This is a significant advantage in tall matrices where $n \gg d$. Sometimes this subspace approximation is sufficient for downstream analysis, since the rowspace is still (approximately) preserved. For instance, in PCA the goal is to compute the most important directions in the row space.

Using either definition of $\hat{A}$, we define two error bounds that we use throughout this dissertation: (1) *projection error* as proj-err $= \|A - \hat{A}\|_F^2 / \|A - A_k\|_F^2$, and (2) *covariance error* as covar-err $= \|A^T A - B^T B\|_2 / \|A\|_F^2$.

One can also bound $\|A^T A - B^T B\|_2 / \|A - A_k\|_F^2$, but this has an extra parameter k, and is less clean to empirically evaluate. The covariance error captures the norm of A along all directions and does not require $\Omega(n)$ space, whereas the projection error indicates how accurate the choice of the subspace of B is, but requires another pass or $\Omega(n)$ space.

1.5 Layout of Dissertation

This dissertation is mainly about FrequentDirections algorithm (FD for short), which was initially introduced by Edo Liberty [92] at KDD 2013 conference. Throughout this dissertation we explain FD's connection to other algorithms, its improved error analysis, and its various extensions.

More specifically, in Chapter 2, we discuss the *Item Frequency Approximation* problem which is tightly connected to *Matrix Sketching* problem and has been inspiring for FrequentDirections algorithm. We introduce our novel algorithms for *Item Frequency Approximation* problem in distributed streaming model in this chapter. Part of our work in this section is published in VLDB 2014 conference [64].

Next, in Chapter 3, we explain FrequentDirections algorithm which was initially introduced by Edo Liberty [91]. We describe our work on improving the error analysis of FrequentDirections, which has appeared as a conference paper in SODA 2014 [70]. In section 3.2 we discuss the optimality of FrequentDirections with respect to the space usage-accuracy trade off. Results of this chapter appeared in SICOMP 2016 [66] jointly with Jeff Phillips, Edo Liberty and David Woodruff, combining three papers [70, 91, 125]. We only state a lower bound result from David Woodruff [124] in this dissertation for completeness.

In Chapter 4, we discuss various extensions of FrequentDirections which are all inspired by different methods on *Item Frequency Approximation* problem. Our work in this chapter is published in ESA 2014 [63] and TKDE 2016 [43].

In Chapter 5, we discuss other matrix sketching approaches in detail. Part of the work in this section has appeared in the M.Sc. Dissertation of Amey Desai, who was a collaborator on the work. Results of this chapter are included in our journal paper at TKDE 2016 [43].

In Chapter 6, we introduce SparseFrequentDirections which is a variant of FrequentDirections for processing sparse matrices efficiently. This section was a joint work with Edo Liberty and is published at KDD 2016 [65].

In Chapter 7, we extend matrix sketching to distributed streaming model where one needs to track a matrix approximation at all times. We define the new data model and propose novel algorithms to address the problem. Our methods in this chapter are inspired by the methods of Chapter 2 and are published in the same paper as them in VLDB 2014 conference [64].

In Chapter 8, we conclude the dissertation.

CHAPTER 2

ITEM FREQUENCY APPROXIMATION

In this chapter, we overview the item frequency approximation problem, and its connection to matrix sketching. The reason for including this chapter in the dissertation is that one of our algorithms for sketching matrices is an extension of a well known algorithm for approximating item frequencies in streams. Understanding these algorithms helps in getting a more intuitive understanding of our matrix sketching method in Chapter 3.

2.1 Problem Definition

In Item Frequency Approximation problem, there is a stream $A = \{a_1, \cdots, a_n\}$ of n items, where each $a_i \in [d]$. We define $f_j = |\{a_i \in A \mid a_i = j\}|$ to be the frequency of item j. In words, f_j is the number of times item j appears in the stream. It is trivial to produce all item frequencies using $O(d)$ space simply by keeping a counter for each item. Although this method computes exact frequencies, it uses space linear to the size of domain which might be unacceptable if d is very large. Therefore, we are interested in using sublinear space while producing approximate frequencies $\hat{f}_j$.

This problem is also studied under FrequentItems or HeavyHitters in literature. The ϕ-heavy hitters are defined as those items e with $f_e \geq \phi n$ for some parameter $\phi \in (0,1)$. Since computing exact ϕ-heavy hitters incurs high cost and is often unnecessary, we allow an ε-approximation. Then the more commonly studied (ε, ϕ)-heavy hitters problem must find the set of items e with $f_e \geq \phi n$, may or may not return items e with $(\phi - \varepsilon)n \leq f_e < \phi n$, and must not return items e with $f_e < (\phi - \varepsilon)n$. From this point on, we refer to these collectively as FrequentItems.

2.2 Sketch Based Solutions

This problem has received multiple solutions in literature; below we describe the most relevant ones to our work.

2.2.1 Misra-Gries (MG) Sketch

This simple and elegant solution is by Misra and Gries [99], which we denote MG sketch. Their algorithm employs a map of $\ell < d$ items to ℓ counters. It maintains the invariant that at least one of the items is mapped to a counter of value zero. The algorithm counts items in the trivial way. If it encounters an item for which it has a counter, that counter is incremented or else, it replaces one of the items mapping to zero value with the new item (setting the counter to one). This is continued until the invariant is violated, that is, ℓ items map to counters of value at least 1. At this point, all counts are decreased by the same amount until at least one item maps to a zero value. The final values in the map give approximate frequencies $\hat{f}_j$ such that $0 \leq f_j - \hat{f}_j \leq n/\ell$ for all $j \in [d]$; unmapped j implies $\hat{f}_j = 0$ and provides the same bounds. The reason for this is simple: since we decrease ℓ counters simultaneously, we cannot do this more that n/ℓ times. And since we decrement *different* counters, each item's counter is decremented at most n/ℓ times. Variants of this very simple (and clever) algorithm were independently discovered several times [41, 73, 83, 97].[1]

Later, Berinde *et al.* [19] proved a tighter bound for FrequentItems. Consider summing up the errors as $\hat{R}_k = \sum_{i=k+1}^{d} |f_j - \hat{f}_j|$ and assume without loss of generality that $f_j \geq f_{j+1}$ for all j. Then, it is obvious that counting only the top k items exactly is the best possible strategy if only k counters are allowed. That is, the optimal solution has a cost of $R_k = \sum_{i=k+1}^{d} f_j$. Berinde *et al.* [19] showed that if FrequentItems uses $\ell > k$ counters then for any $j \in [d]$ we can bound $|f_j - \hat{f}_j| \leq R_k/(\ell - k)$. By summing up the error over top k items, it is easy to obtain that $\hat{R}_k < \frac{\ell}{\ell-k} R_k$. Setting $\ell = \lceil k + k/\varepsilon \rceil$ yields the convenient form of $\hat{R}_k < (1 + \varepsilon) R_k$. The authors also show that to get this kind of guarantee in the streaming setting $\Omega(k/\varepsilon)$ bits are indeed necessary. This make FrequentItems optimal up to the word size factor in that regard.

2.2.2 SpaceSaving Sketch

The SpaceSaving sketch [97] is a counter-based sketch that is simple and deterministic and works very similar to MG sketch. For a parameter ℓ, the SpaceSaving sketch maintains ℓ items with their associated counters. As it receives an item x in the stream, it does one

[1] The reader is referred to [83] for an efficient streaming implementation.

of the following: (1) If x already exists in the sketch, its counter is increased by one. (2) If x does not exist and the sketch currently has fewer than ℓ items, we add x into the sketch and sets its counter to one. (3) If the sketch is full, i.e., it contains exactly ℓ items and x is not one of them, we find any item y with the minimum counter value, replace y with x , and increase the counter by one.

It is shown that in a stream of length n, the SpaceSaving sketch estimates the frequency of any item with error atmost n/ℓ. Setting $\ell = 1/\varepsilon$ for an error parameter $\varepsilon \in (0, 1)$, SpaceSaving solves the frequency estimation problem with additive error εn with space usage $O(1/\varepsilon)$. It is clear to see that SpaceSaving can be used to report the heavy hitters in $O(1/\varepsilon)$ time by going through all counters; any item not maintained cannot have frequency higher than εn. Authors in [8] showed that MG and the SpaceSaving summaries for heavy hitters are isomorphic.

2.2.3 Count-Min Sketch

The Count-Min Sketch [36] is a randomized and hash-based sketch. It maintains a $\ell \times w$ matrix (referred to as *count*), where $w = \lceil e/\varepsilon \rceil$ and $\ell = \lceil \ln(1/\delta) \rceil$ for error parameter and failure parameter $\varepsilon, \delta \in (0, 1)$. It also requires ℓ hash functions $h_1, \cdots, h_\ell : [d] \to [w]$. Each hash function is associated with a row in the matrix, and all hash functions are pair-wise independent. To propagate the count matrix, we apply all hash functions on each data item in the stream as we encounter it. Each hash functions h_i maps an item x to cell $count[i, h_i(x)]$ in ith row of the count matrix, and increment its value by one. At any time in the stream, the frequency of an item x is estimated as $\hat{f}_x = \min_{1 \le i \le \ell} count[i, h_i(x)]$. It is shown that Count-Min sketch guarantees $0 \le \hat{f}_x - f_x \le \varepsilon n$ with probability at least $1 - \delta$.

2.3 Connection to Matrix Sketching

There is a tight connection between the matrix sketching problem and the frequent items problem. Let A be a matrix that is given as a stream of its rows. For now, let us constrain the rows of A to be indicator vectors. In other words, we have $a_i \in \{e_1, ..., e_d\}$, where e_j is the jth standard basis vector. Note that such a matrix can encode a stream of items (as above) [92]. If the ith element in the stream is j, then the ith row of the matrix is set to $a_i = e_j$. The frequency f_j can be expressed as $f_j = \|Ae_j\|^2$. Assume that we construct a matrix $B \in \mathbb{R}^{\ell \times d}$ as follows. First, we run FrequentItems on the input. Then, for every item j for which $\hat{f}_j > 0$ we generate one row in B equal to $\hat{f}_j^{1/2} \cdot e_j$. The result is a low rank approximation of A. Note that $\text{rank}(B) = \ell$ and that $\|Be_j\|^2 = \hat{f}_j$. Notice also that $\|A\|_F^2 = n$ and that $A^T A = \text{diag}(f_1, \ldots, f_d)$ and that $B^T B = \text{diag}(\hat{f}_1, \ldots, \hat{f}_d)$. Porting the results we obtained from FrequentItems we get that $\|A^T A - B^T B\|_2 = \max_j |f_j - \hat{f}_j| \leq \|A\|_F^2 / (\ell - k)$. Moreover, since the rows of A (corresponding to different counters) are orthogonal, the best rank k approximation of A would capture exactly the most frequent items. Therefore, $\|A - A_k\|_F^2 = R_k = \sum_{i=k+1}^{d} f_j$. Using the quantities above we can also reach the conclusion that $\|A - \pi_B^k(A)\|_F^2 \leq \frac{\ell}{\ell - k} \|A - A_k\|_F^2$. We observe that, for the case of matrices whose rows are basis vectors, FrequentItems actually provides a very efficient low rank approximation result.

2.4 Heavy Hitters in Distributed Streams

Tracking frequent items (a.k.a heavy hitters) in distributed streaming model is a fundamental problem and it is vastly studied in literature [17, 84, 95, 128]. Below, we first define the distributed streaming model formally, then describe the most relevant prior works to our proposed methods in this model.

Definition 2.4.1 (Distributed streaming model)**.** *Formally, assume there are m distributed sites $S_1, \cdots, S_m$ and a single coordinator C which has two-way communication channel to all sites. Let $A = (a_1, \cdots, a_n, \cdots)$ be an unbounded stream of items. At any time t, item a_n appears at exactly one of the sites. Although we do not place a bound on the number of items, we let N denote the total size of the stream at the time when a query q is performed. This allows us to discuss results in terms of n at a given point, and in terms of N for the entire run of the algorithm until the time of a query q.*

Definition 2.4.2 (Tracking (ε, ϕ)-heavy hitters in distributed streaming model)**.** *The goal is to continuously maintain an ε approximation to ϕ-heavy hitters in coordinator C.*

Babcock and Olston [17] designed some deterministic heuristics called as *top-k monitoring* to compute top-k frequent items. In their approach, the coordinator computes an initial top-k set by querying the sites, and installs some arithmetic constraints at sites over the partial values maintained there to ensure the continuing validity of initial set.

As update occurs, sites track changes to their partial values and ensure constraints are satisfied. Whenever a constraint at a site becomes violated, the site informs coordinator and coordinator determines whether the top-k set is still accurate. If it is not accurate, coordinator selects a new one if necessary, and then modifies the constraints as needed at a subset of the sites. Fuller and Kantardzid modified their technique and proposed *FIDS* [62], a heuristic method, to track the heavy hitters while reducing communication cost and improving overall quality of results.

Manjhi *et al.* [95] studied ϕ-heavy hitter tracking in a hierarchical communication model, in which periodically at the end of every epoch, each site sends to the root of the hierarchy the counts of all items that appeared at the site over last epoch. The root site combines the count it received from all sites and outputs items whose relative frequency exceed the threshold ϕ. To reduce communication and space requirements, they have defined precision gradient at each level of hierarchy.

Cormode and Garofalakis [35] proposed another method by maintaining a summary of the input stream and a prediction sketch at each site. If the summary varies from the prediction sketch by more than a user defined tolerance amount, the summary and (possibly) a new prediction sketch is sent to a coordinator. The coordinator can use the information gathered from each site to continuously report frequent items. Sketches maintained by each site in this method require $O((1/\varepsilon^2)\log(1/\delta))$ space and $O(\log(1/\delta))$ time per update, where $\delta \in (0,1)$ is a probabilistic confidence.

Yi and Zhang [128] provided a deterministic method with communication $O((m/\varepsilon)\log N)$ and space usage $O(1/\varepsilon)$ at each site to continuously track ϕ-heavy hitters and ϕ-quantiles. In their method, every site and the coordinator have as many counters as the type of items plus one more counter for the total items. Every site keeps track of number of items it receives in each round. Once this number reaches roughly ε/m times of the total counter at the coordinator, the site sends the counter to the coordinator. After the coordinator receives m such messages, it updates its counters and broadcasts them to all sites. Sites reset their counter values and continue to next round. To lower space usage at sites, they suggested using space-saving sketch [97]. The authors also gave matching lower bounds on the communication costs for both problems, showing their algorithms are optimal in the deterministic setting.

Later, Huang *et al.* [78] proposed a randomized algorithm that uses $O(1/(\varepsilon\sqrt{m}))$ space at each site and $O((\sqrt{m}/\varepsilon)\log N)$ total communication and tracks heavy hitters in a distributed stream. For each item a in the stream a site chooses to send a message with a probability $p = \sqrt{m}/(\varepsilon\hat{n})$ where $\hat{n}$ is a 2-approximation of the total count. It then sends $f_e(A_j)$ the total count of messages at site j where $a = e$, to the coordinator. Again an approximation heavy-hitter count $\hat{f}_e(A_j)$ can be used at each site to reduce space. The ε-heavy hitters can be maintained from a random sampling of elements of size $s = O(1/\varepsilon^2)$. This allows one to use the well studied technique of maintaining a random sample of size s from a distributed stream [38, 116], which can be done with roughly $O((m+s)\log(N/s))$ communication.

In summary, there are four main protocols that have formal error guarantees and may be optimal for different restrictions on the problem:

(P1) Runs streaming algorithm on each site, and sends content of memory to C after enough items. This protocol has communication of $O((m/\varepsilon^2)\log\frac{N}{s})$, and is deterministic.

(P2) Each site sends update of f_e to C when local $(f_e - f_e^{\text{last-sent}}) > (\varepsilon/m)n$. This protocol has communication complexity of $O((m/\varepsilon)\log N)$ [128], and is deterministic.

(P3) Maintains a random sample of all items from A of size $O(1/\varepsilon^2)$ on C. This protocol has communication complexity of $O((m+\frac{1}{\varepsilon^2})\log\frac{N}{s})$ [38], and is randomized.

(P4) Each site sends f_e to C for each new $a_i = e$ with probability proportional to $\sqrt{m}/(\varepsilon n)$. This protocol has communication complexity of $O((\sqrt{m}/\varepsilon)\log N)$ [78], and is randomized.

2.5 Weighted Heavy Hitters in Distributed Streams

In this section, we address the problem of tracking weighted heavy hitters in distributed streams. The input is a *distributed* weighted data stream A, which is a sequence of tuples $(a_1, w_1), (a_2, w_2), \ldots, (a_n, w_n), \ldots$ where a_n is an element label and w_n is the weight. For any element $e \in [u]$, define $A_e = \{(a_i, w_i) \mid a_i = e\}$ and let $W_e = \sum_{(a_i, w_i)\in A_e} w_i$. For notational convenience, we sometimes refer to a tuple $(a_i, w_i) \in A$ by just its element a_i.

There are numerous important motivating scenarios for this extension. For example, instead of just monitoring counts of objects, we can measure a total size associated with an object, such as total number of bytes sent to an IP address, as opposed to just a count of packets.

2.5.1 Upper Bound on Weights

Let $W = \sum_{i=1}^{n} w_i$ be the total weight of the problem. However, allowing arbitrary weights can cause problems as demonstrated in the following example.

Suppose we want to maintain a 2-approximation of the total weight (i.e., a value $\hat{W}$ such that $\hat{W} \leq W \leq 2\hat{W}$). If the weight of each item doubles (i.e., $w_i = 2^i$ for tuple $(a_i, w_i) \in A$), every weight needs to be sent to the coordinator. This follows since W more than doubles with every item, so $\hat{W}$ cannot be valid for more than one step. The same issue arises in tracking approximate heavy hitters and matrices.

To make these problems well-posed, often researchers [80] assume weights vary in a finite range, and are then able to bound communication cost. To this end we assume all $w_i \in [1, \beta]$ for some constant $\beta \geq 1$. One option for dealing with weights is to just pretend every item with element e and weight w_i is actually a set of $\lceil w_i \rceil$ distinct items of element e and weight 1 (the last one needs to be handled carefully if w_i is not an integer). But this can increase the total communication and/or runtime of the algorithm by a factor β, and is not desirable.

Our proposed methods take great care to only increase the communication by a $\log(\beta N)/\log N$ factor compared to similar unweighted variants. In unweighted version, each protocol proceeds in $O(\log N)$ rounds (sometimes $O(\frac{1}{\varepsilon}\log N)$ rounds); a new round starts roughly when the total count n doubles. In our settings, the rounds will be based on the total weight W, and will change roughly when the total weight W doubles. Since the final weight $W \leq \beta N$, this will cause an increase to $O(\log W) = O(\log(\beta N))$ rounds.

We next describe how to extend four protocols for heavy hitters to the weighted setting. These are extensions of the unweighted protocols described in Section 2.4.

2.5.2 Estimating Total Weight

An important task is to approximate the current total weight $W = \sum_{i=1}^{n} w_i$ for all items across all sites. This is a special case of the heavy hitters problem where all items are treated as being the same element. So if we can show a result to estimate the weight of any single element using a protocol within εW, then we can get a global estimate $\hat{W}$ such that $|W - \hat{W}| \leq \varepsilon W$. All our subsequent protocols can run a separate process in parallel to return this estimate if they do not do so already.

Recall that the heavy hitter problem typically calls to return all elements $e \in [u]$ if $f_e(A)/W \geq \phi$, and never if $f_e(A)/W < \phi - \varepsilon$. For each protocol we study, the main goal is to ensure that an estimate $\hat{W}_e$ satisfies $|f_e(A) - \hat{W}_e| \leq \varepsilon W$. We show this, along with the $\hat{W}$ bound above, adjusting constants, is sufficient to estimate weighted heavy hitters. We return e as a ϕ-weighted heavy hitter if $\hat{W}_e/\hat{W} > \phi - \varepsilon/2$.

Lemma 2.5.1. *If $|f_e(A) - \hat{W}_e| \leq (\varepsilon/6)W$ and $|W - \hat{W}| \leq (\varepsilon/5)W$, we return e if and only if it is a valid weighted heavy hitter.*

Proof. We need $|\frac{\hat{W}_e}{\hat{W}} - \frac{f_e(A)}{W}| \leq \varepsilon/2$. We show the upper bound, the lower bound argument is symmetric.

$$\begin{aligned}\frac{\hat{W}_e}{\hat{W}} &\leq \frac{f_e(A)}{\hat{W}} + \frac{\varepsilon}{6}\frac{W}{\hat{W}} \leq \frac{f_e(A)}{W}\frac{1}{1-\varepsilon/5} + \frac{\varepsilon}{5}\frac{1+\varepsilon/5}{1-\varepsilon/5} \\ &\leq \frac{f_e(A)}{W} + \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{f_e(A)}{W} + \frac{\varepsilon}{2}.\end{aligned}$$

□

Given this result, we can focus just on approximating the frequency $f_e(A)$ of all items.

2.5.3 Weighted Heavy Hitters, Protocol 1

We start with an intuitive approach to the distributed streaming problem: run a streaming algorithm (for frequency estimation) on each site, and occasionally send the full summary on each site to the coordinator. We next formalize this protocol (P1).

On each site we run the Misha-Gries summary [99] for frequency estimation, modified to handle weights, with $2/\varepsilon = 1/\varepsilon'$ counters. We also keep track of the total weight W_i of all data seen on that site i since the last communication with the coordinator. When W_i reaches a threshold τ, site i sends all of its summaries (of size only $O(m/\varepsilon)$) to the coordinator. We set $\tau = (\varepsilon/2m)\hat{W}$, where $\hat{W}$ is an estimate of the total weight across all sites, provided by the coordinator. At this point the site resets its content to empty. This is summarized in Algorithm 1.

The coordinator can merge results from each site into a single summary without increasing its error bound, due to the mergeability of such summaries [9]. It broadcasts the updated total weight estimate $\hat{W}$ when it increases sufficiently since the last broadcast. See details in Algorithm 2.

Algorithm 1 P1: Tracking heavy-hitters (at site S_i)

```
for $(a_n, w_n)$ in round $j$ do
  Update $G_i \leftarrow \mathsf{MG}_{\varepsilon'}(G_i, (a_n, w_n))$.
  Update total weight on site $W_i \mathrel{+}= w_n$.
  if ($W_i \geq \tau = (\varepsilon/2m)\hat{W}$) then
    Send $(G_i, W_i)$ to coordinator; make $G_i, W_i$ empty.
```

Algorithm 2 P1: Tracking heavy-hitters (at C)

```
On input $(G_i, W_i)$:
Update sketch $S \leftarrow \mathsf{Merge}_{\varepsilon'}(S, G_i)$ and $W_C \mathrel{+}= W_i$.
if ($W_C/\hat{W} > 1 + \varepsilon/2$) then
  Update $\hat{W} \leftarrow W_C$, and broadcast $\hat{W}$ to all sites.
```

Lemma 2.5.2. *(P1) Algorithms 1 and 2 maintain that for any item $e \in [u]$ that $|f_e(S) - f_e(A)| \leq \varepsilon W_A$. The total communication cost is $O((m/\varepsilon^2)\log(\beta N))$ elements.*

Proof. For any item $e \in [u]$, the coordinator's summary S has error coming from two sources. First is the error as a result of merging all summaries sent by each site. By running these with an error parameter $\varepsilon' = \varepsilon/2$, we can guarantee [9] that this leads to at most $\varepsilon' W_C \leq \varepsilon W_A/2$, where W_C is the weight represented by all summaries sent to the coordinator, hence less than the total weight W_A.

The second source is all elements on the sites not yet sent to the coordinator. Since we guarantee that each site has total weight at most $\tau = (\varepsilon/2m)\hat{W} \leq (\varepsilon/2m)W$, then that is also an upper bound on the weight of any element on each site. Summing over all sites, we have that the total weight of any element not communicated to the coordinator is at most $m \cdot (\varepsilon/2m)W = (\varepsilon/2)W$.

Combining these two sources of error implies the total error on each element's count is *always* at most εW, as desired.

The total communication bound can be seen as follows. Each message takes $O(1/\varepsilon)$ space. The coordinator sends out a message to all m sites every (at most) m updates it sees from the coordinators; call this period an epoch. Thus each epoch uses $O(m/\varepsilon)$ communication. In each epoch, the size of W_C (and hence $\hat{W}$) increases by an additive $m \cdot (\varepsilon/2m)\hat{W} \geq (\varepsilon/4)W_A$, which is at least a relative factor $(1+\varepsilon/4)$. Thus starting from a weight of 1, there are k epochs until $1 \cdot (1+\varepsilon/4)^k \geq \beta N$, and thus $k = O(\frac{1}{\varepsilon}\log(\beta N))$. So after all k epochs the total communication is at most $O((m/\varepsilon^2)\log(\beta N))$. $\square$

2.5.4 Weighted Heavy-Hitters Protocol 2

Next we observe that we can significantly improve the communication cost of protocol P1 (above) using an observation, based on an unweighted frequency estimation protocol by Yi and Zhang [128]. Algorithms 3 and 4 summarize this protocol.

Each site takes an approach similar to Algorithm 1, except that when the weight threshold is reached, it does not send the entire summary it has, but only the weight at the site. It still needs to report heavy elements, so it also sends e whenever any element e's weight has increased by more than $(\varepsilon/m)\hat{W}$ since the last time information was sent for e. Note here it only sends that element, not all elements.

After the coordinator has received m messages, then the total weight constraint must have been violated. Since $W \leq \beta N$, at most $O(\log_{(1+\varepsilon)}(\beta N)) = O((1/\varepsilon)\log(\beta N))$ rounds are possible, and each round requires $O(m)$ total weight messages. It is a little trickier (but not too hard) to see it requires only a total of $O((m/\varepsilon)\log(\beta N))$ element messages, as follows from the next lemma; it is in general not true that there are $O(m)$ such messages in one round.

Algorithm 3 P2: Tracking heavy-hitters (at site S_i)

```
for each item $(a_n, w_n)$ do
    $W_i \mathrel{+}= w_n$ and $\Delta_{a_n} \mathrel{+}= w_n$.
    if ($W_i \geq (\varepsilon/m)\hat{W}$) then
        Send $(\text{total}, W_i)$ to $C$ and reset $W_i = 0$.
    if ($\Delta_{a_n} \geq (\varepsilon/m)\hat{W}$) then
        Send $(a_n, \Delta_{a_n})$ to $C$ and reset $\Delta_{a_n} = 0$.
```

Lemma 2.5.3. *After r rounds, at most $O(m \cdot r)$ element update messages have been sent.*

Algorithm 4 P2: Tracking heavy-hitters (at C)

```
On message $(\mathsf{total}, W_i)$:
Set $\hat{W}$ += $W_i$ and #msg += 1.
if (#msg $\geq m$) then
    Set #msg = 0 and broadcast $\hat{W}$ to all sites.
On message $(a_n, \Delta_n)$: set $\hat{W}_{a_n}$ += $\Delta_{a_n}$.
```

Proof. We prove this inductively. Each round gets a budget of m messages, but only uses t_i messages in round i. We maintain a value $T_r = r \cdot m - \sum_{i=1}^{r} t_i$. We show inductively that $T_r \geq 0$ at all times.

The base case is clear, since there are at most m messages in round 1, so $t_1 \leq m$, thus $T_1 = m - t_1 \geq 0$. Then since it takes less than 1 message in round i to account for the weight of a message in a round $i' < i$. Thus, if $\sum_{i=1}^{r-1} t_i = n_r$, so $k_r = (r-1)m - n_r$, then if round i had more than $m + k_r$ messages, the coordinator would have weight larger than having m messages from each round, and it would have at some earlier point ended round r. Thus this cannot happen, and the inductive case is proved. $\square$

The error bounds follow directly from the unweighted case from [128], and is similar to that for (P1). We can thus state the following theorem.

Theorem 2.5.1. *Protocol 2 (P2) sends $O(\frac{m}{\varepsilon}\log(\beta N))$ total messages, and approximates all frequencies within εW.*

One can use the space-saving algorithm [97] to reduce the space on each site to $O(m/\varepsilon)$, and the space on the coordinator to $O(1/\varepsilon)$.

2.5.5 Weighted Heavy-Hitters Protocol 3

The next protocol, labeled (P3), simply samples elements to send to the coordinator, proportional to their weight. Specifically we combine ideas from priority sampling [57] for without replacement weighted sampling, and distributed sampling on unweighted elements [38]. In total we maintain a random sample S of size at least $s = O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ on the coordinator, where the elements are chosen *proportional to their weights*, unless the weights are large enough (say greater than W/s), in which case they are always chosen. By deterministically sending all large enough weighted elements, not only do we reduce the variance of the approach, but it also means the protocol naturally sends the full dataset if the desired sample size s is large enough, such as at the beginning of the stream. Algorithm 5 and Algorithm 6 summarize the protocol. We denote total weight of sample by W_S. On receiving a pair (a_n, w_n), a site generates a random number $r_n \in \mathsf{Unif}(0,1)$ and assigns a priority $\rho_n = w_n/r_n$ to a_n. Then the site sends triple (a_n, w_n, ρ_n) to the coordinator if $\rho_n \geq \tau$, where τ is a global threshold provided by the coordinator.

Initially τ is 1, so sites simply send any items they receive to the coordinator. At the beginning of further rounds, the coordinator doubles τ and broadcasts it to all sites. Therefore at round j, $\tau = \tau_j = 2^j$. In any round j, the coordinator maintains two priority queues Q_j and Q_{j+1}. On receiving a new tuple (a_n, w_n, ρ_n) sent by a site, the coordinator places it into Q_{j+1} if $\rho_n \geq 2\tau$, otherwise it places a_n into Q_j.

Once $|Q_{j+1}| = s$, the round ends. At this time, the coordinator doubles τ as $\tau = \tau_{j+1} = 2\tau_j$ and broadcasts it to all sites. Then it discards Q_j and examines each item (a_n, w_n, ρ_n) in Q_{j+1}, if $\rho_n \geq 2\tau$, it goes into Q_{j+2}, otherwise it remains in Q_{j+1}.

Algorithm 5 P3: Tracking heavy-hitters (at site S_i)

```
for $(a_n, w_n)$ in round $j$ do
  choose $r_n \in \mathsf{Unif}(0,1)$ and set $\rho_n = w_n/r_n$.
  if $\rho_n \geq \tau$ then send $(a_n, w_n, \rho_n)$ to $C$.
```

Algorithm 6 P3: Tracking heavy-hitters (at C)

```
On input of $(a_n, w_n, \rho_n)$ from any site in round $j$:
if $\rho > 2\tau_j$ then put $a_n$ in $Q_{j+1}$,
        else put $a_n$ in $Q_j$.
if $|Q_{j+1}| \geq s$ then
  Set $\tau_{j+1} = 2\tau_j$; broadcast $\tau_{j+1}$ to all sites.
  for $(a_n, w_n, \rho_n) \in Q_{j+1}$ do
    if $\rho_n > 2\tau_{j+1}$, put $a_n$ in $Q_{j+2}$.
```

At any time, a sample of size exactly s can be derived by subsampling from $Q_j \cup Q_{j+1}$. But it is preferable to use a larger sample $S = Q_j \cup Q_{j+1}$ to estimate properties of A, so we always use this full sample.

2.5.5.1 Communication Analysis

The number of messages sent to the coordinator in each round is $O(s)$ with high probability. To see that, consider an arbitrary round j. Any item a_n being sent to coordinator at this round, has $\rho_n \geq \tau$. This item will be added to Q_{j+1} with probability

$$\mathbf{Pr}(\rho_n \geq 2\tau \mid \rho_n \geq \tau) = \frac{\mathbf{Pr}(\rho_n \geq 2\tau)}{\mathbf{Pr}(\rho_n \geq \tau)} = \frac{\mathbf{Pr}(r_n \leq \frac{w_n}{2\tau})}{\mathbf{Pr}(r_n \leq \frac{w_n}{\tau})} = \frac{\min(1, \frac{w_n}{2\tau})}{\min(1, \frac{w_n}{\tau})} \geq \frac{1}{2}.$$

Thus sending $4s$ items to coordinator, the expected number of items in Q_{j+1} would be greater than or equal to $2s$. Using a Chernoff-Hoeffding bound $\mathbf{Pr}(2s - |Q_{j+1}| > s) \leq \exp(-2s^2/4s) = \exp(-s/2)$. So if in each round $4s$ items are sent to coordinator, with high probability (at least $1 - \exp(-s/2)$), there would be s elements in Q_{j+1}. Hence each round has $O(s)$ items sent with high probability.

The next lemma bounds the number of rounds. Intuitively, each round requires the total weight of the stream to double, starting at weight s, and this can happen $O(\log(\beta N/s))$ times.

Lemma 2.5.4. *The number of rounds is at most $O(\log(\beta N/s))$ with probability at least $1 - e^{-\Omega(s)}$.*

Proof. In order to reach round j we must have s items with priority $\rho_n > \tau_j = 2^j$; this happens with probability $\min(1, w_n/2^j) \leq \min(1, \beta/2^j) \leq \beta/2^j$. We assume for now $\beta \leq 2^j$ for values of j we consider; the other case is addressed at the end of the proof.

Let $X_{n,j}$ be a random variable that is 1 if $\rho_n \geq \tau_j$ and 0 otherwise. Let $M_j = \sum_{n=1}^{N} X_{n,j}$. Thus the expected number of items that have a priority greater than τ_j is $\mathrm{E}[M_j] = \sum_{n=1}^{N} w_n/2^j \leq N\beta/2^j$. Setting $j_N = \lceil \log_2(\beta N/s) \rceil$ then $\mathrm{E}[M_{j_N}] \leq s$. We now want to use the following Chernoff bound on the N independent random variables $X_{n,j}$ with $M = \sum_{n=1}^{N} X_{n,j}$ that bounds $\mathbf{Pr}[M_j \geq (1+\alpha)\,\mathrm{E}[M_j]] \leq \exp(-\alpha^2\,\mathrm{E}[M_j]/(2+\alpha))$.

Note that $\mathrm{E}[M_{j_N+1}] \leq (N\beta)/2^{j_N+1} \leq s/2$. Then setting $\alpha = 1$ ensures that

$$(1+\alpha)\,\mathrm{E}[M_{j_N+1}] = 2 \cdot (s/2) \leq s.$$

Thus we can solve

$$\mathbf{Pr}[M_{j_N+1} \geq s] \leq \exp\left(-\frac{s/2}{3}\right) = \exp\left(-\frac{-s}{6}\right).$$

Thus since in order to reach round $j_N + 1 = \log_2(\beta N/s) + 1 = O(\log(\beta N/s))$, we need s items to have priority greater than τ_{j_N+1}, this happens with probability at most $e^{-\Omega(s)}$.

Recall that to be able to ignore the case where $w_n > \tau_{j_N+1}$ we assumed that $\beta < \tau_{j_N+1}$. If this were not true, then $\beta > 2^{\log_2(\beta N/s)+1} = 2\beta N/s$ implies that $s > N$, in which case we would send all elements before the end of the first round, and the number of rounds is 1. $\square$

Since with probability at least $1 - e^{-\Omega(s)}$, in each round the coordinator receives $O(s)$ messages from all sites and broadcasts the threshold to all m sites, we can then combine with Lemma 2.5.4 to bound the total messages.

Lemma 2.5.5. *This protocol sends $O((m+s)\log\frac{\beta N}{s})$ messages with probability at least $1 - e^{-\Omega(s)}$. We set $s = \Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$.*

Note that each site only requires $O(1)$ space to store the threshold, and the coordinator only requires $O(s)$ space.

2.5.5.2 Creating Estimates

To estimate $f_e(A)$ at the coordinator, we use a set $S' = Q_j \cup Q_{j+1}$ which is of size $|S'| = s' > s$. Let $\hat{\rho}$ be the priority of the smallest priority element in S'. Let S be all elements in S' except for this single smallest priority element. For each of the $s' - 1$ elements in S assign them a weight $\bar{w}_i = \max(w_i, \hat{\rho})$, and we set $W_S = \sum_{a_i \in S} \bar{w}_i$. Then via known priority sampling results [57, 115], it follows that $\mathrm{E}[W_S] = W_A$ and that $(1-\varepsilon)W_A \le W_S \le (1+\varepsilon)W_A$ with large probability (say with probability $1 - \varepsilon^2$, based on variance bound $\mathbf{Var}[W_S] \le W_A^2/(s'-2)$ [115] and a Chebyshev bound). Define $S_e = \{a_n \in S \mid a_n = e\}$ and $f_e(S) = \sum_{a_n \in S_e} \bar{w}_n$.

The following lemma shows that the sample maintained at the coordinator gives a good estimate on item frequencies. At a high-level, we use a special Chernoff-Hoeffding bound for negatively correlated random variables [104] (since the samples are without replacement), and then only need to consider the points selected that have small weights, and thus have values in $\{0, \hat{\rho}\}$.

Lemma 2.5.6. *With $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$, the coordinator can use the estimate from the* *sample S such that, with large probability, for each item $e \in [u]$, $|f_e(S) - f_e(A)| \le \varepsilon W_A$.*

Proof. To prove our claim, we use the following Chernoff-Hoeffding bound [104]. Given a set of negatively-correlated random variables $Y_1, \ldots, Y_r$ and $Y = \sum_{n=1}^{r} Y_n$, where each $Y_n \in [a_i, b_i]$, where $\Delta = \max_n (b_n - a_n)$ then $\mathbf{Pr}[|Y - \mathrm{E}[Y]| \ge \alpha] \le \exp(-2\alpha^2 / r\Delta^2)$.

For a given item $e \in [u]$ and any pair $(a_n, w_n) \in S$, we define a random variable[2] $X_{n,e}$ as follows:

$$X_{n,e} = \bar{w}_n \text{ if } a_n = e,\ 0 \text{ otherwise.}$$

Define a heavy set $H = \{a_n \in A \mid w_n \ge \tau_j\}$, these items are included in S deterministically in round j. Let the light set be defined $L = A \setminus H$. Note that for each $a_n \in H$ that $X_{n,e}$ is deterministic, given e. For all $a_n \in S \cap L$, then $X_{n,e} \in [0, 2\tau_j]$ and hence using these as random variables in the Chernoff-Hoeffding bound, we can set $\Delta = 2\tau_j$.

Define $M_e = \sum_{a_n \in S} X_{n,e}$, and note that $f_e(S) = M_e$ is the estimate from S' of $f_e(A)$. Let $W_L = \sum_{a_n \in L} w_i$. Since all light elements are chosen with probability proportionally to their weight, then given an $X_{n,e}$ for $a_i \in S \cap L$ it has label e with probability $f_e(L)/W_L$. And in general $\mathrm{E}[\sum_{a_n \in S \cap L} \bar{w}_n] = \mathrm{E}[W_{S \cap L}] = W_L$. Let $H_e = \{a_n \in H \mid a_n = e\}$. Now we can see

$$\mathrm{E}[f_e(S)] = \mathrm{E}\left[\sum_{n=1}^{|S|} X_{n,e}\right] = \sum_{a_n \in H_e} w_n + \mathrm{E}\left[\sum_{a_n \in S \cap L} \tau_j\right] \cdot \frac{f_e(L)}{W_L} = f_e(H) + W_L \cdot \frac{f_e(L)}{W_L} = f_e(A).$$

Now we can apply the Chernoff-Hoeffding bound.

$$\begin{aligned}\mathbf{Pr}\left(|f_e(S) - f_e(A)| > \varepsilon W_A\right) &= \mathbf{Pr}\left(|M_e - E[M_e]| > \varepsilon W_A\right) \\ &\le \exp\left(\frac{-2\varepsilon^2 W_A^2}{|S| 4\tau_j^2}\right) \le \exp\left(-\frac{1}{2}\varepsilon^2 |S| / (1+\varepsilon)^2\right) \le \delta,\end{aligned}$$

where the last line follows since $(1+\varepsilon) W_A \ge \sum_{a_n \in S} \bar{w}_n \ge |S| \tau_j$, where the first inequality holds with high probability on $|S|$. Solving for $|S|$ yields $|S| \ge \frac{(1+\varepsilon)^2}{2\varepsilon^2} \ln(1/\delta)$. Setting $\delta = O(\varepsilon^2 / \log(1/\varepsilon)) = 1/|S|$ allows the result to hold with probability at least $1 - 1/|S|$. $\square$

Theorem 2.5.2. *Protocol 3 (P3) sends $O((m+s)\log\frac{\beta N}{s})$ messages with large probability; It gets a set S of size $s = \Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ so that $|f_e(S) - f_e(A)| \le \varepsilon W$.*

[2] Note that the random variables $X_{n,e}$ are negatively-correlated, and not independent, since they are derived from a sample set S drawn without replacement. Thus we appeal to the special extension of the Chernoff-Hoeffding bound by Panconesi and Srinivasan [104] that handles this case. We could of course use with-replacement sampling, but these algorithms require more communication and typically provide worse bounds in practice, as demonstrated.

2.5.5.3 Sampling With Replacement

We can show similar results on s samples *with replacement*, using s independent samplers. In round j, for each element (a_n, w_n) arriving at a local site, the site generates s independent r_n values, and thus s priorities ρ_n. If any of them is larger than τ_j, then the site forwards it to coordinator, along with the index (or indices) of success.

For each of s independent samplers, say for sampler $t \in [s]$, the coordinator maintains the top 2 priorities $\rho_t^{(1)}$ and $\rho_t^{(2)}$, $\rho_t^{(1)} > \rho_t^{(2)}$, among all it received. It also keeps the element information a_t associated with $\rho^{(1)}$. For the sampler $i \in [s]$, the coordinator keeps a weight $\bar{w}_i = \rho_i^{(2)}$. One can show that $\mathrm{E}[\bar{w}_i] = W$, the total weight of the entire stream [57]. We improve the global estimate as $\hat{W} = (1/s)\sum_{i=1}^{s} \bar{w}_i$, and then assign each element a_i the same weight $\hat{w}_i = \hat{W}/s$. Now $\mathrm{E}[\sum_{i=1}^{s} \hat{w}_i] = W$, and each a_i is an independent sample (with replacement) chosen proportional to its weight. Then setting $s = O((1/\varepsilon^2)\log(1/\varepsilon))$ it is known that these samples can be used to estimate all heavy hitters within εW with probability at least $1 - e^{-\Omega(s)}$.

The jth round terminates when the $\rho_i^{(2)}$ for all i is larger than $2\tau_j$. At this point, coordinator sets $\tau_{j+1} = 2\tau_j$, informs all sites of the new threshold and begins the $(j+1)$th round.

2.5.5.4 Communication Analysis

Since this protocol is an adaptation of existing results [38], its communication is $O((m + s\log s)\log(\beta N) = O((m + \frac{1}{\varepsilon^2}\log^2\frac{1}{\varepsilon})\log(\beta N))$ messages. This result doesn't improve the error bounds or communication bounds with respect to the without replacement sampler described above, as is confirmed in Section 2.5.7. Also in terms of running time (without parallelism at each site), sampling *without replacement* will be better.

2.5.6 Weighted Heavy-Hitters Protocol 4

This protocol is inspired by the unweighted case from Huang *et al.* [78]. Each site maintains an estimate of the total weight $\hat{W}$ that is provided by the coordinator and always satisfies $\hat{W} \leq W \leq 2\hat{W}$, with high probability. It then sets a probability $p = 2\sqrt{m}/(\varepsilon\hat{W})$. Now given a new element (a, w) with some probability $\bar{p}$, it sends to the coordinator $(e, \bar{w}_{e,j} = f_e(A_j))$ for $a = e \in [u]$; this is the total weight of all items in its stream that equal element e. Finally the coordinator needs to adjust each $\bar{w}_{e,j}$ by adding $1/p - 1$ (for elements that have been witnessed) since that is the expected number of items with element e in the stream until the next update for e.

If w is an integer, then one option is to pretend it is actually w distinct elements with weight 1. For each of the w elements we create a random variable Z_i that is 1 with probability p and 0 otherwise. If *any* $Z_i = 1$, then we send $f_e(A_j)$. However this is inefficient (say if $w = \beta = 1000$), and only works with integer weights.

Instead we notice that at least one Z_i is 1 if none are 0, with probability $1 - (1-p)^w \approx 1 - e^{-pw}$. So in the integer case, we can set $\bar{p} = 1 - (1-p)^w$, and then since we send a more accurate estimate of f_e (as it essentially comes later in the stream) we can directly apply the analysis from Huang *et al.* [78]. To deal with non integer weights, we set $\bar{p} = 1 - e^{-pw}$, and describe the approach formally on a site in Algorithm 7.

Notice that the probability of sending an item is asymptotically the same in the case that $w = 1$, and it is smaller otherwise (since we send at most one update $\bar{w}_{e,j}$ per batch). Hence the communication bound is asymptotically the same, except for the number of rounds. Since the weight is broadcast to the sites from the coordinator whenever it doubles, and now the total weight can be βN instead of N, the number of rounds is $O(\log(\beta N))$ and the total communication is $O((\sqrt{m}/\varepsilon)\log(\beta N))$ with high probability.

Algorithm 7 P4: Tracking of heavy-hitters (at site S_j)

```
Given weight $\hat{W}$ from $C$, set $p = 2\sqrt{m}/(\varepsilon\hat{W})$.
for each item $(a, w)$ it receives do
    For $a = e$ update $f_e(A_j) := f_e(A_j) + w$.
    Set $\bar{p} = 1 - e^{-pw}$.
    With probability $\bar{p}$ send $\bar{w}_{e,j} = f_e(A_j)$ to $C$.
```

When the coordinator is sent an estimate $\bar{w}_{e,j}$ of the total weight of element e at site j, it needs to update this estimate slightly as in Huang *et al.*, so that it has the right expected value. It sets $\hat{w}_{e,j} = \bar{w}_{e,j} + 1/p$, where again $p = 2\sqrt{m}/(\varepsilon\hat{W})$; $\hat{w}_{e,j} = 0$ if no such messages are sent. The coordinator then estimates each $f_e(A)$ as $\hat{W}_e = \sum_{j=1}^{m} \hat{w}_e$.

We first provide intuition how the analysis works, if we used $\bar{p} = 1 - (1-p)^w$ (i.e., $\approx 1 - e^{-pw}$) and w is an integer. In this case, we can consider simulating the process with w items of weight 1; then it is identical to the unweighted algorithm, except we always send $\bar{w}_{e,j}$ at then end of the batch of w items. This means the expected number until the next update is still $1/p - 1$, and the variance of $1/p^2$ and error bounds of Huang *et al.* [78] still hold.

Lemma 2.5.7. *The above protocol guarantees that* $|f_e(A) - \hat{W}_e| \leq \varepsilon W$ *on the coordinator, with probability at least* 0.75.

Proof. Consider a value of k large enough so that $w \cdot 10^k$ is always an integer (i.e., the precision of w in a system is at most k digits past decimal point). Then we can hypothetically simulate the unweighted case using $w_k = w \cdot 10^k$ points. Since now $\hat{W}$ represents 10^k times as many unweighted elements, we have $p_k = p/10^k = \sqrt{m}/(\varepsilon \hat{W} 10^k)$. This means the probability we send an update should be $1 - (1 - p_k)^{w_k}$ in this setting.

Now use that for any x that $\lim_{n \to \infty}(1 - \frac{x}{n})^n = e^{-x}$. Thus setting $n = w_k$ and $x = p_k \cdot w_k = (p/10^k)(w10^k) = pw$ we have $\lim_{k \to \infty} 1 - (1 - p_k)^{w_k} = 1 - e^{-pw}$.

Next we need to see how this simulated process affects the error on the coordinator. Using results from Huang *et al.* [78], where they send an estimate $\bar{w}_{e,j}$, the expected value $\mathrm{E}[\bar{w}_{e,j}] = f_e(A_j) - 1/p + 1$ at any point afterwards where that was the last update. This estimates the count of weight 1 objects, so in the case where they are weight 10^{-k} objects the estimate of $f_e(A_j)^{(k)} = f_e(A_j)10^k$ is using $\bar{w}_{e,j}^{(k)} = \bar{w}_{e,j}10^k$. Then, in the limiting case (as $k \to \infty$), we adjust the weights as follows.

$$\begin{aligned}\mathrm{E}[\bar{w}_{e,j}] &= \mathrm{E}[\bar{w}_{e,j}^{(k)}]10^{-k} = (f_e(A_j)^{(k)} - 1/p_k + 1) \cdot 10^{-k} \\ &= (f_e(A_j)10^k - \frac{10^k}{p} + 1)10^{-k} = f_e(A_j) - \frac{1}{p} + 10^{-k},\end{aligned}$$

so as $\lim_{k \to \infty} \mathrm{E}[\bar{w}_{e,j}] = f_e(A_j) - 1/p$. So our procedure has the right expected value. Furthermore, it also follows that the variance is still $1/p^2$, and thus the error bound from [78] that any $|f_e(A) - \hat{W}_e| \leq \varepsilon W$ with probability at least 0.75 still holds. □

Theorem 2.5.3. *Protocol 4 (P4) sends* $O(\frac{\sqrt{m}}{\varepsilon}\log(\beta N))$ *total messages and with probability* 0.75 *has* $|f_e(A) - \hat{W}_e| \leq \varepsilon W$.

The bound can be made to hold with probability $1 - \delta$ by running $\log(2/\delta)$ copies and taking the median. The space on each site can be reduced to $O(1/\varepsilon)$ by using a weighted variant of the space-saving algorithm [97]; the space on the coordinator can be made $O(m/\varepsilon)$ by just keeping weights for which $\bar{w}_{i,e} \geq 2\varepsilon\hat{W}_j$, where $\hat{W}_j$ is a 2-approximation of the weight on site j.

2.5.7 Experimental Evaluation

In this section, we describe our experiments, and how our different protocols compare.

2.5.7.1 Datasets

For tracking the distributed weighted heavy hitters, we generated data from Zipfian distribution, and set the skew parameter to 2 in order to get meaningful distributions that produce some heavy hitters per run. The generated dataset contained 10^7 points, in order to assign them weights we fixed the upper bound (default $\beta = 1,000$) and assigned each point a uniform random weight in range $[1, \beta]$. Weights are not necessarily integers.

2.5.7.2 Metrics

The efficiency and accuracy of the weighted heavy hitters protocols are controlled with input parameter ε specifying desired error tolerance. We compare them on:

- *Recall:* The number of true heavy hitters returned by a protocol over the correct number of true heavy hitters.
- *Precision:* The number of true heavy hitters returned by a protocol over the total number of heavy hitters returned by the protocol.
- *err*: Average relative error of the frequencies of the true heavy hitters returned by a protocol.
- *msg*: Number of messages sent during a protocol.

We observed that both the approximation errors and communication costs of all methods *are very stable with respect to query time*, by executing estimations at the coordinator at randomly selected time instances. Hence, we only report the average err from queries in the very end of the stream (i.e., results of our methods on really large streams).

2.5.7.3 Distributed Weighted Heavy Hitters

We denote four protocols for tracking distributed weighted heavy hitters as P1, P2, P3 and P4, respectively. As a baseline, we could send all 10^7 stream elements to the coordinator. This would have no error. All of our heavy hitters protocols return an element e as heavy hitter only if $\hat{W}_e/\hat{W} \geq \phi - \varepsilon/2$ while the exact weighted heavy hitter method which our protocols are compared against, returns e as heavy hitter if $f_e(A)/W \geq \phi$.

We set the heavy-hitter threshold ϕ to 0.05 and we varied error guarantee ε in the range $\{5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$. When the plots do not vary ε, we use the default value of $\varepsilon = 10^{-3}$. Also we varied number of sites (m) from 10 to 100, otherwise we have as default $m = 50$.

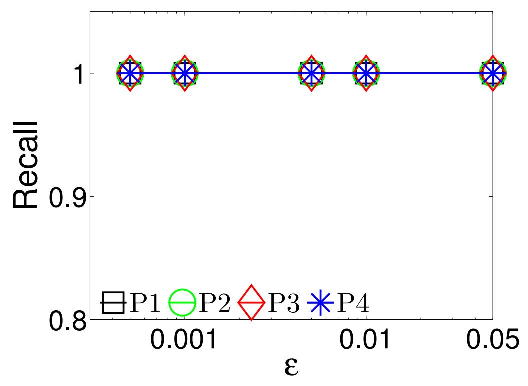
All four algorithms prove to be highly effective in estimating weighted heavy hitters accurately, as shown in *recall* (Figure 2.1(a)) and *precision* (Figure 2.1(b)) plots. In particular, the recall values for all algorithms are constant 1.0.

Note that precision values dip, but this is because the true heavy hitters have $f_e(A)/W$ above ϕ where our algorithms only return a value if $\hat{W}_e/\hat{W} \geq \phi - \varepsilon/2$, so they return more false positives as ε increases. For ε smaller than 0.01, all protocols have a precision of 1.0.

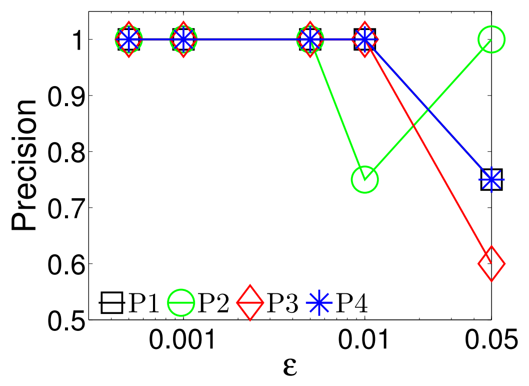
When measuring (the measured) *err* as seen in Figure 2.1(c), our protocols consistently outperform the error parameter ε. The only exception is P4, which has slightly larger error than predicted for very small ε; recall this algorithm is randomized and has a constant probability of failure. P1 has almost no error for $\varepsilon = 0.01$ and below; this can be explained by improved analysis for Misra-Gries [18] on skewed data, which applies to our Zipfian data. Protocols P2 and P3 also greatly underperform their guaranteed error.

The protocols are quite communication efficient, saving several orders of magnitude in communication as shown in Figure 2.1(d). For instance, all protocols use roughly 10^5 messages at $\varepsilon = 0.01$ out of 10^7 total stream elements. To further understand different protocols, we tried to compare them by making them using (roughly) the same number of messages. This is achieved by using *different* ε values. As shown in Figure 2.1(e), all protocols achieved excellent approximation quality, and the measured error drops quickly as we allocate more budget for the number of messages. In particular, P2 is the best if fewer than 10^5 messages are acceptable with P3 also shown to be quite effective. P1 performs best if 10^6 messages are acceptable.

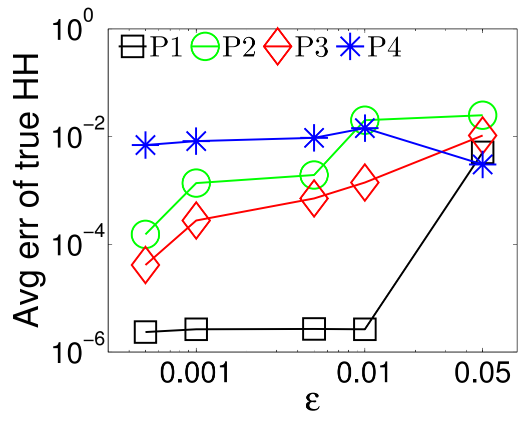
In another experiment, we tuned all protocols to obtain (roughly) the same measured error of *err* = 0.1 to compare their communication cost versus the upper bound on the element weights (β). Figure 2.1(f) shows that they are all robust to the parameter β; P2 or



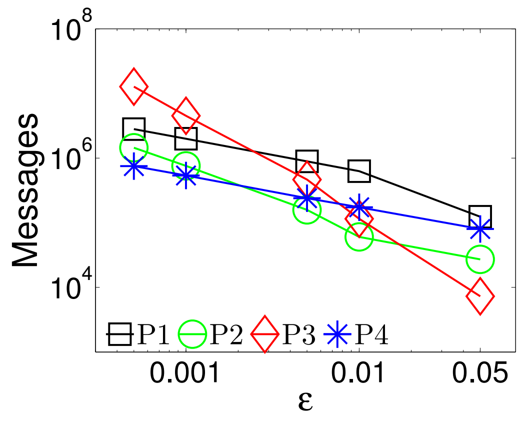
(a) recall vs. ε



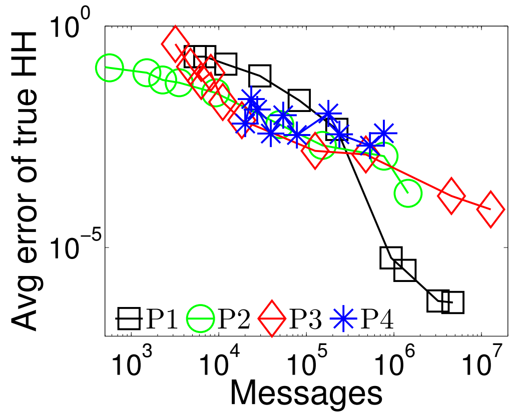
(b) precision vs. ε



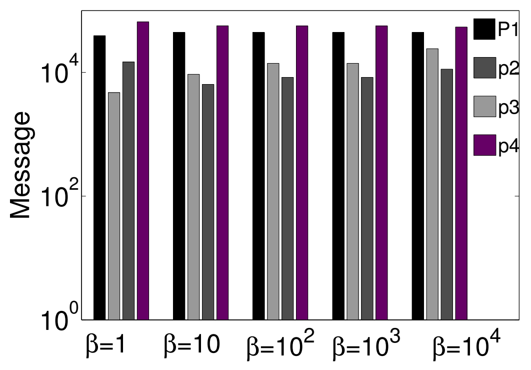
(c) err vs. ε



(d) msg vs. ε



(e) err vs. msg



(f) msg vs. β

Figure 2.1: Results for distributed weighted heavy hitters protocols on Zipfian distribution with skew=2.

P3 performs the best.

CHAPTER 3

FREQUENT DIRECTIONS ALGORITHM

In this chapter, we describe FrequentDirections algorithm [92] (short for FD) initially introduced by Edo Liberty. FrequentDirections is a deterministic and iterative matrix sketching method that works over a stream. FrequentDirections draws on the similarity between matrix sketching problem and item frequency estimation problem, and is directly inspired by MG sketch which was described in Chapter 2.

3.1 Frequent Directions, Intuition and Algorithm

The intuition behind FD is very similar to that of FrequentItems. In the same way that FrequentItems periodically deletes ℓ different elements, FD periodically 'shrinks' ℓ orthogonal vectors by roughly the same amount. This means that during shrinking steps, the squared Frobenius norm of the sketch reduces ℓ times faster than its squared projection on any single direction. Since the Frobenius norm of the final sketch is non negative, we are guaranteed that no direction in space is reduced by "too much". As a remark, when presented with an item indicator matrix, FD exactly mimics a variant of FrequentItems.

More explicitly, the algorithm keeps an $\ell \times d$ sketch matrix B that is updated every time a new row from the input matrix A is added. The algorithm maintains the invariant that the last row of sketch B is always all-zero valued. During the execution of the algorithm, rows from A simply replace the all-zero valued row in B. Once B is full, we compute the full singular value decomposition (SVD) of B and nullify its last row in a two-stage process. First, the sketch B is rotated (from left) using its SVD such that its rows are orthogonal and in descending magnitude order. Then, the norm of sketch rows are "shrunk" so that at least one of them is set to zero.

Algorithm 8 FD

```
Input: $\ell, A \in R^{n \times d}$
$B \leftarrow 0^{\ell \times d}$
for $i \in 1, \ldots, n$ do
    $B_\ell \leftarrow a_i$                                # $i$th row of $A$ replaces (all-zeros) $\ell$th row of $B$
    $[U, \Sigma, V] \leftarrow \mathrm{SVD}(B)$
    $\delta \leftarrow \sigma_\ell^2$
    $B \leftarrow \sqrt{\Sigma^2 - \delta I_\ell} \cdot V^T$              # The last row of $B$ is again zero
return $B$
```

3.1.1 Error Bounds Analysis

Here we state and prove two different error bounds for Algorithm 8 which is our simplest and most space efficient algorithm. The first error bound provides an additive approximation guarantee:

Theorem 3.1.1. *Let $B \in \mathbb{R}^{\ell \times d}$ be the sketch of FD on an input matrix $A \in \mathbb{R}^{n \times d}$. Then for any unit vector $x \in \mathbb{R}^d$ it holds that*

$$0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A - A_k\|_F^2 / (\ell - k).$$

Or equivalently

$$\|A^T A - B^T B\|_2 \leq \|A - A_k\|_F^2 / (\ell - k) \quad \text{and} \quad A^T A \succeq B^T B.$$

This holds for all $k < \ell$ including $k = 0$ where we define A_0 as the $n \times d$ all zeros matrix. Note that setting $\ell = \lceil 1/\varepsilon + k \rceil$ yields error of $\varepsilon \|A - A_k\|_F^2$ using $O(d\ell) = O(dk + d/\varepsilon)$ space.

The second error bound offers a stronger multiplicative bound which requires projecting A onto the sketch computed by FD:

Theorem 3.1.2. *Let $B \in \mathbb{R}^{\ell \times d}$ be the sketch FD produces on an input matrix $A \in \mathbb{R}^{n \times d}$. Then for any $k < \ell$,*

$$\|A - \pi_B^k(A)\|_F^2 \leq (1 + \frac{k}{\ell - k}) \|A - A_k\|_F^2,$$

where $\pi_B^k(A)$ denotes the projection of A onto the best rank k approximation of B. Note that by setting $\ell = \lceil k + k/\varepsilon \rceil$ FD achieves the standard error bound of form $\|A - \pi_B^k(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$ using only $O(d\ell) = O(dk/\varepsilon)$ space.

Below we provide the proofs for the above theorems. The reader will notice that we occasionally use inequalities instead of equalities at different parts of the proof to obtain three Properties. This is not unintentional. The reason is that we want the same exact proofs to hold also for Algorithm 9 which is described in Section 3.1.2. Algorithm 9 is conceptually identical to Algorithm 8. It requires only twice as much space but is asymptotically faster by a factor of ℓ. Moreover, *any algorithm* that produces an approximate matrix B which satisfies the following properties (for any choice of Δ) achieves the error bounds stated in Theorem 3.1.1 and Theorem 3.1.2.

In what follows, let $C \leftarrow \Sigma V^T$ in every iteration of Algorithm 9. In addition, we denote by δ_i, $B_{[i]}$, $C_{[i]}$ the values of δ, B and C, respectively, *after* the ith row of A was processed. We set $\Delta = \sum_{i=1}^{n} \delta_i$ as the total mass we subtract from the stream during our algorithm. To prove our result we first prove three auxiliary properties.

Property 3.1.1. *For any vector x we have $\|Ax\|^2 - \|Bx\|^2 \geq 0$.*

Proof. Use the observations that $\langle a_i, x\rangle^2 + \|B_{[i-1]}x\|^2 = \|C_{[i]}x\|^2$.

$$\|Ax\|^2 - \|Bx\|^2 = \sum_{i=1}^{n}[\langle a_i, x\rangle^2 + \|B_{[i-1]}x\|^2 - \|B_{[i]}x\|^2] \geq \sum_{i=1}^{n}[\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2] \geq 0. \qquad \square$$

Property 3.1.2. *For any unit vector $x \in \mathbb{R}^d$ we have $\|Ax\|^2 - \|Bx\|^2 \leq \Delta$.*

Proof. To see this, first note that $\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 \leq \|C_{[i]}^T C_{[i]} - B_{[i]}^T B_{[i]}\| \leq \delta_i$. Now, consider the fact that $\|C_{[i]}x\|^2 = \|B_{[i-1]}x\|^2 + \langle a_i, x\rangle^2$. Substituting for $\|C_{[i]}x\|^2$ above and taking the sum yields

$$\begin{aligned}\sum_i \|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 &= \sum_i(\|B_{[i-1]}x\|^2 + \langle a_i, x\rangle^2) - \|B_{[i]}x\|^2 \\ &= \|Ax\|^2 + \|B_{[0]}x\|^2 - \|B_{[n]}x\|^2 = \|Ax\|^2 - \|Bx\|^2.\end{aligned}$$

Combining this with $\sum_i \|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 \leq \sum_i \delta_i = \Delta$ yields that $\|Ax\|^2 - \|Bx\|^2 \leq \Delta$. $\square$

Property 3.1.3. $\Delta\ell \leq \|A\|_F^2 - \|B\|_F^2$.

Proof. In the ith round of the algorithm $\|C_{[i]}\|_F^2 \geq \|B_{[i]}\|_F^2 + \ell\delta_i$ and $\|C_{[i]}\|_F^2 = \|B_{[i-1]}\|_F^2 + \|a_i\|^2$. By solving for $\|a_i\|^2$ and summing over i we get

$$\|A\|_F^2 = \sum_{i=1}^{n} \|a_i\|^2 \geq \sum_{i=1}^{n} \|B_{[i]}\|_F^2 - \|B_{[i-1]}\|_F^2 + \ell\delta_i = \|B\|_F^2 + \ell\Delta. \qquad \square$$

Equipped with the above three properties, and *no additional requirements about the construction of B*, we can prove Theorem 3.1.1:

Proof of Theorem 3.1.1. Let $x \in \mathbb{R}^d$ be an arbitrary unit vector. First note that

$$\|Ax\|^2 - \|Bx\|^2 = x^T A^T Ax - x^T B^T Bx = x^T(A^T A - B^T B)x.$$

Since Property 3.1.2 holds for any unit vector $x \in \mathbb{R}^d$, it does so for $x^* = \text{argmax}_{x \in \mathbb{R}^d, \|x\|=1} x^T(A^T A - B^T B)x$ which is the first eigenvector of $A^T A - B^T B$. Therefore

$$\|A^T A - B^T B\|_2 = {x^*}^T(A^T A - B^T B)x^* = \|Ax^*\|^2 - \|Bx^*\|^2 \leq \Delta.$$

Next, we use Property 3.1.2 verbatim and bootstrap Property 3.1.3 to prove a tighter bound on Δ. In the following, the vectors y_i correspond to singular vectors of A ordered with respect to decreasing corresponding singular values.

$$\begin{aligned}
\Delta\ell &\leq \|A\|_F^2 - \|B\|_F^2 && \text{via Property 3.1.3} \\
&= \sum_{i=1}^{k} \|Ay_i\|^2 + \sum_{i=k+1}^{d} \|Ay_i\|^2 - \|B\|_F^2 && \|A\|_F^2 = \sum_{i=1}^{d} \|Ay_i\|^2 \\
&= \sum_{i=1}^{k} \|Ay_i\|^2 + \|A - A_k\|_F^2 - \|B\|_F^2 && \\
&\leq \|A - A_k\|_F^2 + \sum_{i=1}^{k} \left(\|Ay_i\|^2 - \|By_i\|^2\right) && \sum_{i=1}^{k} \|By_i\|^2 < \|B\|_F^2 \\
&\leq \|A - A_k\|_F^2 + k\Delta. && \text{via Property 3.1.2}
\end{aligned}$$

Solving $\Delta\ell \leq \|A - A_k\|_F^2 + k\Delta$ for Δ to obtain $\Delta \leq \|A - A_k\|_F^2/(\ell - k)$, which proves the right hand side of the bound as $\|A^T A - B^T B\|_2 = \|Ax^*\|^2 - \|Bx^*\|^2 \leq \Delta \leq \|A - A_k\|_F^2/(\ell - k)$. Property 6.3.3 proves the left hand side of the bound and completes the proof. □

Next, we prove Theorem 3.1.2 that states a multiplicative error bound for FD.

Proof of Theorem 3.1.2. Let the the vectors y_i and v_i correspond to the singular vectors of A and B, respectively.

$$\begin{aligned}
\|A - \pi_B^k(A)\|_F^2 &= \|A\|_F^2 - \|\pi_B^k(A)\|_F^2 = \|A\|_F^2 - \sum_{i=1}^{k} \|Av_i\|^2 && \text{Pythagorean theorem} \\
&\le \|A\|_F^2 - \sum_{i=1}^{k} \|Bv_i\|^2 && \text{via Property 6.3.3} \\
&\le \|A\|_F^2 - \sum_{i=1}^{k} \|By_i\|^2 && \text{since } \sum_{i=1}^{k} \|Bv_i\|^2 \ge \sum_{i=1}^{k} \|By_i\|^2 \\
&\le \|A\|_F^2 - \sum_{i=1}^{k} (\|Ay_i\|^2 - \Delta) && \text{via Property 3.1.2} \\
&= \|A\|_F^2 - \|A_k\|_F^2 + k\Delta \\
&\le \|A - A_k\|_F^2 + \frac{k}{\ell - k}\|A - A_k\|_F^2 && \text{by } \Delta \le \|A - A_k\|_F^2/(\ell - k) \\
&= \frac{\ell}{\ell - k}\|A - A_k\|_F^2.
\end{aligned}$$

This concludes the proof of Theorem 3.1.2. It is convenient to set $\ell = \lceil k + k/\varepsilon \rceil$ which results in the standard bound form $\|A - \pi_B^k(A)\|_F^2 \le (1+\varepsilon)\|A - A_k\|_F^2$. □

3.1.2 Running Time Analysis

Each iteration of Algorithm 8 is dominated by the computation of the singular value decomposition of B. The standard running time of this operation is $O(d\ell^2)$ [74]. Since this loop is executed once per row in A, the total running time would naïvely be $O(nd\ell^2)$. However, we can improve it to $O(nd\ell)$ time at the expense of doubling the space used by the algorithm. Algorithm 9 gives the details.

Algorithm 9 FAST-FD

```
Input: $\ell, A \in R^{n\times d}$
$B \leftarrow$ all zeros matrix $\in R^{2\ell\times d}$
for $i \in 1, \dots, n$ do
    Insert $a_i$ into a zero valued row of $B$
    if $B$ has no zero valued rows then
        $[U, \Sigma, V] \leftarrow \mathrm{SVD}(B)$
        $\delta \leftarrow \sigma_\ell^2$
        $B \leftarrow \sqrt{\max(\Sigma^2 - I_\ell\delta, 0)} \cdot V^T$        # The last $\ell + 1$ rows of $B$ are zero valued.
return $B$
```

Note that in Algorithm 9 the SVD of B is computed only $n/(\ell+1)$ times because the "if" statement is only triggered once every $\ell + 1$ iterations, thereby exhibiting a total running time of $O((n/\ell)d\ell^2) = O(nd\ell)$. The reader should revisit the proofs in Section 3.1.1

and observe that they still hold. Consider the values of i for which the "if" statement is triggered, and let $C_{[i]} = \Sigma V^T$ in the i-the execution of "if" statement. It still holds that $0 \preceq C_{[i]}^T C_{[i]} - B_{[i]}^T B_{[i]} \preceq \delta I_d$ and that $\|C_{[i]}\|_F^2 - \|B_{[i]}\|_F^2 \geq \ell\delta$. For the other values of i, the sketch simply aggregates the input rows and there is clearly no incurred error in doing that. This is sufficient for the same analysis to go through and complete the discussion on correctness of Algorithm 9.

3.1.2.1 Update Time

The total running time of Algorithm 9 is $O(nd\ell)$ and the *amortized* running time per row update is $O(d\ell)$. However, the worst case update time is still $\Omega(d\ell^2)$ in those cases where the full singular value decomposition is computed. Using the fact that FD sketches are mergeable, we can actually use a simple trick to guarantee a worst case $O(d\ell)$ update time. The idea is to double the space usage (once again) and hold two sketches, one in 'active' mode and one in SVD 'maintenance' mode. For any row in the input, we first add it to the active sketch and then spend $O(d\ell)$ floating point operations in completing the SVD of the sketch in maintenance mode. After ℓ updates, the active sketch runs out of space and must go into maintenance mode. But, at the same time, a total of $O(d\ell^2)$ floating point operations were invested on the inactive sketch which completed its SVD computation. At this point, we switch the sketch roles and continue. Once the entire matrix is processed, we combine the two sketches using their mergeable property.

3.1.3 Parallelization and Merging Sketches

In extremely large datasets, processing is often distributed among several machines. Each machine receives a disjoint input of raw data and is tasked with creating a small space summary. Then to get a global summary of the entire data, these summaries need to be combined. The core problem is illustrated for the case of just two machines; each processes a dataset A_1 or A_2, where $A = [A_1; A_2]$, and separately create two summaries B_1 and B_2, respectively. Then the goal is to create a single summary B which approximates A using only B_1 and B_2. If B can achieve the same formal space/error tradeoff as each B_i to A_i in a streaming algorithm, then the summary is called a *mergeable summary* [9].

Here we show that the FD sketch is indeed mergeable under the following procedure. Consider $B' = [B_1; B_2]$ which has 2ℓ rows; then run FD (in particular Algorithm 9) on B' to create sketch B with ℓ rows. Given that B_1 and B_2 satisfy Properties 3.1.1, 3.1.2, and 3.1.3 with parameters Δ_1 and Δ_2, respectively, we will show that B satisfies the same properties with $\Delta = \Delta_1 + \Delta_2 + \delta$, where δ is taken from the single shrink operation used in Algorithm 9. This implies B automatically inherits the bounds in Theorem 3.1.1 and Theorem 3.1.2 as well.

First note that B' satisfies all three properties with $\Delta' = \Delta_1 + \Delta_2$, by additivity of squared spectral norm along any direction x (e.g. $\|B_1x\|^2 + \|B_2x\|^2 = \|B'x\|^2$) and squared Frobenius norms (e.g. $\|B_1\|_F^2 + \|B_2\|_F^2 = \|B'\|_F^2$), but has space twice as large as desired. Property 6.3.3 holds since B only shrinks all directions in relation to B'. Property 3.1.2 follows by considering any unit vector x and expanding $\|Bx\|^2$ as

$$\|Bx\|^2 \geq \|B'x\|^2 - \delta \geq \|Ax\|^2 - (\Delta_1 + \Delta_2) - \delta = \|Ax\|^2 - \Delta.$$

Similarly, Property 3.1.3 can be seen as

$$\|B\|_F^2 \leq \|B'\|_F^2 - \delta\ell \leq \|A\|_F^2 - (\Delta_1 + \Delta_2)\ell - \delta\ell = \|A\|_F^2 - \Delta\ell.$$

This property trivially generalizes to any number of partitions of A. It is especially useful when the matrix (or data) is distributed across many machines. In this setting, each machine can independently compute a local sketch. These sketches can then be combined in an arbitrary order using FD.

3.2 Space Lower Bounds

FrequentDirections is space optimal with respect to the both covariance error guarantee and projection error guarantee it achieves.

In [98] we presented nearly-matching lower bounds for covariance error guarantee of FrequentDirections. More precisely, we showed any algorithm that achieves these error guarantees requires the space (in *bits*) of at least d times the number of rows FD requires. The theorem below states this result.

Theorem 3.2.1. *Let B be a $\ell \times d$ matrix approximating a $n \times d$ matrix A such that $\|A^TA - B^TB\|_2 \leq \|A - A_k\|_F^2/(\ell - k)$ for all $0 \leq k < \ell$. Assuming constant number of bits is required to describe a word (i.e. a unit of memory), then any matrix sketching algorithm with guarantee $\|A^TA - B^TB\|_2 \leq \|A - A_k\|_F^2/(\ell - k)$ requires $\Omega(d\ell)$ bits of space.*

The consequence of Theorem 3.2.1 is that the space complexity of FD is optimal regardless of streaming issues. In other words, any algorithm satisfying $\|A^TA - B^TB\|_2 \leq \|A - A_k\|_F^2/(\ell - k)$ must use space $\Omega(\ell d)$ since this is the information lower bound for representing B; or equivalently any algorithm satisfying $\|A^TA - B^TB\|_2 \leq \varepsilon\|A - A_k\|_F^2$ must have $\ell = \Omega(k + 1/\varepsilon)$ and hence use $\Omega(d\ell) = \Omega(dk + d/\varepsilon)$ space.

There is nearly-matching lower bound for projection error guarantee of FrequentDirections, as well. In [124], David Woodruff proved that FrequentDirections is nearly tight with respect to the multiplicative error bound $\|A - \pi_B^k(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$ it achieves in Theorem 3.1.2 with $\ell = \lceil k + k/\varepsilon \rceil$ rows (i.e., $O(kd/\varepsilon)$ space) in the row-update streaming setting. The following theorem which appeared in our joint journal paper [98] shows this result.

Theorem 3.2.2. *Assuming a constant number of bits is required to describe a word (i.e., a unit of memory), any randomized matrix approximation streaming algorithm in the row-update model, which guarantees $\|A - \pi_B^k(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$ and succeeds with probability at least $2/3$, must use $\Omega(kd/\varepsilon)$ space.*

In fact, this theorem shows that finding B such that $\|A - AB^\dagger B\|_F^2 = \|A - \pi_B(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$ requires $\Omega(dk/\varepsilon)$ space in a row-update streaming setting.

CHAPTER 4

EXTENSIONS OF FREQUENT DIRECTIONS

In this section, we focus on improving the error/size tradeoff of FrequentDirections. The key operation in FrequentDirections is the rank reduction, where in each iteration it reduces the rank of the intermediate sketch to make space for processing the next upcoming rows in the stream. Below, we introduce variants of FrequentDirections by modifying rank reduction step in the algorithm.

The main structure of all variants is presented in Algorithm 10, where $S' \leftarrow$ ReduceRank(S) is a subroutine that reduces the rank of singular values matrix S, and differs for each variant. It sets at least one non-zero in S to 0 in S'; this leads to a reduced rank for $B_{[i]}$, in particular with one row as all 0s. Notationally we use σ_j as the jth singular value in S, and σ'_j as the jth singular value in S'.

Algorithm 10 (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 a zero valued rows of $B_{[i-1]}$;          # result is $B_{[i]}$
  if ($B_{[i]}$ has no zero valued rows) then
    $[U, S, V] \leftarrow \text{SVD}(B_{[i]})$
    $C_{[i]} = SV^T$                                 # Only needed for proof notation
    $S' \leftarrow$ ReduceRank$(S)$
    $B_{[i]} \leftarrow S'V^T$
return $B = B_{[n]}$
```

For FD, ReduceRank sets each $\sigma'_j = \sqrt{\sigma_j^2 - \delta_i}$ where $\delta_i = \sigma_\ell^2$. The runtime of FD can be improved 3.1.2 by doubling the space, and batching the SVD call. A similar approach is possible for variants we consider.

Since all our proposed algorithms on FrequentDirections share the same structure, to avoid repeating the proof steps, we abstract out three facts that these algorithms follow and prove that *any* algorithm with these facts satisfy the desired error bounds. This slightly generalizes (allowing for $\alpha \neq 1$) Properties 3.1.1, 3.1.2, and 3.1.3 in Section 3.1.1.

4.1 Generalized Bounds on FrequentDirections

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

- Property 1: For any unit vector x we have $\|Ax\|^2 - \|Bx\|^2 \geq 0$.
- Property 2: For any unit vector x we have $\|Ax\|^2 - \|Bx\|^2 \leq \Delta$.
- Property 3: $\|A\|_F^2 - \|B\|_F^2 \geq \alpha\Delta\ell$.

Lemma 4.1.1. *In any such algorithm, for any unit vector x:*

$$0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A - A_k\|_F^2/(\alpha\ell - k).$$

Proof. In the following, y_i correspond to the singular vectors of A ordered with respect to a decreasing corresponding singular value order.

$$\begin{aligned}
\alpha\Delta\ell &\leq \|A\|_F^2 - \|B\|_F^2 && \text{via Property 3}\\
&= \sum_{i=1}^{k}\|Ay_i\|^2 + \sum_{i=k+1}^{d}\|Ay_i\|^2 - \|B\|_F^2 && \|A\|_F^2 = \sum_{i=1}^{d}\|Ay_i\|^2\\
&= \sum_{i=1}^{k}\|Ay_i\|^2 + \|A - A_k\|_F^2 - \|B\|_F^2 &&\\
&\leq \|A - A_k\|_F^2 + \sum_{i=1}^{k}\left(\|Ay_i\|^2 - \|By_i\|^2\right) && \sum_{i=1}^{k}\|By_i\|^2 < \|B\|_F^2\\
&\leq \|A - A_k\|_F^2 + k\Delta. && \text{via Property 2}
\end{aligned}$$

Solving $\alpha\Delta\ell \leq \|A - A_k\|_F^2 + k\Delta$ for Δ to obtain $\Delta \leq \|A - A_k\|_F^2/(\alpha\ell - k)$, which combined with Property 1 and Property 2 proves the lemma. □

Lemma 4.1.2. *Any such algorithm described above satisfies the following error bound*

$$\|A - \pi_{B_k}(A)\| \leq \alpha\ell/(\alpha\ell - k)\|A - A_k\|_F^2,$$

where $\pi_{B_k}(\cdot)$ represents the projection operator onto B_k, the top k singular vectors of B.

Proof. Here, y_i correspond to the singular vectors of A as above and v_i to the singular vectors of B in a similar fashion.

$$\begin{aligned}
\|A - \pi_{B_k}(A)\|_F^2 = \|A\|_F^2 - \|\pi_{B_k}(A)\|_F^2 = \|A\|_F^2 - \sum_{i=1}^{k} \|Av_i\|^2 & \qquad \text{Pythagorean theorem} \\
\le \|A\|_F^2 - \sum_{i=1}^{k} \|Bv_i\|^2 \text{via Property 1} & \\
\le \|A\|_F^2 - \sum_{i=1}^{k} \|By_i\|^2 & \qquad \text{since } \sum_{i=1}^{j} \|Bv_i\|^2 \ge \sum_{i=1}^{j} \|By_i\|^2 \\
\le \|A\|_F^2 - \sum_{i=1}^{k} (\|Ay_i\|^2 - \Delta) & \qquad \text{via Property 2} \\
= \|A\|_F^2 - \|A_k\|_F^2 + k\Delta & \\
\le \|A - A_k\|_F^2 + \frac{k}{\alpha\ell - k}\|A - A_k\|_F^2 & \qquad \text{by } \Delta \le \|A - A_k\|_F^2/(\alpha\ell - k) \\
= \frac{\alpha\ell}{\alpha\ell - k}\|A - A_k\|_F^2. &
\end{aligned}$$

This completes the proof of lemma. □

Thus setting $\ell = k + 1/\varepsilon$ achieves $\|A^TA - B^TB\|_2 \le \varepsilon\|A - A_k\|_F^2$, and setting $\ell = k + k/\varepsilon$ achieves $\|A - \pi_{B_k}(A)\|_F^2 \le (1+\varepsilon)\|A - A_k\|_F^2$. FD maintains an $\ell \times d$ matrix B (i.e., using $O(\ell d)$ space), and it is shown [71] that there exists a value Δ that FD satisfies three above-mentioned facts with $\alpha = 1$.

4.2 Iterative SVD

The simplest variant of this procedure is a heuristic rediscovered several times [26, 74, 76, 90, 111], with a few minor modifications, and we refer to it as *iterative SVD* or iSVD. Here ReduceRank(S, V) simply keeps $\sigma'_j = \sigma_j$ for $j < \ell$ and sets $\sigma'_\ell = 0$. This has no worst case guarantees (despite several claims), and sometimes performs chaotically [63].

Consider an example where the first k rows of a dataset A generate a matrix A_k with kth singular value $\sigma_k = 10$. Then each row thereafter a_i for $i > k$ is orthogonal to the first k rows of A, and has norm 5. This will cause the $(k+1)$th right singular vector and value σ_{k+1} of SVD$([A_k^i; a_i])$ to exactly describe the subspace of a_i with $\sigma_{k+1} = 5$. Thus this row a_i will always be removed on the processing step and A_k^{i+1} will be unchanged from A_k^i. If all rows a_i for $i > k$ are pointing in the same direction, this can cause arbitrarily bad errors of all forms of measuring approximation error considered above.

4.3 Parameterized FD

Parameterized FD uses the following subroutine (Algorithm 11) to reduce the rank of the sketch; it zeros out row ℓ. 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 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 11 ReduceRank-PFD(S, α)

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

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 4.3.1. *For any unit vector x and any $\alpha \geq 0$: $0 \leq \|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 \leq \delta_i$.*

Proof. The right hand side is shown by just expanding $\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2$.

$$\|C_{[i]}x\|^2 - \|B_{[i]}x\|^2 = \sum_{j=1}^{\ell} \sigma_j^2 \langle v_j, x\rangle^2 - \sum_{j=1}^{\ell} \sigma'^2_j \langle v_j, x\rangle^2 = \sum_{j=1}^{\ell} (\sigma_j^2 - \sigma'^2_j)\langle v_j, x\rangle^2$$

$$= \delta_i \sum_{j=(1-\alpha)\ell+1}^{\ell} \langle v_j, x\rangle^2 \leq \delta_i \|x\|^2 = \delta_i$$

To see the left side of the inequality $\delta_i \sum_{j=(1-\alpha)\ell+1}^{\ell} \langle v_j, x\rangle^2 \geq 0$. □

Then summing over all steps of the algorithm (using $\|a_i x\|^2 = \|C_{[i]}x\|^2 - \|B_{[i-1]}x\|^2$) it follows 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 4.3.2. *For any $\alpha \in (0,1]$, $\|A\|_F^2 - \|B\|_F^2 = \alpha\Delta\ell$, proving Property 3.*

Proof. We expand that $\|C_{[i]}\|_F^2 = \sum_{j=1}^{\ell} \sigma_j^2$ to get

$$\|C_{[i]}\|_F^2 = \sum_{j=1}^{(1-\alpha)\ell} \sigma_j^2 + \sum_{j=(1-\alpha)\ell+1}^{\ell} \sigma_j^2 = \sum_{j=1}^{(1-\alpha)\ell} \sigma'^2_j + \sum_{j=(1-\alpha)\ell+1}^{\ell} (\sigma'^2_j + \delta_i) = \|B_{[i]}\|_F^2 + \alpha\ell\delta_i.$$

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. □ □

The combination of the three Properties, provides the following results.

Theorem 4.3.1. *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 $k > \alpha\ell$*

$$0 \le \|Ax\|^2 - \|Bx\|^2 \le \|A - A_k\|_F^2/(\alpha\ell - k)$$

and projection of A onto B_k, the top k rows of B satisfies

$$\|A - \pi_{B_k}(A)\|_F^2 \le \frac{\alpha\ell}{\alpha\ell - k}\|A - A_k\|_F^2.$$

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

4.3.1 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 $\|A^TA - B^TB\|_2 \le \varepsilon\|A - A_k\|_F^2$ and setting $\ell = 2(k + k/\varepsilon)/\alpha$ yields $\|A - \pi_{B_k}(A)\|_F^2 \le (1+\varepsilon)\|A - A_k\|_F^2$. In experiments we consider Fast 0.2-FD.

4.4 SpaceSaving Directions

SpaceSaving Directions (abbreviated SSD) uses Algorithm 12 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 12 ReduceRank-SS(S)

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

Then to understand the error bounds for ReduceRank-SS, we will consider an arbitrary unit vector x. We can decompose $x = \sum_{j=1}^{d} \beta_j v_j$ where $\beta_j^2 = \langle x, v_j \rangle^2 > 0$ and $\sum_{j=1}^{d} \beta_j^2 = 1$. For notational convenience, without loss of generality, we assume that $\beta_j = 0$ for $j > \ell$. Thus $v_{\ell-1}$ represents the entire component of x in the null space of B (or $B_{[i]}$ after processing row i).

To analyze this algorithm, at iteration $i \geq \ell$, we consider a $d \times d$ matrix $\bar{B}_{[i]}$ that has the following properties: $\|B_{[i]} v_j\|^2 = \|\bar{B}_{[i]} v_j\|^2$ for $j < \ell - 1$ and $j = \ell$, and $\|\bar{B}_{[i]} v_j\|^2 = \delta_i$ for $j = \ell - 1$ and $j > \ell$. This matrix provides the constant but bounded overcount similar to the SS sketch. Also let $A_{[i]} = [a_1; a_2; \ldots; a_i]$.

Lemma 4.4.1. *For any unit vector x we have* $0 \leq \|\bar{B}_{[i]} x\|^2 - \|A_{[i]} x\|^2 \leq 2\delta_i$

Proof. We prove the first inequality by induction on i. It holds for $i = \ell - 1$, since $B_{[\ell-1]} = A_{[\ell-1]}$, and $\|\bar{B}_{[i]} x\|^2 \geq \|B_{[i]} x\|^2$. We now consider the inductive step at i. Before the reduce-rank call, the property holds, since adding row a_i to both $A_{[i]}$ (from $A_{[i-1]}$) and $C_{[i]}$ (from $B_{[i-1]}$) increases both squared norms equally (by $\langle a_i, x \rangle^2$) and the left rotation by U^T also does not change norms on the right. On the reduce-rank, norms only change in directions v_ℓ and $v_{\ell-1}$. Direction v_ℓ increases by δ_i, and in $\bar{B}_{[i]}$ the directions $v_{\ell-1}$ also does not change, since it is set back to δ_i, which it was before the reduce-rank.

We prove the second inequality also by induction, where it also trivially holds for the base case $i = \ell - 1$. Now we consider the inductive step, given it holds for $i - 1$. First observe that $\delta_i \geq \delta_{i-1}$ since δ_i is at least the $(\ell-1)$st squared singular value of $B_{[i-1]}$, which is at least δ_{i-1}. Thus, the property holds up to the reduce rank step, since again, adding row a_i and left-rotating does not affect the difference in norms. After the reduce rank, we again only need to consider the two directions changed $v_{\ell-1}$ and v_ℓ. By definition

$$\|A_{[i]}v_{\ell-1}\|^2 + 2\delta_i \geq \delta_i = \|\bar{B}_{[i]}v_{\ell-1}\|^2,$$

so direction $v_{\ell-1}$ is satisfied. Then

$$\|\bar{B}_{[i]}v_\ell\|^2 = \|B_{[i]}v_\ell\|^2 = \delta_i + \|C_{[i]}v_\ell\|^2 \leq 2\delta_i$$

and $0 \leq \|A_{[i]}v_\ell\|^2 \leq \|\bar{B}_{[i]}v_\ell\|^2$. Hence $\|\bar{B}_{[i]}v_\ell\|^2 - \|A_{[i]}v_\ell\|^2 \leq 2\delta_i - 0$, satisfying the property for direction v_ℓ, and completing the proof. □ □

Now we would like to prove the three Properties needed for relative error bounds for $B = B_{[n]}$. But this does not hold since $\|B\|_F^2 = \|A\|_F^2$ (an otherwise nice property), and $\|\bar{B}\|_F^2 \gg \|A\|_F^2$. Instead, we first consider yet another matrix $\hat{B}$ defined as follows with respect to B. B and $\hat{B}$ have the same right singular values V. Let $\delta = \delta_n$, and for each singular value σ_j of B, adjust the corresponding singular values of $\hat{B}$ to be $\hat{\sigma}_j = \max\{0, \sqrt{\sigma_j^2 - 2\delta}\}$.

Lemma 4.4.2. *For any unit vector x we have $0 \leq \|Ax\|^2 - \|\hat{B}x\|^2 \leq 2\delta$ and $\|A\|_F^2 - \|\hat{B}\|_F^2 \geq \delta(\ell-1)$.*

Proof. Directions v_j for $j > \ell - 1$, the squared singular values are shrunk by at least δ. The squared singular value is already 0 for direction $v_{\ell-1}$. And the singular value for direction v_ℓ is shrunk by δ to be exactly 0. Since before shrinking $\|B\|_F^2 = \|A\|_F^2$, the second expression in the lemma holds.

The first expression follows by Lemma 4.4.1 since $\bar{B}$ only increases the squared singular values in directions v_j for $j = \ell - 1$ and $j > \ell$ by δ, which are 0 in $\hat{B}$. And other directions v_j are the same for $\bar{B}$ and B and are at most 2δ larger than in A. □

Thus $\hat{B}$ satisfies the three Properties. We can now state the following property about B directly, setting $\alpha = (1/2)$, adjusting ℓ to $\ell - 1$, then adding back the at most $2\delta = \Delta \leq \|A - A_k\|_F^2/(\alpha\ell - \alpha - k)$ to each directional norm.

Theorem 4.4.1. *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 $k < \frac{\ell-1}{2}$, $|\|Ax\|^2 - \|Bx\|^2| \leq \|A - A_k\|_F^2/(\ell/2 - 1/2 - k)$.*

- *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 $\|A - \pi_{B_k}(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$.

4.5 Compensative Frequent Directions

In original FD, the computed sketch B underestimates Frobenius norm of stream [69]. In Compensative FrequentDirections (abbreviated CFD), 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 $\hat{\sigma}_j = \sqrt{\sigma_j'^2 + \Delta}$, 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 $\|A - \pi_B^k(A)\|_F^2 \leq \|A - A_k\|_F^2 \ell/(\ell - k)$. 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 $\ell = kk/\varepsilon$ yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$.

4.6 Experimental Evaluation

Herein we describe an extensive set of experiments on a wide variety of large input datasets. We focus on comparing the amount of space used by each type of sketch (measured in rows) against several error measures. We show improvements over FD (and in one instance iSVD) by our proposed algorithm 0.2-FD. Each dataset is an $n \times d$ matrix A, and the n rows are processed one-by-one in a stream.

4.6.1 Datasets

We compare performance of our algorithms on both synthetic and real datasets; see a summary in Table 4.1. We also generate adversarial data to show that iSVD performs poorly under specific circumstances. This explains why there is no theoretical guarantee for them.

For Random Noisy, we generate the input $n \times d$ matrix A synthetically, mimicking the approach by Liberty [91]. We compose $A = SDU + F/\zeta$, where SDU is the m-dimensional signal (for $m < d$) and G/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 with 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 $m \in \{10, 20, 30, 50\}$ (the default is $m = 50$).

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 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 three real-world datasets. Birds [3] 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 [1] 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. ConnectUS is from University of Florida Sparse Matrix collection [27], representing 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 webpages preferences with any other users.

4.6.2 Approximation Error vs. Sketch Size

We measure error for all algorithms as we change the parameter ℓ (Sketch Size) determining the number of rows in matrix B. We measure covariance error as $\text{err} = \|A^TA - B^TB\|_2 / \|A\|_F^2$ (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 also consider $\text{proj-err} = \|A - \pi_{B_k}(A)\|_F^2 / \|A - A_k\|_F^2$, always using $k = 10$

(Projection Error); for FD we should have proj-err $\leq \ell/(\ell - 10)$, and ≥ 1 in general.

We denote each variant of Parameterized FD as α-FD in Figure 4.1. 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 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 seem to be approaching the same small error. We also explore the effect on α-FD in Figure 4.2 on Random Noisy data by varying $m \in \{10, 20, 30\}$, and $m = 50$ in Figure 4.1. 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.

In Figure 4.3, we compare iSVD, FD, and 0.2-FD with the other variants based on the SS streaming algorithm: CFD and SSD. We see that these typically perform slightly better than FD, but not nearly as well as 0.2-FD and iSVD. Perhaps it is surprising that although SpaceSaving 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.

Finally, we show that iSVD is not always better in practice. Using the Adversarial construction in Figure 4.4, we see that iSVD can perform much worse than the 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 $\ell = 100$. Moreover, all version of α-FD obtain roughly err=0.005 already for $\ell = 20$. 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 Parameterized FD algorithm.

To conclude we demonstrate the scalability of these approaches on a much larger real dataset ConnectUS. Figure 4.5 shows variants of Parameterized FD, those compared against SpaceSaving variants on this dataset. As the derived bounds on covariance error based on sketch size do not all 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 of differences to point out. First, no algorithm converges as close to 0 error as with the other smaller datasets; this is likely because with the much larger size, there is some variation that cannot 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 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.

Table 4.1: Datasets; Numeric Rank Is Defined $\|A\|_F^2/\|A\|_2^2$.

DataSet	# Datapoints	# Attributes	Rank	Numeric Rank
Random Noisy	10000	500	500	m=50 m=30 m=20 m=10 21.62, 15.39, 11.82, 8.79
Adversarial	10000	500	500	1.69
Birds[3]	11788	312	312	12.50
Spam[1]	9324	499	499	3.25
connectUS	394792	512	512	4.83

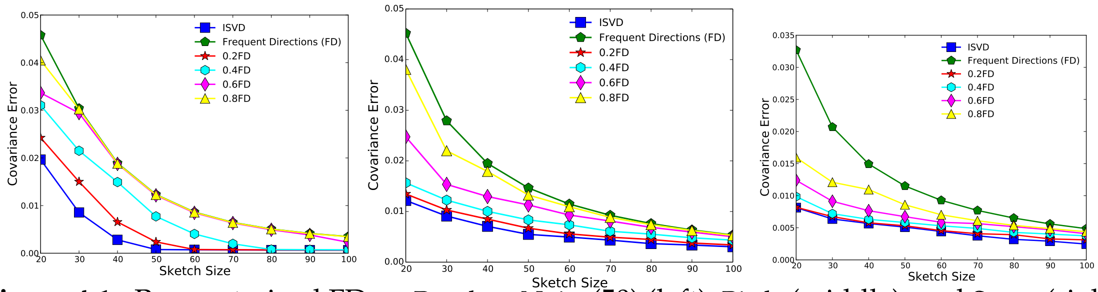


Figure 4.1: Parameterized FD on Random Noisy(50) (left), Birds (middle), and Spam (right).

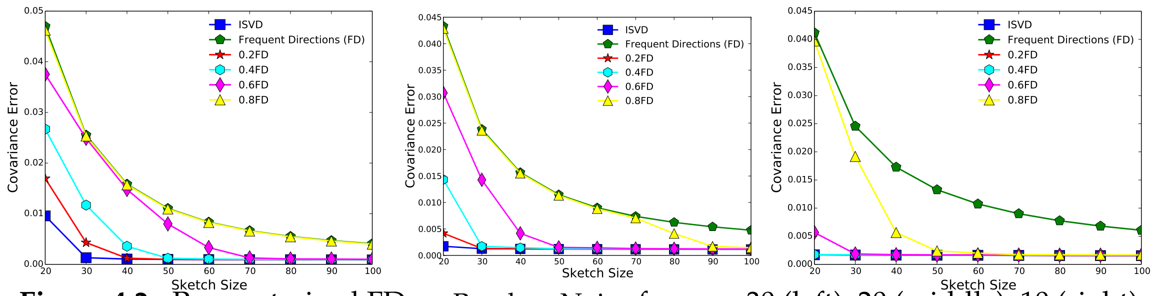


Figure 4.2: Parameterized FD on Random Noisy for $m = 30$ (left), 20 (middle), 10 (right).

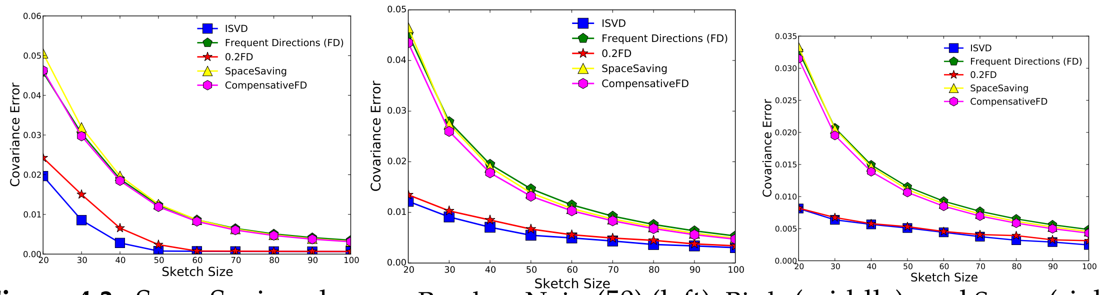


Figure 4.3: SpaceSaving algos on Random Noisy(50) (left), Birds (middle), and Spam (right).

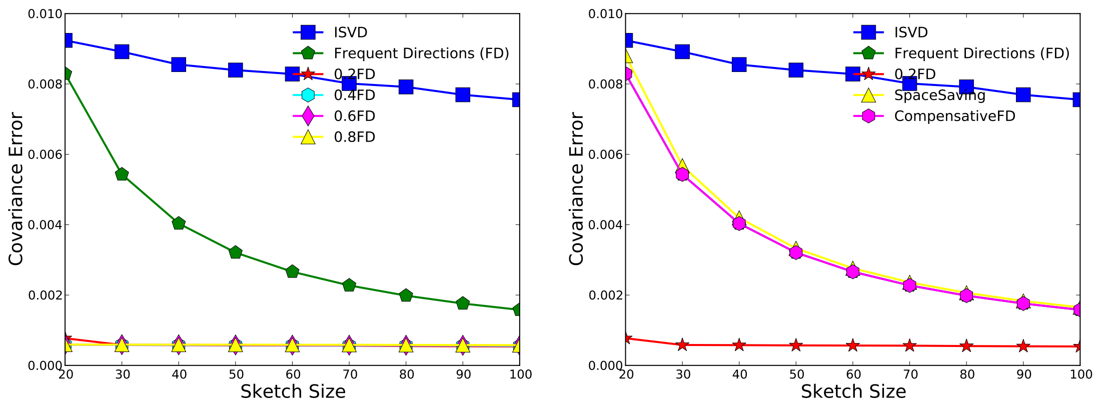


Figure 4.4: Demonstrating dangers of iSVD on Adversarial data.

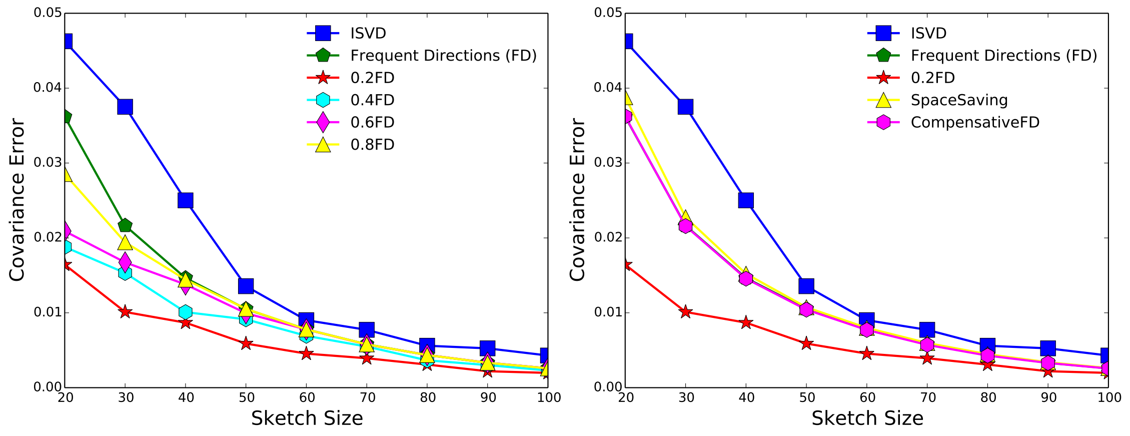


Figure 4.5: Parameterized FD (left), SpaceSaving-based (right) on ConnectUS dataset.

CHAPTER 5

OTHER MATRIX SKETCHING APPROACHES

In Chapter 1, we briefly reviewed main matrix sketching approaches. Among those, were the three following:

- sampling algorithms: these select a subset of rows or columns from A to use as the sketch B;
- projection algorithms: these project the n rows of A onto ℓ rows of B, sometimes using hashing;
- incremental algorithms: these maintain B as a low-rank version of A updated as more rows are added.

In Chapters 3 and 4 we studied instances of iterative sketching approach. In this chapter, we introduce a few new variants, and then compare all with the new FD-related iterative algorithms.

5.1 Column/Row Sampling Techniques

Sampling algorithms assign a probability p_i for each row a_i and then selecting 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.* [49] and[61]. These algorithms typically advocate sampling ℓ items independently (with replacement) using ℓ distinct reservoir samplers, taking $O(\ell)$ time per element. Another version [54] samples each row independently, and only retains ℓ rows in expectation. We discuss two improvements to this process in Section 5.1.3.

Much of the related literature describes selecting columns instead of rows (called the *column subset selection problem*) [24]. This is just a transpose of the data and has no real difference from what is described here. There are also techniques [54] that select both columns and rows. This include CUR decomposition [48, 54, 94] where they construct an approximate matrix $\hat{A} = CUR$ such that C is a sample of columns of A, and R is a sample of rows of A. In the construction, matrix U is small and dense, and C and R are sparse and skinny, or others [33] where the middle matrix is still diagonal. The sparsity is often preserved by constructing the wrapper matrices (e.g., C and R) from the original columns or rows of A.

This family of techniques has the advantage that the resulting sketch is *interpretable* in that each row of B corresponds to data point in A, not just a linear combination of them.

Almost all column sampling algorithms have a projection bound in terms of $\|A - \pi_B A\|_F^2 \leq f(\varepsilon)\|A - A_k\|_F^2$ where $A \in \mathbb{R}^{n\times d}$ is the input matrix and $B \in \mathbb{R}^{\ell\times d}$ is the output sketch. Here we derive the other type of bound, cov-err, for the two main algorithms in this regime, i.e., Norm Sampling and Leverage Sampling. Table 5.1 summarizes this result. In our proof, we use a variant of Chernoff-Hoeffding inequality: Consider a set of r independent random variables $\{X_1, \cdots, X_r\}$ where $0 \leq X_i \leq \Delta$. Let $M = \sum_{i=1}^{r} X_i$, then for any $\alpha \in (0, 1/2)$

$$\mathbf{Pr}\left[|M - \mathrm{E}[M]| > \alpha\right] \leq 2\exp\left(\frac{-2\alpha^2}{r\Delta^2}\right).$$

Lemma 5.1.1. *Let $B \in \mathbb{R}^{\ell\times d}$ with $\ell = O(d/\varepsilon^2)$ be the output of Norm Sampling. Then with probability* 99/100, *for all unit vectors $x \in \mathbb{R}^d$*

$$\textit{cov-err}(A, B) = \frac{|\|Bx\|^2 - \|Ax\|^2|}{\|A\|_F^2} \leq \varepsilon.$$

Proof. Consider any unit vector $x \in \mathbb{R}^d$. Define ℓ independent random variables $X_i = \langle b_i, x\rangle^2$ for $i = 1, \ldots, \ell$. Recall that Norm Sampling selects row a_j to be row b_i in the sketch matrix B with probability $\mathbf{Pr}(b_i \leftarrow a_j) = \|a_j\|^2/\|A\|_F^2$, and rescales the sampled row as $\|b_i\|^2 = \|a_j\|^2/(\ell\mathbf{Pr}(b_i \leftarrow a_j)) = \|A\|_F^2/\ell$. Knowing these, we bound each X_i as $0 \leq X_i \leq \|b_i\|^2 = \|A\|_F^2/\ell$ therefore $\Delta_i = \|A\|_F^2/\ell$ for all X_is. Setting $M = \sum_{i=1}^{\ell} X_i = \|Bx\|^2$, we observe

$$\mathrm{E}[M] = \sum_{i=1}^{\ell}\mathrm{E}[X_i] = \sum_{i=1}^{\ell}\sum_{j=1}^{n}\mathbf{Pr}(b_i \leftarrow a_j)\left\langle \frac{a_j}{\sqrt{\ell\mathbf{Pr}(b_i \leftarrow a_j)}}, x\right\rangle^2 = \sum_{i=1}^{\ell}\sum_{j=1}^{n}\frac{1}{\ell}\langle a_j, x\rangle^2 = \|Ax\|^2.$$

Finally using the Chernoff-Hoeffding bound and setting $\alpha = \varepsilon\|A\|_F^2$ yields

$$\mathbf{Pr}\left[|\|Bx\|^2 - \|Ax\|^2| > \varepsilon\|A\|_F^2\right] \leq 2\exp\left(\frac{-2(\varepsilon\|A\|_F^2)^2}{\ell(\|A\|_F^2/\ell)^2}\right) = 2\exp\left(-2\varepsilon^2\ell\right) \leq \delta.$$

Letting the probability of failure for that x be $\delta = 1/100$, and solving for ℓ in the last inequality, we obtain $\ell \geq \frac{1}{2\varepsilon^2}\ln(2/\delta) = \frac{1}{2\varepsilon^2}\ln(200)$.

However, this only holds for a single direction x; we need this to hold for all unit vectors x. It can be shown that allowing $\alpha = O(\varepsilon)\cdot\|A\|_F^2$, we actually only need this to hold for a net T of size $t = 2^{O(d)}$ such directions x [125]. Then by the union bound, setting $\delta = 1/(100t)$ this will hold for all unit vectors in T, and thus (after scaling ε by a constant) we can solve for $\ell = O((1/\varepsilon^2)\ln(t)) = O(d/\varepsilon^2)$. Hence with probability at least 99/100 we have $\text{cov-err}(A,B) = \frac{|\|Bx\|^2 - \|Ax\|^2|}{\|A\|_F^2} \leq \varepsilon$ for all unit vectors x. □

We would like to apply a similar proof for Leverage Sampling, but we do not obtain a good bound for Δ in the Chernoff-Hoeffding bound. We rescale norm of each selected row b_i as

$$\|b_i\|^2 = \frac{\|a_j\|^2}{\ell}\cdot\frac{1}{\mathbf{Pr}(b_i \leftarrow a_j)} = \frac{\|a_j\|^2}{\ell}\cdot\frac{S_k}{s_j^{(k)}},$$

where $s_j^{(k)}$ is the rank-k leverage score of a_j, and $S_k = \sum_{j=1}^n s_j^{(k)} = k$. Unfortunately, $s_j^{(k)}$ can be arbitrarily small compared to S_k (e.g., if a_j lies almost entirely outside the best rank-k subspace). And thus we do not have a finite bound on Δ. As such we assume that $S_k/s_j^{(k)} \leq \beta$ for an absolute constant β, and then obtain a bound based on β.

Lemma 5.1.2. *Let $B \in \mathbb{R}^{\ell\times d}$ with $\ell = O(d\beta^2/\varepsilon^2)$ be the output of Leverage Sampling. Under the assumption that rank-k leverage score of row a_j is bounded as $s_j^{(k)} \geq \beta S_k$ for a fixed constant $\beta > 0$, then with probability 99/100, for all unit vectors $x \in \mathbb{R}^d$*

$$\textit{cov-err} = |\|Bx\|^2 - \|Ax\|^2|/\|A\|_F^2 \leq \varepsilon$$

Proof. Similar to Lemma 5.1.1, we define ℓ random variables $X_i = \langle b_i, x\rangle^2$ for $i = 1,\cdots,\ell$. Leverage Sampling algorithm selects row a_j to be row b_i with probability $\mathbf{Pr}(b_i \leftarrow a_j) = s_j^{(k)}/S_k$ and rescales sampled rows as $\|b_i\|^2 = \|a_j\|^2/(\ell\mathbf{Pr}(b_i \leftarrow a_j)) = (\|a_j\|^2/\ell)(S_k/s_j^{(k)}) \leq \|a_j\|^2/(\beta\ell) \leq \|A\|_F^2/(\beta\ell)$. Therefore $\Delta_i = \|A\|_F^2/(\beta\ell)$ for all X_is. Setting $M = \sum_{i=1}^{\ell} X_i = \|Bx\|^2$ we observe:

$$\mathrm{E}[M] = \sum_{i=1}^{\ell}\mathrm{E}\left[X_i\right] = \sum_{i=1}^{\ell}\sum_{j=1}^{n}\mathbf{Pr}(b_i \leftarrow a_j)\left\langle\frac{a_j}{\sqrt{\ell\mathbf{Pr}(b_i \leftarrow a_j)}}, x\right\rangle^2 = \sum_{i=1}^{\ell}\sum_{j=1}^{n}\frac{1}{\ell}\langle a_j, x\rangle^2 = \|Ax\|^2$$

Using Chernoff-Hoeffding bound with parameter $\alpha = \varepsilon \|A\|_F^2$ gives:

$$\mathbf{Pr}\left[|\|Bx\|^2 - \|Ax\|^2| > \varepsilon\|A\|_F^2\right] \leq 2\exp\left(\frac{-2\varepsilon^2\|A\|_F^4}{\sum_{j=1}^{\ell}(\beta^2/\ell^2)\|A\|_F^4}\right) = 2\exp\left(\frac{-2\ell\varepsilon^2}{\beta^2}\right) \leq \delta$$

Again letting the probability of failure for that x be $\delta = 1/100$, we obtain $\ell \geq \frac{\beta^2}{2\varepsilon^2}\ln(2/\delta) = \frac{\beta^2}{2\varepsilon^2}\ln(200)$.

In order to hold this for all unit vectors x, again we consider a net T of size $t = 2^{O(d)}$ unit directions x [125]. Then by the union bound, setting $\delta = 1/(100t)$ this will hold for all such vectors in T, and thus we can solve for $\ell = O((\beta^2/\varepsilon^2)\ln(t)) = O(d\beta^2/\varepsilon^2)$. Hence with probability at least $99/100$ we have $\text{cov-err}(A, B) = \frac{|\|Bx\|^2 - \|Ax\|^2|}{\|A\|_F^2} \leq \varepsilon$ for all unit vectors x. $\square$

5.1.1 Leverage Sampling

An insightful adaptation changes the probability p_i using *leverage scores* [53] or *simplex volume* [44,45]. 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 [54] as a representative; it samples rows according to leverage scores (described below). Simplex volume calculations [44,45] were too involved to be practical. There are also recent techniques to improve on the theoretical runtime for leverage sampling [52] 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 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. $p_i = s_i/k$. Note that $\sum_i s_i = k$).

5.1.2 Deterministic Leverage Scores

Another option is to deterministically select rows with the highest s_i values instead of at random. This can be implemented with a simple priority queue of size ℓ. This has been applied to using the leverage scores by Papailiopoulos *et al.* [108], which again first requires calculating the SVD of A. We refer to this algorithm as Deterministic Leverage Sampling.

5.1.3 New Without Replacement Sampling Algorithms

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 [120]; however, variants for weighted data have been much less resolved until recently [34, 58].

5.1.3.1 Priority Sampling

A simple technique [58] 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 (which 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 $\rho_i \le \tau$, 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 $w_i \ge \tau$ 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.

5.1.3.2 VarOpt Sampling

VarOpt (or Variance Optimal) sampling [34] is a modification of priority sampling that takes more care in selecting the threshold τ. In priority sampling, τ is generated so $\mathrm{E}[\sum_{a_i \in B} \hat{w}_i] = \|A\|_F^2$, 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. Table 5.2 summarizes runtime and error bounds of VarOpt and Priority sampling algorithms.

A similar algorithm using priority sampling was considered in a distributed streaming setting [64], 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 similar analysis as above. It is an open question to bound the projection error for these algorithms, but we conjecture the bounds will match those of Norm Sampling.

5.2 Random Projection Techniques

These methods linearly project the n rows of A to ℓ rows of B. A survey by Woodruff [125] (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 [81] as first shown by Sarlos [113] and strengthened by Clarkson and Woodruff [32]. Gaussian random variables $s_{j,i}$ can be replaced with (appropriately scaled) $\{-1, 0, +1\}$ or $\{-1, +1\}$ random variables [5]. We call the version with scaled $\{-1, +1\}$ random variables as Random Projection. Table 5.3 contains runtime and error bounds of a few methods of this group that we discuss shortly.

Most random projection algorithms state a bound (with constant probability) that they can create a matrix $B = SA$ such that $\|Bx\| = (1 \pm \varepsilon)\|Ax\|$ (e.g., $(1-\varepsilon)\|Ax\| \leq \|Bx\| \leq (1+\varepsilon)\|Ax\|$) for all $x \in \mathbb{R}^d$.

Here we relate this to cov-err and proj-err. We first show that whether this bound is squared only affects ε by a constant factor.

Lemma 5.2.1. *For $\varepsilon \in (0, \frac{1}{4})$, $\|Bx\| = (1 \pm \varepsilon)\|Ax\|$ implies $\|Bx\|^2 = (1 \pm 3\varepsilon)\|Ax\|^2$.*

Proof. The upper bound follows since $(1+\varepsilon)^2 = 1 + 2\varepsilon + \varepsilon^2 \leq 1 + 3\varepsilon$ for $\varepsilon \in (0, \frac{1}{4})$. The lower bound follows since $(1-\varepsilon)^2 = 1 + \varepsilon^2 - 2\varepsilon \geq 1 - 2\varepsilon$ for $\varepsilon \in (0, \frac{1}{4})$. □

Now to relate this bound to cov-err, we will use $\rho(A) = \|A\|_F^2/\|A\|_2^2$, the *numeric rank* of A, which is always at least 1.

Lemma 5.2.2. *Given a matrix B such that $\|Bx\| = (1 \pm \varepsilon)\|Ax\|$ for all $x \in \mathbb{R}^d$, then when $\varepsilon \in (0, \frac{1}{4})$*

$$\text{cov-err} = \|A^T A - B^T B\|_2 / \|A\|_F^2 \leq 3\varepsilon/\rho(A).$$

Proof. Using Lemma 5.2.1, $|\|Ax\|^2 - \|Bx\|^2| \leq 3\varepsilon\|Ax\|^2$. Now restrict $\|x\| = 1$, so then

$$x^T(A^T A - B^T B)x = x^T A^T A x - x^T B^T B x = |\|Ax\|^2 - \|Bx\|^2| \leq 3\varepsilon\|Ax\|^2.$$

Since this holds for all x such that $\|x\| = 1$, then it holds for the $x = x^*$ which maximizes the left hand side so ${x^*}^T(A^T A - B^T B)x^* = \|A^T A - B^T B\|_2$ so

$$\|A^T A - B^T B\|_2 = {x^*}^T(A^T A - B^T B)x^* \leq 3\varepsilon\|Ax^*\|^2 \leq 3\varepsilon\|A\|_2^2.$$

Dividing both sides by $\|A\|_F^2 = \rho(A)/\|A\|_2^2$ completes the proof. □

We next relate this to the proj-err. We note that it may be possible that the full property $\|Bx\| = (1 \pm \varepsilon)\|Ax\|$ may not be necessary to obtain a bound on proj-err, but we are not aware of an explicit statement otherwise. There are bounds (see [125]) where one reconstructs a matrix $\hat{A}$ which obtains the bounds below in place of $\pi_{B_k}(A)$, and these have roughly $1/\varepsilon$ dependence on ε; however, they also have a factor n in their size.

Lemma 5.2.3. *Given a matrix B such that $\|Bx\| = (1 \pm \varepsilon)\|Ax\|$ for all $x \in \mathbb{R}^d$, then when $\varepsilon \in (0, \frac{1}{4})$*

$$\text{proj-err} = \|A - \pi_{B_k}(A)\|_F^2 / \|A - A_k\|_F^2 \leq (1 + 24\varepsilon).$$

Proof. Let $V = [v_1, v_2, \ldots, v_d]$ be the right singular vectors of A, so that $\|A_k\|_F^2 = \sum_{i=1}^k \|Av_i\|^2$ and $\|A - A_k\|_F^2 = \sum_{i=k+1}^d \|Av_i\|^2$. It follows from $\|B_k\|_F^2 \geq \sum_{i=1}^k \|Bv_i\|^2$ and Lemma 5.2.1 that

$$\|B - B_k\|_F^2 \leq \sum_{i=k+1}^d \|Bv_i\|^2 \leq \sum_{i=k+1}^d \frac{1}{1-3\varepsilon}\|Av_i\|^2 \leq \frac{1}{1-3\varepsilon}\|A - A_k\|_F^2.$$

Let $R = [r_1, r_2, \ldots, r_d]$ be the right singular vectors of B. Then by matrix Pythagorean and Lemma 5.2.1

$$\begin{aligned}\|A - \pi_{B_k}(A)\|^2 = \sum_{i=k+1}^{d} \|Ar_i\|^2 \leq \sum_{i=k+1}^{d} (1+3\varepsilon)\|Br_i\|^2 = (1+3\varepsilon)\|B - B_k\|_F^2 \\ \leq \frac{1+3\varepsilon}{1-3\varepsilon}\|A - A_k\|_F^2 \leq (1+24\varepsilon)\|A - A_k\|_F^2.\square\end{aligned}$$

5.2.1 Fast JLT

Using a sparse projection matrix X would improve the runtime, but these lose guarantees if the input is also sparse (if the non-zero elements do not align). This is circumvented by rotating the space with a Hadamard matrix [11], which can be applied more efficiently using FFT tricks, despite being dense. More precisely, we use three matrices: P is $\ell \times n$ and has entries with iid 0 with probability $1-q$ and a Gaussian random variable with variance ℓ/q with probability $q = \min\{1, \Theta((\log^2 n)/d)\}$. H is $n \times n$ and a random Hadamard (this requires n to be padded to a power of 2). D is diagonal with random $\{-1, +1\}$ in each diagonal element. And 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 [12, 40] which we do not pursue here; we point the reader here [118] for a discussion of some of these extensions.

5.2.2 Sparse Random Projections

Clarkson and Woodruff [33] analyzed a very sparse projection matrix S, conceived of earlier [40, 122]; 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 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 [29] on rows instead of counts. We refer to this as Hashing.

A slight modification by Nelson and Nguyen [102], 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 $s = 4$).

5.3 Experimental Comparison

We divide our experimental evaluation into four sections: The first two sections contain comparisons within algorithms of each group (sampling and random projection), while the third compares accuracy and runtime of exemplar algorithm in each group against each other and against exemplars of iterative sketching discussed in the previous Chapter.

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

5.3.1 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.

5.3.1.1 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 for them. Each dataset is an $n \times d$ matrix A, and the n rows are processed one-by-one in a stream.

Table 5.4 lists all datasets with information about their n, d, $\text{rank}(A)$, 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 fatter 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 [92]. 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 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 [2]. 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 webpages preferences with any other users. Birds [3] 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 [1] 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 [85].

The singular values distribution of the datasets is given in Figure 5.1. The x-axis is the singular value index, and the y-axis shows the normalized singular values, i.e., singular values divided by σ_1, where σ_1 is the largest singular value of dataset. Birds, ConnectUS and Spam have consistent drop-offs in singular values. Random Noisy has initial sharp and consistent drops in singular values, and then a more gradual decrease. The drop-offs in CIFAR-10 and Adversarial are more dramatic.

We will focus most of our experiments on three datasets Birds (dense, tall, large numeric rank), Spam (sparse, not tall, negative kurtosis, high numeric rank), and Random Noisy (dense, tall, synthetic). However, 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.

5.3.2 Row/Column Sampling Algorithms

Figure 5.2 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 little 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 fat, 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 under some restrictions for Leverage Sampling, while unrestricted for the other randomized sampling algorithms.

5.3.3 Random Projection Algorithms

Figure 5.3 plots the covariance and projection error, as well as the runtime for various sketch sizes of 20 to 100 for the projection algorithms.

Otherwise, there were two clear classes of algorithms. For the same sketch size, Hashing and OSNAP perform a bit worse on projection error (most clearly on Noisy Random), and roughly the same in covariance error, compared to Random Projections and Fast JLT. Note that Fast JLT seems consistently better than others in cov-err, but we have chosen the best q parameter (sampling rate) by trial and error, so this may give an unfair advantage. Moreover, Hashing and OSNAP also have significantly faster runtime, 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. Section 5.3.4 on larger datasets and sketch sizes shows that if the size of the sketch is not as important as runtime, Hashing and OSNAP have the advantage.

5.3.4 Global Comparison

Figure 5.4 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 the 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 Random Noisy dataset where Hashing and OSNAP result in worse projection error.

However, if we allow a much larger sketch size for faster runtime and small error, then these plots do not effectively demonstrate which algorithm performs best. Thus in Figure 5.5 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 this 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 the 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.

Table 5.1: Theoretical Bounds for Sampling Algorithms. The Proj-Err Bounds Are Based on a Slightly Weaker $\|A - \pi_B(A)\|_F^2$ Numerator Instead of $\|A - \pi_{B_k}(A)\|_F^2$ One Where We First Enforce B_k Is Rank k. ($\star$) 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 (†) [50]	k/ε^2	$1+\varepsilon\frac{\|A\|_F^2}{\|A-A_k\|_F^2}$ [50]	$\text{nnz}(A)\cdot\ell$
Leverage Sampling	d/ε^2	ε (†)	$(k\log k)/\varepsilon^2$	$1+\varepsilon$[94]	$\text{SVD}(A)+\text{nnz}(A)\cdot\ell$
Deterministic Leverage	ℓ	-	$(k/\eta\varepsilon)^{1/\eta}(\star)$	$1+\varepsilon$ [108]	$\text{SVD}(A)+\text{nnz}(A)\cdot\ell\log\ell$

Table 5.2: Theoretical Bounds for New Sampling Algorithms.

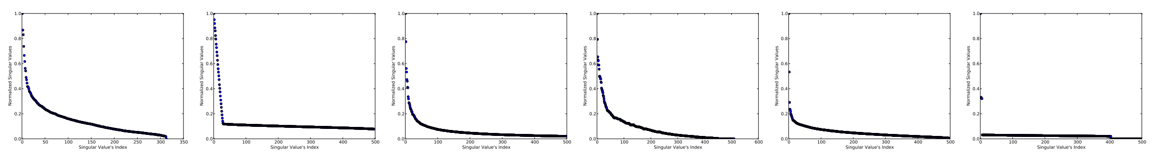
	ℓ	cov-err	ℓ	proj-err	runtime
Priority	d/ε^2	ε	ℓ	-	$\text{nnz}(A)\log\ell$
VarOpt	d/ε^2	ε	ℓ	-	$\text{nnz}(A)\log\ell$

Table 5.3: Theoretical Bounds for Projection Algorithms (Via an ℓ_2 Subspace Embedding) 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.

	ℓ	cov-err	ℓ	proj-err	runtime
Random Projection	d/ε^2[113]	$\varepsilon/\rho(A)$	d/ε^2 [113]	$1+\varepsilon$	$\text{nnz}(A)\cdot\ell$
Fast JLT	d/ε^2 [113]	$\varepsilon/\rho(A)$	d/ε^2 [113]	$1+\varepsilon$	$nd\log d + (d/\varepsilon^2)\log n$ [11]
Hashing	d^2/ε^2 [33, 102]	$\varepsilon/\rho(A)$	d^2/ε^2 [33, 102]	$1+\varepsilon$	$\text{nnz}(A) + n\,\text{poly}\,(d/\varepsilon)$
OSNAP	$d^{1+o(s/\varepsilon)}/\varepsilon^2$ [102]	$\varepsilon/\rho(A)$	$d^{1+o(s/\varepsilon)}/\varepsilon^2$ [102]	$1+\varepsilon$	$\text{nnz}(A)\cdot s + n\,\text{poly}\,(d/\varepsilon)$

Table 5.4: Dataset Statistics.

DataSet	**# datapoints**	**# attributes**	**rank**	**numeric rank**	**nnz%**	**excess kurtosis**
Birds	11789	312	312	12.50	100	1.72
Random Noisy	10000	500	500	14.93	100	0.95
CIFAR-10	60000	3072	3072	1.19	99.75	1.34
Connectus	394792	512	512	4.83	0.0055	17.60
Spam	9324	499	499	3.25	0.07	3.79
Adversarial	10000	500	500	1.69	100	5.80



(a) Birds (b) Random Noisy (c) CIFAR-10 (d) ConnectUS (e) Spam (f) Adversarial

Figure 5.1: Singular values distribution for datasets in Table 5.4. The x-axis is singular value index, and the y-axis shows normalized singular values such that the highest singular value is one, i.e., each value divided by largest singular value of dataset

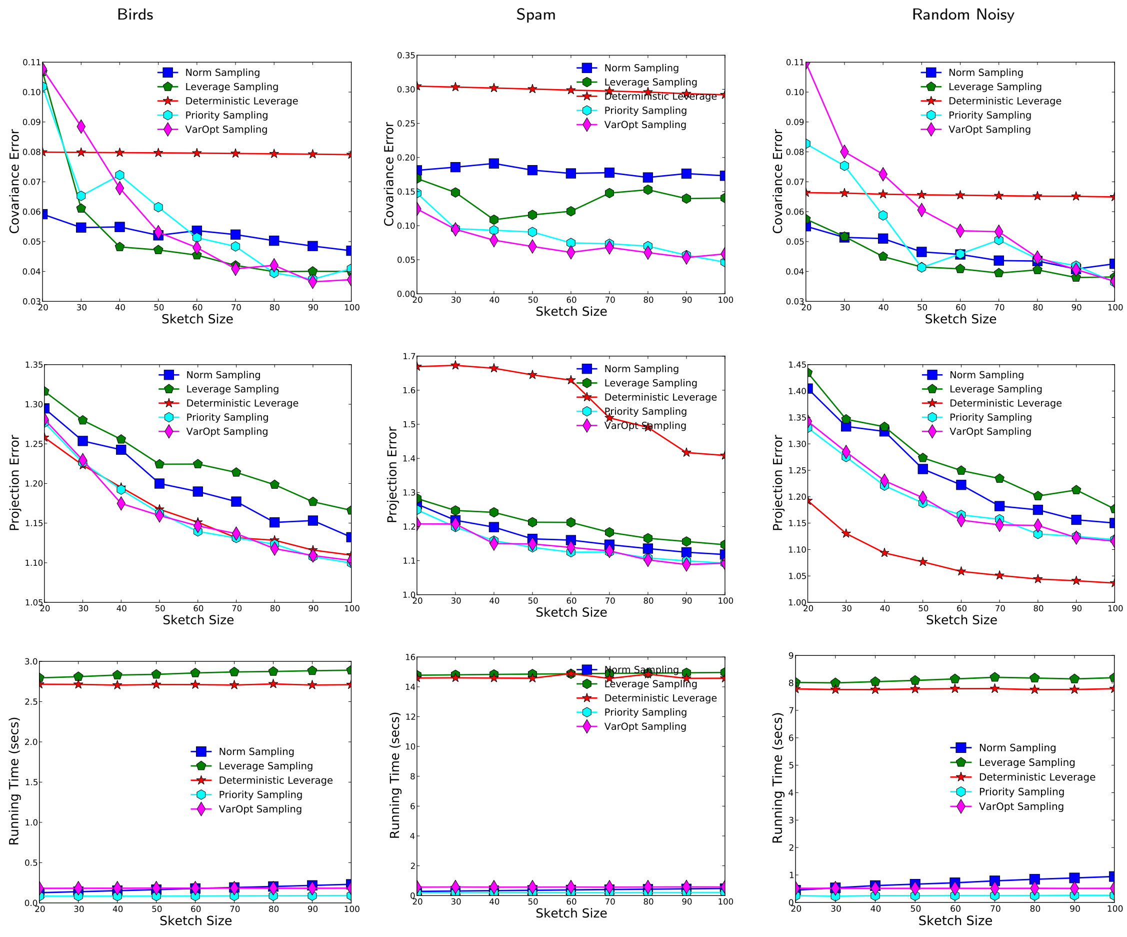


Figure 5.2: Sampling algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

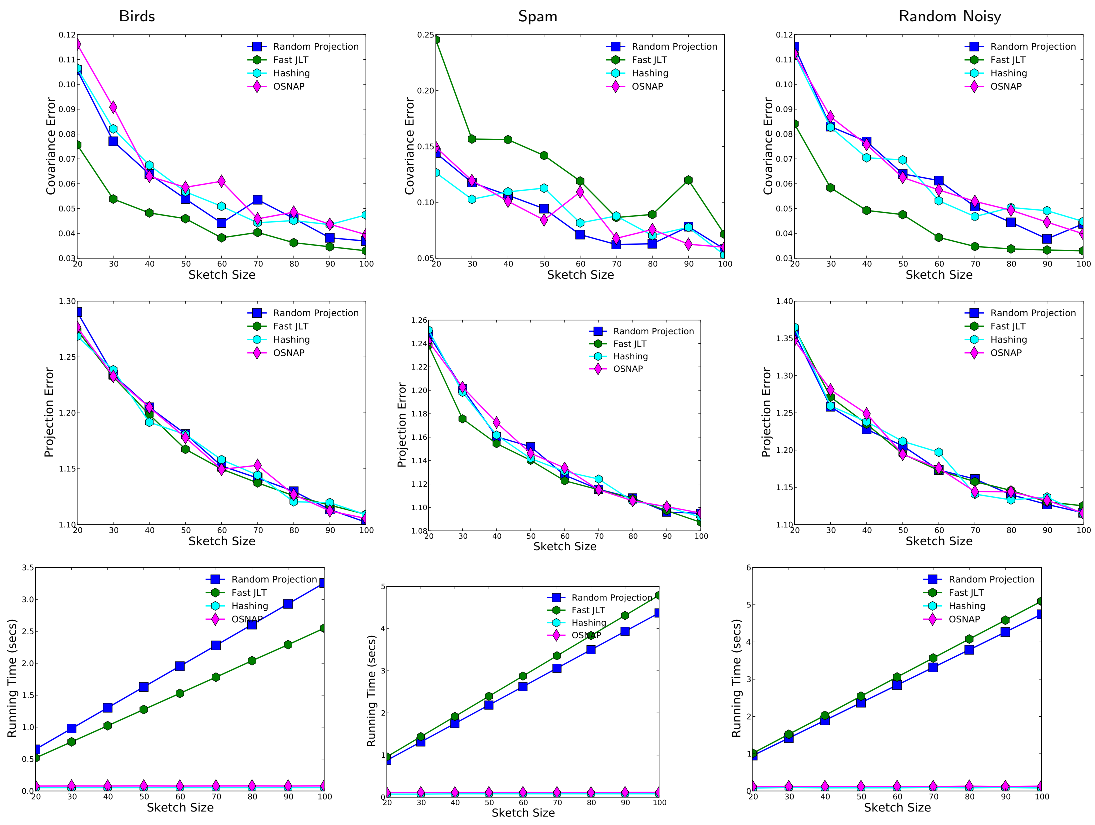


Figure 5.3: Random projection algorithms on Birds(left), Spam(middle), and Random Noisy (30)(right).

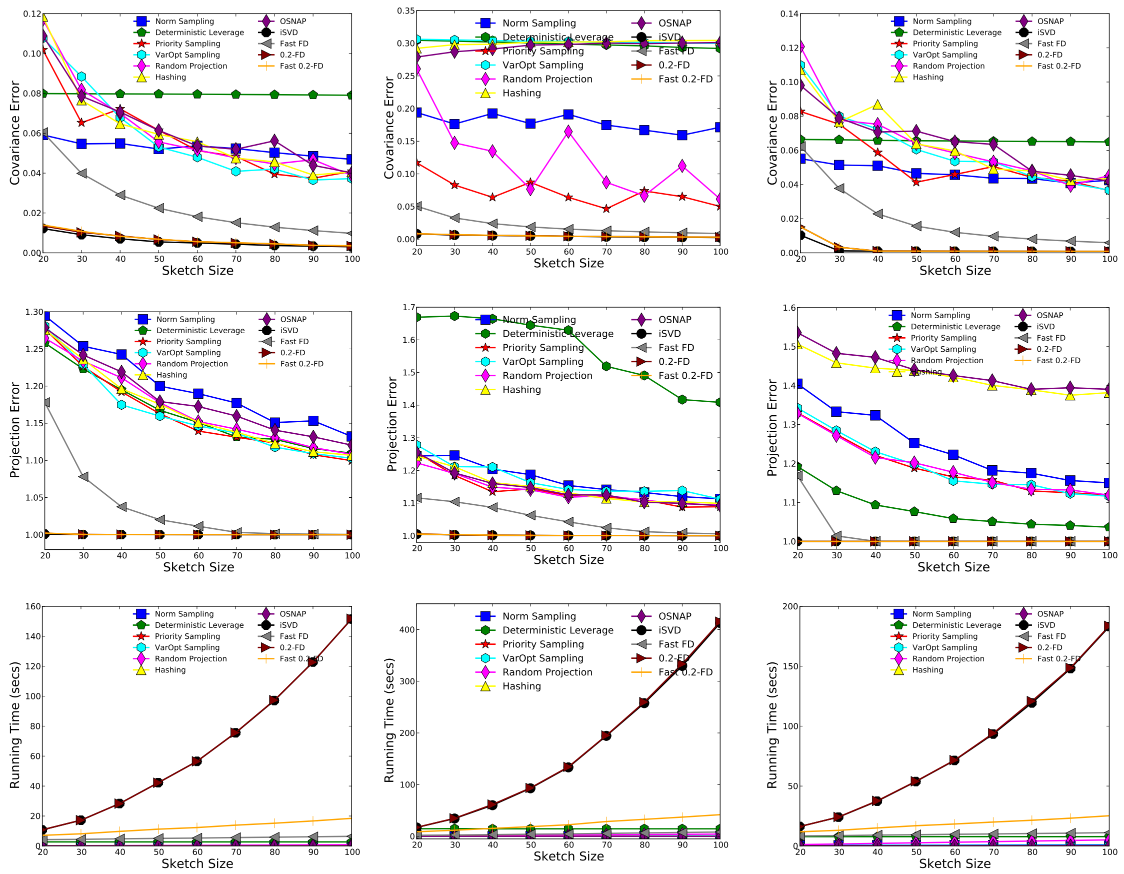


Figure 5.4: Leading algorithms on Birds(left), Spam(middle), and Random Noisy(30)(right).

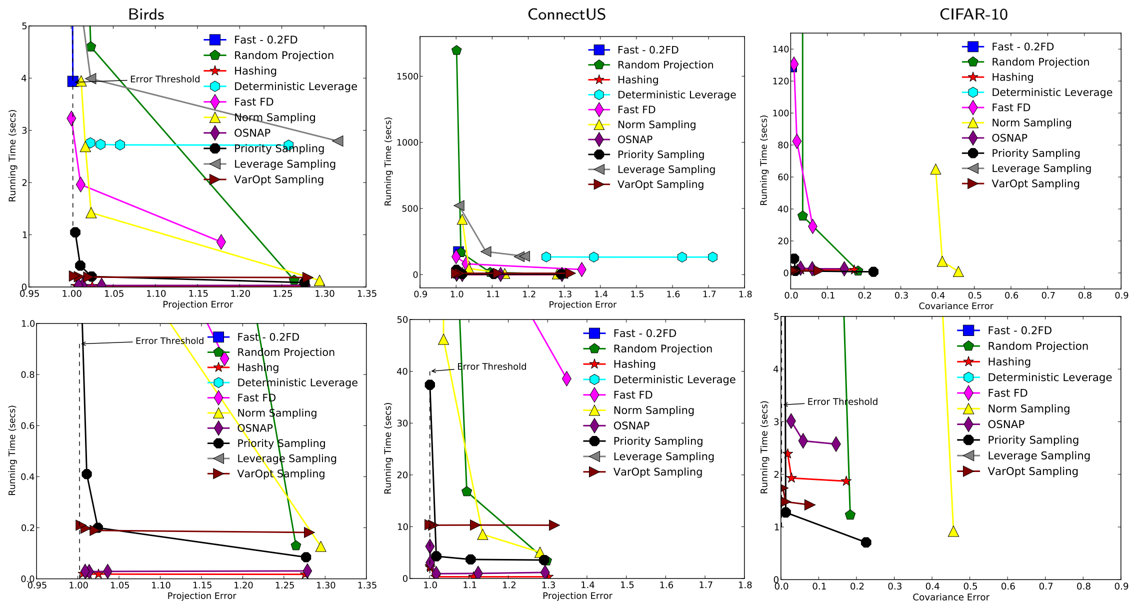


Figure 5.5: Projection error versus time on Birds and ConnectUS as well as Covariance error versus time on CIFAR-10. The second line shows close-ups

CHAPTER 6

SPARSE FREQUENT DIRECTIONS ALGORITHM

In this chapter, we describe a sparse version of FrequentDirections that makes it suitable for processing sparse matrices in runtime proportional to the density of the matrix rather than its ambient dimensions.

6.1 Motivation

Undeniably, many large matrices are sparse; most of their entries are zero. The work of [47] argues that typical term-document matrices are sparse; documents contain no more than 5% of all words. On wikipedia, most words appear on only a small constant number of pages. Similarly, in recommendation systems on average a user rates or interacts with a small fraction of the available items: less than 6% in some user-movies recommendation tasks [15] and much fewer in physical purchases or online advertising. As such, most of these datasets are stored as sparse matrices.

There exist several techniques for producing low rank approximations of sparse matrices whose running time is $O(\mathrm{nnz}(A)\,\mathrm{poly}(k, 1/\varepsilon))$ for some error parameter $\varepsilon \in (0, 1)$. Here $\mathrm{nnz}(A)$ denotes the number of non-zeros in the matrix A. Examples include the power method [74], random projection techniques [113], projection-hashing [33], and instances of column selection techniques [50].

However, for FrequentDirections [92]) there is no known way to take advantage of the sparsity of the input matrix. While it is deterministic and its space-error bounds are known to be optimal for dense matrices in the row-update model [67], it runs in $O(nd\ell)$ time to produce a sketch of size $\ell \times d$. In particular, it maintains a sketch with ℓ rows and updates it iteratively over a stream, periodically invoking a full SVD which requires $O(d\ell^2)$ time.

6.2 Existing Sparse Matrix Sketching Methods

In this section, we review exsiting techniques for sketching sparse matrices.

6.2.1 Row/Column Sampling techniques

These methods are not typically streaming, nor running in input sparsity time. The only method of this group which achieves both is [50] by Drineas *et al.* which uses reservoir sampling to become streaming. They select $O(k/\varepsilon^2)$ columns proportional to their squared norm and achieve the Frobenius norm error bound $\|A - \pi_{B_k}(A)\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon\|A\|_F^2$ with time complexity of $O((k^2/\varepsilon^4)(d + k/\varepsilon^2) + \text{nnz}(A))$. In addition, they show that the spectral norm error bound $\|A - \pi_{B_k}(A)\|_2^2 \leq \|A - A_k\|_2^2 + \varepsilon\|A\|_F^2$ holds if one selects $O(1/\varepsilon^2)$ columns. Rudelson *et al.* [112] improved the latter error bound to $\|A - \pi_{B_k}(A)\|_2^2 \leq \|A - A_k\|_2^2 + \varepsilon\|A\|_2^2$ by selecting $O(r/\varepsilon^4 \log(r/\varepsilon^4))$ columns, where $r = \|A\|_F^2/\|A\|_2^2$ is the numeric rank of A. Note that in the result by [50], one would need $O(r^2/\varepsilon^2)$ columns to obtain the same bound.

Another similar line of work is the CUR factorization [25, 48, 51, 54, 94] where methods select c columns and r rows of A to form matrices $C \in \mathbb{R}^{n\times c}$, $R \in \mathbb{R}^{r\times d}$ and $U \in \mathbb{R}^{c\times r}$, and constructs the sketch as $B = CUR$. The only instance of this group that runs in input sparsity time is [25] by Boutsidis and Woodruff, where they select $r = c = O(k/\varepsilon)$ rows and columns of A and construct matrices C, U and R with $\text{rank}(U) = k$ such that with constant probability $\|A - CUR\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$. Their algorithm runs in $O(\text{nnz}(A)\log n + (n + d)\,\text{poly}(\log n, k, 1/\varepsilon))$ time.

6.2.2 Random Projection Techniques

These techniques [93, 105, 113, 117] operate data-obliviously and maintain a $r \times d$ matrix $B = SA$ using a $r \times n$ random matrix S which has the Johnson-Lindenstrauss Transform (JLT) property [96]. Random projection methods work in the streaming model, are computationally efficient, and sufficiently accurate in practice [42]. The state-of-the-art method of this approach is by Clarkson and Woodruff [33] which was later improved slightly in [102]. It uses a hashing matrix S with only one non-zero entry in each column. Constructing this sketch takes only $O(\text{nnz}(A) + n \cdot \text{poly}(k/\varepsilon) + \text{poly}(dk/\varepsilon))$ time, and guarantees that for any unit vector x that $(1 - \varepsilon)\|Ax\| \leq \|Bx\| \leq (1 + \varepsilon)\|Ax\|$. For these sparsity-efficient sketches using $r = O(d^2/\varepsilon^2)$ also guarantees that $\|A - \pi_B(A)\|_F \leq (1+\varepsilon)\|A - A_k\|_F$.

6.2.3 Power Method Based Techniques

Another group of methods that run in input sparsity time are techniques based on Power Method [74] which efficiently computes the singular vectors and values of a matrix. Recent results give very strong approximation guarantees for block power method techniques [110][126][93][75]. Several variants of this algorithm were studied under different names in the literature, e.g., Simultaneous Iteration, Subspace Iteration, or Orthogonal Iteration [74]. We refer to this group of algorithms collectively as SimultaneousIteration. A generic version of SimultaneousIteration for rectangular matrices is described in Algorithm 13.

Algorithm 13 SimultaneousIteration

```
  Input: $A \in \mathbb{R}^{n\times d}$, rank $k \leq \min(n,d)$, and error $\varepsilon \in (0,1)$
  $q = \Theta(\log(n/\varepsilon)/\varepsilon)$
  $G \sim \mathcal{N}(0,1)^{d\times k}$
  $Z = \text{GramSchmidt}(A(A^TA)^q G)$
  return $Z$                                        # $Z \in \mathbb{R}^{n\times k}$
```

While this algorithm was already analyzed by [74], the proofs of [75, 100, 110, 123] manage to prove stable results that hold for any matrix independent of spectral gap issues.

SimultaneousIteration (Algorithm 13) guarantees the three following error bounds with high probability:

1. Frobenius norm error bound: $\|A - ZZ^TA\|_F \leq (1+\varepsilon)\|A - A_k\|_F$

2. Spectral norm error bound: $\|A - ZZ^TA\|_2 \leq (1+\varepsilon)\|A - A_k\|_2$

3. Per vector error bound: $|u_i^TAA^Tu_i - z_i^TAA^Tz_i| \leq \varepsilon\sigma_{k+1}^2$ for all i. Here u_i denotes the ith left singular vector of A, and σ_{k+1} is the $(k+1)$th singular value of A, and z_i is the ith column of the matrix Z returned by SimultaneousIteration.

In addition, for a constant ε, SimultaneousIteration runs in $\tilde{O}(\text{nnz}(A))$ time.

6.3 Sparse Frequent Directions

The SparseFrequentDirections (SFD) algorithm is described in Algorithm 14, and is an extension of FrequentDirections to sparse matrices.

Algorithm 14 SparseFrequentDirections

```
Input: $A \in \mathbb{R}^{n \times d}$, an integer $\ell \le d$, failure probability $\delta$
$B = 0^{\ell \times d}$,  $A' = 0^{0 \times d}$
for $a \in A$ do
    $A' = [A'; a]$
    if $\mathrm{nnz}(A') \ge \ell d$ or $\mathrm{rows}(A') = d$ then
        $B' = \mathrm{BoostedSparseShrink}(A', \ell, \delta)$
        $B = \mathrm{DenseShrink}([B; B'], \ell)$
        $A' = 0^{0 \times d}$
return $B$
```

It receives the rows of an input matrix A in a streaming fashion and maintains a sketch B of ℓ rows. Initially B is empty. On receiving rows of A, SFD stores non-zeros in a buffer matrix A'. The buffer is deemed full when it contains ℓd non-zeros or d rows. SFD then calls BoostedSparseShrink to produce its sketch matrix B' of size $\ell \times d$. Then, it updates its ongoing sketch B of the entire stream by merging it with the (dense) sketch B' using DenseShrink.

Algorithm 15 BoostedSparseShrink

```
Input: $A' \in \mathbb{R}^{m \times d}$, integer $\ell \le m$, failure probability $\delta$
while True do
    $B' = \mathrm{SparseShrink}(A', \ell)$
    $\Delta = (\|A'\|_F^2 - \|B'\|_F^2)/\alpha\ell$    for $\alpha = 6/41$
    if VerifySpectral($(A'^T A' - B'^T B')/(\Delta/2), \delta$) then
        return $B'$
```

BoostedSparseShrink amplifies the success probability of another algorithm SparseShrink in Algorithm 16. SparseShrink runs SimultaneousIteration instead of a full SVD to take advantage of the sparsity of its input A'. However, as we will discuss, by itself SparseShrink has a high failure probability. Thus we use BoostedSparseShrink which keeps running SparseShrink and probabilistically verifying the correctness of its result using VerifySpectral, until it decides that the result is correct with high enough probability.

Algorithm 16 SPARSESHRINK

```
Input: $A' \in \mathbb{R}^{m \times d}$, an integer $\ell \leq m$
$Z = \text{SimultaneousIteration}(A', \ell, 1/4)$
$P = Z^T A'$,   $[H, \Lambda, V] = \text{SVD}(P, \ell)$
$\tilde{\Lambda} = \sqrt{\Lambda^2 - \lambda_\ell^2 I_\ell}$
$B' = \tilde{\Lambda} V^T$
return $B'$
```

Each of DenseShrink, SparseShrink, and BoostedSparseShrink produces sketch matrices of size $\ell \times d$.

Algorithm 17 DENSESHRINK

```
Input: $A \in \mathbb{R}^{m \times d}$, an integer $\ell \leq m$
$[H, \Lambda, V] = \text{SVD}(A, \ell)$
$\tilde{\Lambda} = \sqrt{\Lambda^2 - \lambda_\ell^2 I_\ell}$
$B = \tilde{\Lambda} V^T$
Return $B$
```

We prove that SparseFrequentDirections satisfies the following theorem:

Theorem 6.3.1 (main result). *Given a sparse matrix $A \in \mathbb{R}^{n \times d}$ and an integer $\ell \leq d$, SparseFrequentDirections computes a small sketch $B \in \mathbb{R}^{\ell \times d}$ such that with probability at least $1 - \delta$ for $\alpha = 6/41$ and any $0 \leq k < \alpha\ell$,*

$$\|A^T A - B^T B\|_2 \leq \frac{1}{\alpha\ell - k}\|A - A_k\|_F^2$$

and

$$\|A - \pi_{B_k}(A)\|_F^2 \leq \frac{\ell}{\ell - k/\alpha}\|A - A_k\|_F^2.$$

The total memory footprint of the algorithm is $O(d\ell)$ and its expected running time is

$$O\left(\text{nnz}(A)\ell\log(d) + \text{nnz}(A)\log(n/\delta) + n\ell^2 + n\ell\log(n/\delta)\right).$$

It is convenient to set $\ell = \lceil 1/\varepsilon\alpha + k/\alpha \rceil$ which yields $\|A^T A - B^T B\|_2 \leq \varepsilon\|A - A_k\|_F^2$ or to set $\ell = \lceil k/\varepsilon\alpha + k/\alpha \rceil$ which yields $\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\varepsilon)\|A - A_k\|_F^2$. Moreover, it is reasonable to expect the number of non-zeros per row in A to be larger than ℓ and for n, d and $1/\delta$ to be at most polynomial in one another. In this setting, the running time is dominated by $O\left(\text{nnz}(A)\ell \log(d)\right)$.

6.3.1 Success Probability

SparseShrink, described in Algorithm 16, calls SimultaneousIteration to approximate the top rank ℓ subspace of A'. As SimultaneousIteration is randomized, it fails to converge to a good subspace when the initial choice of the random matrix G does not sufficiently align with the top ℓ singular vectors of A' (see Algorithm 13). This occurs with probability at most $\rho_\ell = O(1/\sqrt{\ell})$. In Section 6.3.3.1, we prove that with probability of at least $1 - \rho_\ell$ that SparseShrink satisfies the three properties required for Lemma 4.1.1 using $\alpha = 6/41$ and $\Delta = 41/8\ s_\ell^2$, but replacing Property 2 with a stronger version

- Property 2 (strengthened): $\|A'^T A' - B'^T B'\|_2 \leq (\Delta/2) = 41/16\ s_\ell^2$

where s_ℓ denotes the ℓth singular value of A'.

However, for the proof of SparseFrequentDirections we require that *all* SparseShrink runs be successful. The failure probability of SparseShrink, which is upper bounded by $O(1/\sqrt{\ell})$, is high enough that a simple union bound would not give a meaningful bound on the failure probability of SparseFrequentDirections. We therefore reduce the failure probability of each BoostedSparseShrink, by wrapping each call of SparseShrink in the verifier VerifySpectral. If VerifySpectral does not verify the correctness, then it reruns SparseShrink and tries again until it can verify it. But to perform this verification efficiently, we need to loosen the definition of correctness. In particular, we say SparseShrink is successful if the sketch B' computed from its output satisfies $\|A'^T A' - B'^T B'\|_2 \leq \Delta$ (the original Property 2 specification in Section 4.1), where $\Delta = (\|A'\|_F^2 - \|B'\|_F^2)/\alpha\ell$. Combining the two inequalities through Δ, a successful run implies that $\|A'^T A' - B'^T B'\|_2 \leq (\|A'\|_F^2 - \|B'\|_F^2)/\alpha\ell$. VerifySpectral verifies the success of the algorithm by approximating the spectral norm of $(A'^T A' - B'^T B')/(\Delta/2)$; it does so by running the power method for $c \cdot \log(d/\delta_i)$ steps for some constant c.

Algorithm 18 VerifySpectral

```
**Initialization** persistent $i = 0$ ($i$ retains its state between invocations of this method)
**Input**: Matrix $C \in \mathbb{R}^{d\times d}$, failure probability $\delta$
$i = i + 1$ and $\delta_i = \delta/2i^2$
Pick $x$ uniformly at random from the unit sphere in $\mathbb{R}^d$.
**if** $\|C^{c\cdot\log(d/\delta_i)}x\| \leq 1$ **return** TRUE
**else return** FALSE
```

Lemma 6.3.1. *The VerifySpectral algorithm returns* TRUE *if* $\|C\|_2 \leq 1$. *If* $\|C\|_2 \geq 2$ *it returns* FALSE *with probability at least* $1 - \delta_i$.

Proof. If $\|C\| \leq 1$ than $\|C^{c\cdot\log(d/\delta_i)}x\| \leq \|C\|^{c\cdot\log(d/\delta_i)}\|x\| \leq 1$. If $\|C\| \geq 2$, consider execution i of the method. Let v_1 denote the top singular vector of C. Then $\|C^{c\cdot\log(d/\delta_i)}x\| \geq |\langle v_1, x\rangle|2^{c\cdot\log(d/\delta_i)} \geq 1$, for some constant c as long as $|\langle v_1, x\rangle| = \Omega(\text{poly}(\delta_i/d))$. Let $\Phi(t')$ denote the density function of the random variable $t' = \langle v_1, x\rangle$. Then $\mathbf{Pr}[|\langle v_1, x\rangle| \leq t] = \int_{-t}^{t}\Phi(t')dt' \leq 2t\Phi(0) = O(t\sqrt{d})$. Setting the failure probability to be at most δ_i, we conclude that $|\langle v_1, x\rangle| = \Omega(\delta_i/\sqrt{d})$ with probability at least $1 - \delta_i$. □

Therefore, VerifySpectral fails with probability at most δ_i during execution i. If any of VerifySpectral runs fail, BoostedSparseShrink and hence SparseFrequentDirections potentially fail. Taking the union bound over all invocations of VerifySpectral we obtain that SparseFrequentDirections fails with probability at most $\sum \delta_i \leq \sum_{i=1}^{\infty} \delta/2i^2 \leq \delta$, hence it succeeds with probability at least $1 - \delta$.

6.3.2 Space Usage and Runtime Analysis

Throughout this manuscript we assume the constant-word-size model. Integers and floating point numbers are represented by a constant number of bits. Random access into memory is assumed to require $O(1)$ time. In this model, multiplying a sparse matrix A' by a dense vector requires $O(\text{nnz}(A'))$ operations and storing A' requires $O(\text{nnz}(A'))$ bits of memory.

Fact 6.3.1. *The total memory footprint of SparseFrequentDirections is* $O(d\ell)$.

Proof. It is easy to verify that, except for the buffer matrix A', the algorithm only manipulates $\ell \times d$ matrices; in particular, observe that the $(\text{rows}(A') = d)$ condition in

SparseFrequentDirections ensures that $m = d$ in SparseShrink, and in DenseShrink also $m = 2\ell$. Each of these $\ell \times d$ matrices clearly require at most $O(d\ell)$ bits of memory. The buffer matrix A' contains at most $O(d\ell)$ non-zeros and therefore does not increase the space complexity of the algorithm. □

We turn to bounding the expected runtime of SparseFrequentDirections which is dominated by the cumulative running times of DenseShrink and BoostedSparseShrink. Denote by T the number of times they are executed. It is easy to verify $T \leq \text{nnz}(A)/d\ell + n/d$. Since DenseShrink runs in $O(d\ell^2)$ time deterministically, the total time spent by DenseShrink through T iterations is $O(Td\ell^2) = O(\text{nnz}(A)\ell + n\ell^2)$.

The running time of BoostedSparseShrink is dominated by those of SparseShrink and VerifySpectral, and its expected number of iterations. Note that, in expectation, they are each executed on any buffer matrix A'_i a small constant number of times because VerifySpectral succeeds with probability (much) greater than 1/2. For asymptotic analysis it is identical to assuming they are each executed once.

Note that the running time of SparseShrink on A'_i is $O(\text{nnz}(A'_i)\ell\log(d))$. Since $\sum_i \text{nnz}(A'_i) = \text{nnz}(A)$ we obtain a total running time of $O(\text{nnz}(A)\ell\log(d))$. The ith execution of VerifySpectral requires $O(d\ell\log(d/\delta_i))$ operations. This, because it multiplies $A'^TA' - B'^TB'$ by a single vector $O(\log(d/\delta_i))$ times, and both $\text{nnz}(A') \leq O(d\ell)$ and $\text{nnz}(B') \leq d\ell$. In expectation VerifySpectral is executed $O(T)$ times. Therefore total running time of it is

$$\begin{aligned} O(d\ell\sum_{i=1}^{O(T)}\log(d/\delta_i)) &= O(d\ell\sum_{i=1}^{O(T)}\log(di^2/\delta)) \\ &= O(Td\ell\log(Td/\delta)) = O((\text{nnz}+n\ell)\log(n/\delta)). \end{aligned}$$

Combining the above contributions to the total running time of the algorithm we obtain the following fact.

Fact 6.3.2. *Algorithm SparseFrequentDirections runs in expected time of*

$$O(\text{nnz}(A)\ell\log(d) + \text{nnz}(A)\log(n/\delta) + n\ell^2 + n\ell\log(n/\delta)).$$

6.3.3 Error Analysis

We turn to proving the error bounds of Theorem 6.3.1. Our proof is divided into three parts. We first show that SparseShrink obtains the three properties needed for Lemma 4.1.1 with probability at least $1 - \rho_\ell$, and with the constraint on Property 2 strengthed by a factor $1/2$. Then we show how loosening Property 2 back to its original bound enables BoostedSparseShrink to succeed with probability $1 - \delta_i$ for some $\delta_i \ll \rho_\ell$. Finally we show that due to the mergeability of FrequentDirections 3.1.3, discussed in Section 3.1.3, the SparseFrequentDirections algorithm obtains the same error guarantees as BoostedSparseShrink with probability $1 - \delta$ for a small δ of our choice.

In what follows, we mainly consider only a single execution of SparseShrink or BoostedSparseShrink and let s_ℓ and u_ℓ denote the ℓth singular value and ℓth left singular vector of A', respectively.

6.3.3.1 Error Analysis: SparseShrink

Here we show that with probability at least $1 - \rho_\ell$ that B' computed from SparseShrink(A', ℓ) satisfies the three properties discussed in Section 4.1 required for Lemma 4.1.1.

- Property 1: For any unit vector $x \in \mathbb{R}^d$, $\|A'x\|^2 - \|B'x\|^2 \geq 0$,
- Property 2 (strengthened): For any unit vector $x \in \mathbb{R}^d$, $\|A'x\|^2 - \|B'x\|^2 \leq \Delta/2 = (41/16)s_\ell^2$,
- Property 3: $\|A'\|_F^2 - \|B'\|_F^2 \geq \ell\alpha\Delta = \ell(3/4)s_\ell^2$.

Lemma 6.3.2. *Property 1 holds deterministically for SparseShrink: $\|A'x\|^2 - \|B'x\|^2 \geq 0$ for all x.*

Proof. Let $P = Z^T A'$ be as defined in SparseShrink. Consider an arbitrary unit vector $x \in \mathbb{R}^d$, and let $y = A'x$.

$$\|A'x\|^2 - \|Px\|^2 = \|A'x\|^2 - \|Z^T A'x\|^2 = \|y\|^2 - \|Z^T y\|^2 = \|(I - ZZ^T)y\|^2 \geq 0$$

and

$$\|Px\|^2 - \|B'x\|^2 = \lambda_\ell^2 \sum_{i=1}^{\ell} \langle x, v_i \rangle^2 \geq 0,$$

therefore $\|A'x\|^2 - \|B'x\|^2 = (\|A'x\|^2 - \|Px\|^2) + (\|Px\|^2 - \|B'x\|^2) \geq 0.$ □

Lemma 6.3.3. *With probability at least $1 - \rho_\ell$, Property 2 holds for SparseShrink: for any unit vector $x \in \mathbb{R}^d$, $\|A'x\|^2 - \|B'x\|^2 \leq 41/16\ s_\ell^2$.*

Proof. Consider an arbitrary unit vector $x \in \mathbb{R}^d$, and note that

$$\|A'x\|^2 - \|B'x\|^2 = \left(\|A'x\|^2 - \|Px\|^2\right) + \left(\|Px\|^2 - \|B'x\|^2\right).$$

We bound each term individually. The first term is bounded as

$$\begin{aligned}
\|A'x\|^2 - \|Px\|^2 &= x^T(A'^T A' - P^T P)x && (6.1)\\
&\leq \|A'^T A' - P^T P\|_2 && (6.2)\\
&= \|A'^T A' - A'^T Z Z^T A'\|_2 && (6.3)\\
&= \|A'^T (I - ZZ^T) A'\|_2 && (6.4)\\
&= \|A'^T (I - ZZ^T)^T (I - ZZ^T) A'\|_2 && (6.5)\\
&= \|(I - ZZ^T) A'\|_2^2 && (6.6)\\
&\leq 25/16\ s_{\ell+1}^2 \leq 25/16\ s_\ell^2. && (6.7)
\end{aligned}$$

where transition 5 is true because $(I - ZZ^T)$ is a projection. Transition 7 also holds by the spectral norm error bound of [100] for $\varepsilon = 1/4$. To bound the second term, note that $\|Px\| = \|Z^T A'x\| = \|\Lambda V^T x\|$, since $[H, \Lambda, V] = \text{SVD}(P, \ell)$ as defined in SparseShrink.

$$\|Px\|^2 - \|B'x\|^2 = \sum_{i=1}^{\ell} \lambda_i^2 \langle x, v_i\rangle^2 - \sum_{i=1}^{\ell} \tilde{\lambda}_i^2 \langle x, v_i\rangle^2 =$$
$$\sum_{i=1}^{\ell} (\lambda_i^2 - \tilde{\lambda}_i^2)\langle x, v_i\rangle^2 = \sum_{i=1}^{\ell} \lambda_\ell^2 \langle x, v_i\rangle^2 \leq \lambda_\ell^2 \leq s_\ell^2,$$

where last inequality follows by the Courant-Fischer min-max principle, i.e., as λ_ℓ is the ℓth singular value of the projection of A' onto Z, then $\lambda_\ell \leq s_\ell$. Summing the two terms yields $\|A'x\|^2 - \|B'x\|^2 \leq 41/16\ s_\ell^2$. □

The original bound $\|A'^T A' - B'^T B'\|_2 \leq \Delta = 41/8\ s_\ell^2$ discussed in Section 6.3.1 is also immediately satisfied.

Lemma 6.3.4. *With probability at least $1 - \rho_\ell$, Property 3 holds for SparseShrink: $\|A'\|_F^2 - \|B'\|_F^2 \geq \ell(3/4)s_\ell^2$.*

Proof.

$$\|A'\|_F^2 - \|P\|_F^2 = \|A'\|_F^2 - \|Z^T A'\|_F^2 = \|A' - ZZ^T A'\|_F^2 \geq 0$$

In addition,

$$\|P\|_F^2 - \|B'\|_F^2 = \ell \lambda_\ell^2 \geq \ell(3/4)s_\ell^2 \,.$$

The last inequality holds by the per vector error bound of [100] for $i = \ell$ and $\varepsilon = 1/4$, i.e., $|u_\ell^T A' A'^T u_\ell - z_\ell^T A' A'^T z_\ell| = |s_\ell^2 - \lambda_\ell^2| \leq 1/4 s_{\ell+1}^2 \leq 1/4 s_\ell^2$, which means $\lambda_\ell^2 \geq 3/4\ s_\ell^2$. Therefore

$$\|A'\|_F^2 - \|B'\|_F^2 = (\|A'\|_F^2 - \|P\|_F^2) + (\|P\|_F^2 - \|B'\|_F^2) \geq \ell(3/4)s_\ell^2.$$

□

6.3.3.2 Error Analysis: BoostedSparseShrink

We now consider the BoostedSparseShrink algorithm, and the looser version of Property 2 (the original version) as

- Property 2: For any unit vector $x \in \mathbb{R}^d$, $\|A'x\|^2 - \|B'x\|^2 \leq \Delta = (41/8)s_\ell^2$.

By invoking VerifySpectral($(A'^T A' - B'^T B')/(\Delta/2), \delta$), then VerifySpectral always returns TRUE if $\|A'^T A' - B'^T B'\|_2 \leq \Delta/2$ (as is true of the input with probability at least $1 - \rho_\ell$ by Lemma 6.3.3), and VerifySpectral catches a failure event where $\|A'^T A' - B'^T B'\|_2 \geq \Delta$ with probability at least $1 - \delta_i$ by Lemma 6.3.1. As discussed in Section 6.3.1 all invocations of VerifySpectral succeed with probability at most $1 - \delta$, hence all runs of BoostedSparseShrink succeed and satisfy Property 2 (as well as Properties 1 and 3) with $\alpha = 6/41$ and $\Delta = 41/8\ s_\ell^2$, and with probability at least $1 - \delta$. Finally, we can invoke the mergeability property of *FrequentDirections* [91] and Lemmas 4.1.1 and 4.1.2 to obtain the error bounds in our main result, Theorem 6.3.1.

6.4 Experimental Evaluation

In this section we empirically validate that SparseFrequentDirections matches (and often improves upon) the accuracy of FrequentDirections, while running significantly faster on sparse real and synthetic datasets.

We do not implement SparseFrequentDirections exactly as described above. Instead we directly call SparseShrink in Algorithm 14 in place of BoostedSparseShrink. The randomized error analysis of SimultaneousIteration indicates that we may occasionally miss a subspace within a call of SimultaneousIteration and hence SparseShrink; but in practice this is not a catastrophic event, and as we will observe, does not prevent SparseFrequentDirections from obtaining small empirical error.

The empirical comparison of FrequentDirections to other matrix sketching techniques is now well-trodden 6.4. FrequentDirections (and, as we observe, by association SparseFrequentDirections) has much smaller error than other sketching techniques which operate in a stream. However, FrequentDirections is somewhat slower by a factor of the sketch size ℓ up to some leading coefficients. We do not repeat these comparison experiments here.

6.4.1 Setup

We ran all the algorithms under a common implementation framework to test their relative performance as accurately as possible. We ran the experiments on an Intel(R) Core(TM) 2.60 GHz CPU with 64GB of RAM running Ubuntu 14.04.3. All algorithms were coded in C, and compiled using gcc 4.8.4. All linear algebra operations on dense matrices (such as SVD) invoked those implemented in LAPACK.

6.4.2 Datasets

We compare the performance of the two algorithms on both synthetic and real datasets. Each dataset is an $n \times d$ matrix A containing n datapoints in d dimensions.

The real dataset is part of the 20 Newsgroups dataset [88], which is a collection of approximately 20,000 documents, partitioned across 20 different newsgroups. However we use the 'by date' version of the data, where features (columns) are tokens and rows correspond to documents. This data matrix is a zero-one matrix with 11,314 rows and 117,759 columns. In our experiment, we use the transpose of the data and picked the first $d = 3000$ columns, hence the subset matrix has $n = 117{,}759$ rows and $d = 3000$ columns; roughly 0.15% of the subset matrix is non-zeros.

The synthetic data generate n rows i.i.d. Each row receives exactly $z \ll d$ non-zeros (with default $z = 100$ and $d = 1000$), with the remaining entries as 0. The non-zeros are chosen as either 1 or -1 at random. Each non-zero location is chosen without duplicates among the columns. The first $1.5z$ columns (e.g., 150), the "head", have a higher probability of receiving a non-zero than the last $d - 1.5z$ columns, the "tail". The process to place a non-zero first chooses the head with probability 0.9 or the tail with probability 0.1. For whichever set of columns it chooses (head or tail), it places the non-zero uniformly at random among those columns.

6.4.3 Measurements

Each algorithm outputs a sketch matrix B of ℓ rows. For each of our experiments, we measure the efficiency of algorithms against one parameter and keep others fixed at a default value. Table 6.1 lists all parameters along with their default value and the range they vary in for synthetic dataset. We measure the accuracy of the algorithms with respect to:

- Projection Error: $\mathsf{proj\text{-}err} = \|A - \pi_{B_k}(A)\|_F^2 / \|A - A_k\|_F^2$,
- Covariance Error: $\mathsf{cov\text{-}err} = \|A^T A - B^T B\|_2 / \|A\|_F^2$,
- Runtime in seconds.

In all experiments, we have set $k = 10$. Note that proj-err is always larger than 1, and for FrequentDirections and SparseFrequentDirections the cov-err is always smaller than $1/(\frac{6}{41}\ell - k)$ due to our error guarantees.

6.4.4 Observations

By considering Table 6.1 on synthetic data and Figure 6.5 on the real data, we can vary and learn many aspects of the runtime and accuracy of SparseFrequentDirections and FrequentDirections.

6.4.4.1 Runtime

Consider the last row of Table 6.1, the "Runtime" row, and the last column of Figure 6.5. SparseFrequentDirections is clearly faster than FrequentDirections for all datasets, except when the synthetic data becomes dense in the last column of the "Runtime" row, where $d = 1000$ and $\text{nnz per row} = 500$ in the right-most data point. For the default values the improvement is between about a factor of 1.5x and $2x$, but when the matrix is very sparse the improvement is 10x or more. Very sparse synthetic examples are seen in the left data points of the last column, and in the right data points of the second column, of the "Runtime" row.

In particular, these two plots (the second and fourth columns of the "Runtime" row) really demonstrate the dependence of SparseFrequentDirections on $\mathrm{nnz}(A)$ and of FrequentDirections on $n \cdot d$. In the last column, we fix the matrix size n and d, but increase the number of non-zeros $\mathrm{nnz}(A)$; the runtime of FrequentDirections is basically constant, while for SparseFrequentDirections it grows linearly. In the second column, we fix n and $\mathrm{nnz}(A)$, but increase the number of columns d; the runtime of FrequentDirections grows linearly while the runtime for SparseFrequentDirections is basically constant.

These algorithms are designed for datasets with extremely large values of n; yet we only run on datasets with n up to 60,000 in Table 6.1, and 117,759 in Figure 6.5. However, both FrequentDirections and SparseFrequentDirections have runtime that grows linearly with respect to the number of rows (assuming the sparsity is at an expected fixed rate per row for SparseFrequentDirections). This can also be seen empirically in the first column of the "Runtime" row where, after a small start-up cost, both FrequentDirections and SparseFrequentDirections grow linearly as a function of the number of data points n. Hence, it is valid to directly extrapolate these results for datasets of increased n.

6.4.4.2 Accuracy

We will next discuss the accuracy, as measured in Projection Error in the top row of Table 6.1 and left plot of Figure 6.5, and in Covariance Error in the middle row of Table 6.1 and middle plot of Figure 6.5. We observe that both FrequentDirections and SparseFrequentDirections obtain very small error (much smaller than upper bounded by the theory), as has been observed elsewhere [42, 67]. Moreover, the error for SparseFrequentDirections always nearly matches, or improves over FrequentDirections. We can likely attribute this improvement to being able to process more rows in each batch, and hence needing to perform the shrinking operation fewer overall times. The one small exception to SparseFrequentDirections having less Covariance Error than FrequentDirections is for extreme sparse datasets in the leftmost data points of Table 6.1, last column – we attribute this to some peculiar orthogonality of columns with near equal norms due to extreme sparsity.

Table 6.1: Parameter Values

	default	**range**
datapoints (n)	10000	$[10^4 - 6 \times 10^4]$
dimension (d)	1000	$[10^3 - 6 \times 10^3]$
sketch size (ℓ)	50	$[5 - 100]$
nnz per row	100	$[5 - 500]$

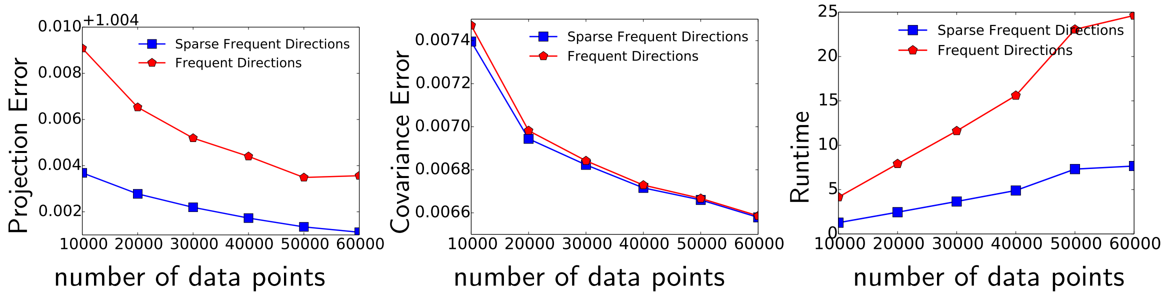


Figure 6.1: Comparing performance of FrequentDirections and SparseFrequentDirections against number of data points (n) on synthetic data. Table 6.1 lists default value of all parameters.

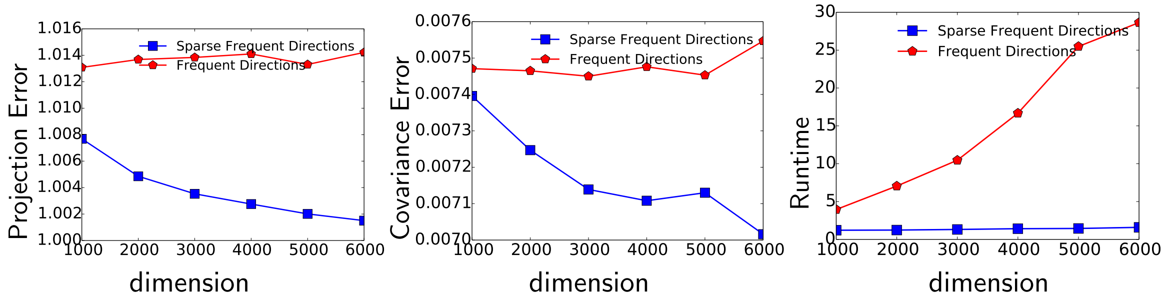


Figure 6.2: Comparing performance of FrequentDirections and SparseFrequentDirections against dimension (d) on synthetic data. Table 6.1 lists default value of all parameters.

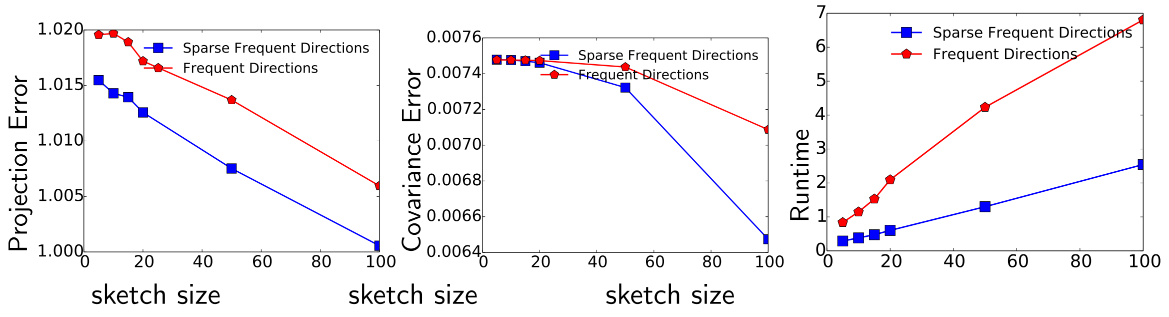


Figure 6.3: Comparing performance of FrequentDirections and SparseFrequentDirections against sketch size (ℓ) on synthetic data. Table 6.1 lists default value of all parameters.

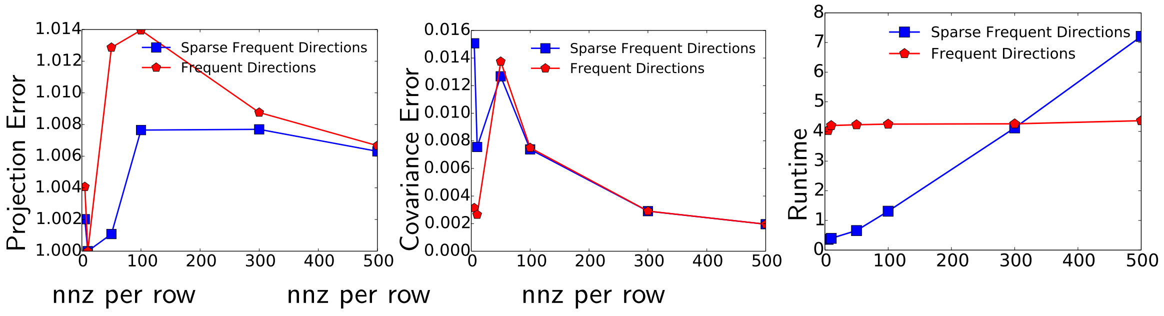


Figure 6.4: Comparing performance of FrequentDirections and SparseFrequentDirections against number of non-zeros per row (nnz) on synthetic data. Table 6.1 lists default value of all parameters.

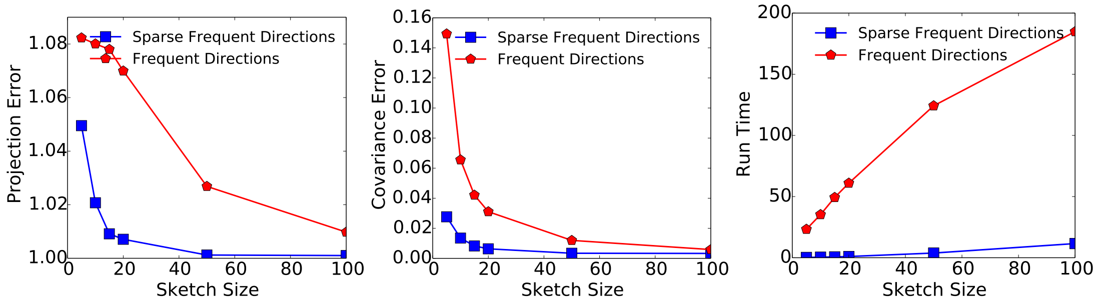


Figure 6.5: Comparing performance of FrequentDirections and SparseFrequentDirections on 20 Newsgroups dataset. We plot Projection Error, Covariance Error, and Runtime as a function of sketch size (ℓ).

CHAPTER 7

DISTRIBUTED STREAMING MATRIX TRACKING

In this chapter, we extend weighted frequent item estimation protocols of Chapter 2 to solve the problem of tracking an approximation to a distributed matrix. This problem can be easily found in distributed network monitoring applications [107], distributed data mining, cloud computing [109], stream mining [79], and log analysis from multiple data centers [59]. Examples of its application include in large scale image analysis, where each row in the matrix corresponds to one image and contains either pixel values or other derived feature values (e.g., 128-dimensional SIFT features). A search engine company has image data continuously arriving at many data centers, or even within a single data center at many nodes in a massive cluster. This forms a distributed matrix and it is critical to obtain excellent, real-time approximation of the distributed streaming image matrix with little communication overhead. Yet another example is for large-scale distributed web crawling or server access log monitoring/mining, where data in the bag-of-words model is a matrix whose columns correspond to words or tags/labels (for textual analysis, e.g., LSI, and/or for learning and classification purpose) and rows correspond to documents or log records (which arrive continuously at distributed nodes).

Despite prior works on distributed streaming model, and distributed matrix computations (e.g., the MadLINQ library [109]), little is known on continuously tracking a matrix approximation in the distributed streaming model. Below, we define the problem formally and introduce our solutions.

7.1 Problem Definition

The formal definition of the distributed matrix tracking problem is as follows:

Definition 7.1.1 (Tracking distributed streaming matrix)**.** *Formally, assume there are m*

distributed sites $S_1, ..., S_m$ and a single designated coordinator C, and each site has a two-way communication channel with C. Assume $A = (a_1, \ldots, a_n, \cdots)$ is an unbounded stream of items, where a_n is a record with d attributes, a row from a matrix in an application. At each time step, we assume the item a_n appears at exactly one of m sites. At the current time t_{now}, let n denote the number of items the system has seen so far. Thus, at time t_{now}, $A = (a_1, \ldots, a_n)$ forms a $n \times d$ distributed streaming matrix. And although we do not place a bound on the number of items, we let N denote the total size of the stream at the time when a query q is performed.

The goal is to continuously track a small approximation of matrix A, while each site must process its incoming elements in streaming fashion. The objective is to minimize the total communication between C and all sites. Formally, for any time instance t_{now} (i.e., for any n), C needs to maintain a smaller matrix $B \in \mathbb{R}^{\ell \times d}$ as an approximation to the distributed streaming matrix $A \in \mathbb{R}^{n \times d}$ such that $\ell \ll n$ and for any unit vector x: $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2$.

Note that the above expression is equivalent to $\|A^T A - B^T B\|_2 \leq \varepsilon \|A\|_F^2$. Thus, the approximation guarantee we preserve shows that the covariance of A is well-approximated by B. And the covariance is the critical property of a matrix that needs to be (approximately) preserved as the basis for most downstream data analysis, e.g., for PCA or LSI. $\square$

As we will show soon in our analysis, it will be convenient to associate a weight with each element defined as the squared norm of the row, i.e., $w_n = \|a_n\|^2$. Hence, for reasons outlined in Section 2.5.1, we assume in our analysis that the squared norm of every row is bounded by a value β.

Our measures of complexity will be the communication cost and the space used at each site to process the stream. We measure communication in terms of the number of messages, where each message is a row of length d, the same as the input stream. Clearly, the space and computational cost at each site and coordinator is also important, but since we show that all proposed protocols can be run as streaming algorithms at each site, and will thus not be space or computation intensive.

7.2 Overview of Protocols

The protocols for matrix tracking mirror those of weighted item frequency tracking. This starts with a similar batched streaming baseline P1. Protocol P2 again reduces the total communication bound, where a global threshold is given for each "direction" instead of the total squared Frobenius norm. Both P1 and P2 are deterministic. Then matrix tracking protocol P3 randomly selects rows with probability proportional to their squared norm and maintains an ε-sample at the coordinator. Using this sample set, we can derive a good approximation.

Given the success of protocols P1, P2, and P3, it is tempting to also extend protocol P4 for item frequency tracking in Section 2.5.6 to distributed matrix tracking. However, unlike the other protocols, we show that the approach described in Algorithm 7 cannot be extended to matrices in any straightforward way while still maintaining the same communication advantages it has (in theory) for the weighted heavy-hitters case.

7.2.1 Distributed Matrix Tracking Protocol 1

We again begin with a batched version of a streaming algorithm, shown as Algorithm 19 and 20. That is we run a streaming algorithm (e.g., Frequent Directions [91], labeled FD, with error $\varepsilon' = \varepsilon/2$) on each site, and periodically send the contents of the memory to the coordinator. Again this is triggered when the total weight (in this case squared norm) has increased by $(\varepsilon/2m)W$.

Algorithm 19 P1: Deterministic Matrix Tracking (at S_i)

```
for $(a_n, w_n)$ in round $j$ do
  Update $B_i \leftarrow \mathsf{FD}_{\varepsilon'}(B_i, a_n)$; and $F_i$ += $\|a_n\|^2$.
  if ($F_i \geq \tau = (\varepsilon/2m)\hat{F}$) then
    Send $(B_i, F_i)$ to coordinator; make $B_i, F_i$ empty.
```

Algorithm 20 P1: Deterministic Matrix Tracking (at C)

```
On input $(B_i, F_i)$:
Update sketch $B \leftarrow \mathsf{Merge}_{\varepsilon'}(B, B_i)$ and $F_C$ += $F_i$.
if ($F_C/\hat{F} > 1 + \varepsilon/2$) then
  Update $\hat{F} \leftarrow F_C$, and broadcast $\hat{F}$ to all sites.
```

As with the similar frequency tracking algorithm, based on Frequent Directions [91] satisfying the mergeable property [9], we can show this maintains at most $\varepsilon\|A\|_F^2$ total error at all times, and requires a total of $O((m/\varepsilon^2)\log(\beta N))$ total rows of communication.

7.2.2 Distributed Matrix Tracking Protocol 2

Again, this protocol is based very closely on a weighted heavy-hitters protocol, this time the one from Section 2.5.4. Each site S_j maintains a matrix B_j of the rows seen so far at this site and not sent to coordinator. In addition, it maintains $\hat{F}$, an estimate of $\|A\|_F^2$, and $F_j = \|B_j\|_F^2$, denoting the total squared Frobenius norm received since its last communication to C about $\hat{F}$. The coordinator C maintains a matrix B approximating A, and $\hat{F}$, an ε-approximation of $\|A\|_F^2$.

Initially each $\hat{F}$ is set to zero for all sites. When site j receives a new row, it calls Algorithm 21, which basically sends $\|B_j x\|^2$ in direction x when it is greater than some threshold provided by the coordinator, if one exists.

Algorithm 21 P2: Deterministic Matrix Tracking (at S_j)

```
$F_j \mathrel{+}= \|a_i\|^2$
**if** $(F_j \geq \frac{\varepsilon}{m}\hat{F})$ **then**
    Send $F_j$ to coordinator; set $F_j = 0$.
Set $B_j \leftarrow [B_j; a_i]$
$[U, \Sigma, V] = \mathrm{SVD}(B_j)$
**for** $((v_\ell, \sigma_\ell)$ such that $\sigma_\ell^2 \geq \frac{\varepsilon}{m}\hat{F})$ **do**
    Send $\sigma_\ell v_\ell$ to coordinator; set $\sigma_\ell = 0$.
$B_j = U\Sigma V^T$
```

Algorithm 22 P2: Deterministic Matrix Tracking (at C)

```
On a scalar message $F_j$ from site $S_j$
Set $\hat{F} \mathrel{+}= F_j$ and #msg $\mathrel{+}= 1$.
**if** (#msg $\geq m$) **then**
    Set #msg $= 0$ and broadcast $\hat{F}$ to all sites.
On a vector message $r = \sigma v$: append $B \leftarrow [B; r]$
```

On the coordinator side, it either receives a vector form message σv, or a scalar message F_j. For a scalar F_j, it adds it to $\hat{F}$. After at most m such scalar messages, it broadcasts $\hat{F}$ to all sites. For vector message $r = \sigma v$, the coordinator updates B by appending r to $B \leftarrow [B; r]$. The coordinator's protocol is summarized in Algorithm 22.

Lemma 7.2.1. *At all times the coordinator maintains B such that for any unit vector x*

$$\|Ax\|^2 - \varepsilon\|A\|_F^2 \leq \|Bx\|^2 \leq \|Ax\|^2 \tag{7.1}$$

Proof. To prove this, we also need to show it maintains another property on the total squared Frobenius norm:

$$(1-2\varepsilon)\|A\|_F^2 < \hat{F} \leq \|A\|_F^2. \tag{7.2}$$

This follows from the analysis in Section 2.5.4 since the squared Frobenius norm is additive, just like weights. The following analysis for the full lemma is also similar, but requires more care in dealing with matrices. First, for any x we have

$$\|Ax\|^2 = \|Bx\|^2 + \sum_{j=1}^{m} \|B_j x\|^2.$$

This follows since $\|Ax\|^2 = \sum_{i=1}^{n} \langle a_i, x\rangle^2$, so if nothing is sent to the coordinator, the sum can be decomposed like this with B empty. We just need to show the sum is preserved when a message $r = \sigma_1 v_1$ is sent. Because of the orthogonal decomposition of B_j by the SVD$(B_j) = [U, \Sigma, V]$, then $\|B_j x\|^2 = \sum_{\ell=1}^{d} \langle \sigma_\ell v_\ell, x\rangle^2$. Thus if we send any $\sigma_\ell v_\ell$ to the coordinator, append it to B, and remove it from B_j, the sum is also preserved. Thus, since the norm on B is always less than on A, the right side of (7.1) is proven. To see the left side of (7.1) we need to use (7.2), and show that not too much mass remains on the sites. First we bound $\|B_j x\|^2$.

$$\|B_j x\|^2 = \sum_{\ell=1}^{d} \sigma_\ell^2 \langle v_\ell, x\rangle^2 \leq \sum_{\ell=1}^{d} \frac{\varepsilon}{m} \hat{F} \langle v_\ell, x\rangle^2 = \frac{\varepsilon}{m}\hat{F} \leq \frac{\varepsilon}{m}\|A\|_F^2.$$

And thus $\sum_{j=1}^{m} \|B_j x\|^2 \leq m\frac{\varepsilon}{m}\|A\|_F^2 = \varepsilon\|A\|_F^2$ and hence

$$\|Ax\|^2 \leq \|Bx\|^2 + \sum_{j=1}^{m} \|B_j x\|^2 \leq \|Bx\|^2 + \varepsilon\|A\|_F^2.$$

□

The communication bound follows directly from the analysis of the weighted heavy hitters since the protocols for sending messages and starting new rounds are identical with $\|A\|_F^2$ in place of W, and with the squared norm change along the largest direction (the top right singular value) replacing the weight change for a single element. Thus the total communication is $O(\frac{m}{\varepsilon}\log(\beta N))$.

Theorem 7.2.1. *For a distributed matrix A whose squared norm of rows is bounded by β and for any $0 \leq \varepsilon \leq 1$, the above protocol (P2) continuously maintains $\hat{A}$ such that $0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \varepsilon\|A\|_F^2$ and incurs a total communication cost of $O((m/\varepsilon)\log(\beta N))$ messages.*

7.2.2.1 Bounding Space at Sites

It is possible to also run a small space streaming algorithm on each site j, and also maintain the same guarantees. The Frequent Directions algorithm [91] presented a stream of rows a_i forming a matrix A, maintains a matrix $\tilde{A}$ using $O(1/\varepsilon')$ rows such that $0 \leq \|Ax\|^2 - \|\tilde{A}x\|^2 \leq \varepsilon' \|A\|_F^2$ for any unit vector x.

In our setting we run this on two matrices on each site with $\varepsilon' = \varepsilon/4m$. (It can actually just be run on B_j, but then the proof is much less self-contained.) It is run on A_j, the full matrix. Then instead of maintaining B_j that is A_j after subtracting all rows sent to the coordinator, we maintain a second matrix S_j that contains all rows sent to the coordinator; it appends them one by one, just as in a stream. Now $\|B_jx\|^2 = \|A_jx\|^2 - \|S_jx\|^2$. Thus if we replace both A_j with $\tilde{A}_j$ and S_j with $\tilde{S}_j$, then we have

$$\|B_jx\|^2 = \|A_jx\|^2 - \|S_jx\|^2 \leq \|\tilde{A}_jx\|^2 - \|\tilde{S}_jx\|^2 + \frac{\varepsilon}{4m}\|A_j\|_F^2,$$

and similarly $\|B_jx\|^2 \geq \|\tilde{A}_jx\|^2 - \|\tilde{S}_jx\|^2 - \frac{\varepsilon}{4m}\|A_j\|_F^2$ (since $\|S_j\|_F^2 \leq \|A_j\|_F^2$). From here we will abuse notation and write $\|\tilde{B}_jx\|^2$ to represent $\|\tilde{A}_jx\|^2 - \|\tilde{S}_jx\|^2$.

Now we send the top singular vectors v_ℓ of $\tilde{B}_j$ to the coordinator only if $\|\tilde{B}_jv_\ell\|^2 \geq \frac{3\varepsilon}{4m}\hat{F}$. Using our derivation, thus we only send a message if $\|B_jv_\ell\|^2 \geq \frac{\varepsilon}{2m}\|A\|_F^2$, so it only sends at most twice as many as the original algorithm. Also if $\|B_jv_\ell\|^2 > \frac{\varepsilon}{m}\|A\|_F^2$ we always send a message, so we do not violate the requirements of the error bound.

The space requirement per site is then $O(1/\varepsilon') = O(m/\varepsilon)$ rows. This also means, as with Frequent Directions [91], we can run Algorithm 21 in batch mode, and only call the SVD operation once every $O(1/\varepsilon')$ rows.

It is straightforward to see the coordinator can also use Frequent Directions to maintain an approximate sketch, and only keep $O(1/\varepsilon)$ rows.

7.2.3 Distributed Matrix Tracking Protocol 3

Our next approach is very similar to that discussed in Section 2.5.5. On each site we run Algorithm 5. The only difference is that for an incoming row a_i, it treats it as an element $(a_i, w_i = \|a_i\|^2)$. The coordinator's communication pattern is also the same as Algorithm 6. The only difference is how it interprets the data it receives.

As such, the communication bound follows directly from Section 2.5.5; we need $O((m + (1/\varepsilon^2)\log(1/\varepsilon))\log(\beta N\varepsilon))$ messages, and we obtain a set S of at least $s =$

$\Theta((1/\varepsilon)^2 \log(1/\varepsilon))$ rows chosen proportional to their squared norms; however if the squared norm is large enough, then it is in the set S deterministically. To simplify notation we will say that there are exactly s rows in S.

7.2.3.1 Estimation by Coordinator

The coordinator "stacks" the set of rows $\{a_1, \ldots, a_s\}$ to create an estimate $B = [a_1; \ldots; a_s]$. We will show that for any unit vector x that $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon \|A\|_F^2$.

If we had instead used the weighted sampling with replacement protocol from Section 2.5.5.3, and retrieved $s = O(1/\varepsilon^2)$ rows of A onto the coordinator (sampled proportionally to $\|a_i\|^2$ and then rescaled to have the same weight), we could immediately show the desired bound was achieved using know results on column sampling [50]. However, as is the case with weighted heavy-hitters, we can achieve the same error bound for the "without replacement sampling" in our protocol, and this uses less communication and running time.

Recall for rows a_i such that $\|a_i\|^2 \geq \hat{\rho}$, (for a priority $\hat{\rho} < 2\tau$) it keeps them as is; for other rows, it rescales them so their squared norm is $\hat{\rho}$. And $\hat{\rho}$ is defined so that $\mathrm{E}[\|B\|_F^2] = \|A\|_F^2$, thus $\hat{\rho} \leq W/s$.

Theorem 7.2.2. *Protocol 3 (P3) uses $O((m+s)\log(\beta N/s))$ messages of communication, with $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$, and for any unit vector x we have $|\|Ax\|^2 - \|Bx\|^2| \leq \varepsilon\|A\|_F^2$, with probability at least $1 - 1/s$.*

Proof. The error bound roughly follows that of Lemma 2.5.6. We apply the same negatively correlated Chernoff-Hoeffding bound but instead define random variable $X_{i,x} = \langle a_i, x\rangle^2$. Thus $M_x = \sum_{i=1}^{s} X_{i,x} = \|Bx\|^2$. Again $\Delta = \hat{\rho}$ (since elements with $\|a_i\|^2 > \hat{\rho}$ are not random) and $\mathrm{E}[M_x] = \|Ax\|^2$. It again follows that

$$\mathbf{Pr}[|\|Bx\|^2 - \|Ax\|^2| \leq \varepsilon\|A\|_F^2/2] \leq \exp(-\varepsilon^2 s/32) \leq \delta.$$

Setting $\delta = \Omega(1/s)$ yields that when $s = \Theta((1/\varepsilon^2)\log(1/\varepsilon))$ this holds with probability at least $1 - \delta = 1 - 1/s = 1 - 1/\Theta((1/\varepsilon)^2\log(1/\varepsilon))$, for any unit vector x. □

We need $O(1)$ space per site and $O(s)$ space on coordinator.

7.2.4 Distributed Matrix Tracking Protocol 4

Again treating each row a_i as having weight $w_i = \|a_i\|^2$, then to mimic the weighted heavy-hitters protocol 4 we want to select each row with probability $\hat{p} = 1 - e^{-p\|a_i\|^2}$. Here $p = 2\sqrt{m}/(\varepsilon\hat{F})$ represents the probability to send a weight 1 item and $\hat{F}$ is a 2-approximation of $\|A\|_F^2$ (i.e. $\hat{F} \leq \|A\|_F^2 \leq 2\hat{F}$) and is maintained and provided by the coordinator. We then only want to send a message from the coordinator if that row is selected, and then it follows from the analysis in Section 2.5.5 that in total $O((\sqrt{m}/\varepsilon)\log(\beta N))$ messages are sent, since $O(\sqrt{m}/\varepsilon)$ messages are sent each round in between $\hat{F}$ doubling and being distributed by the coordinator, and there are $O(\log(\beta N))$ such rounds.

But replicating the approximation guarantees of protocol 4 is hard. In Algorithm 5, on each message a particular element e has its count updated *exactly* with respect to a site j. Because of this, we only need to bound the expected weight of stream elements until another exact update is seen (at $1/p$) and then to compensate for this we increase this weight by $1/p$ so it has the right expected value. It also follows that the variances are bounded by $1/p^2$, and thus when p is set $\Theta(\sqrt{m}/(\varepsilon W))$ we get at most εW error.

Thus the most critical part is to update the representation (of local matrices from m sites) on the coordinator so it is exact for some query. We show that this can only be done for a limited set of queries (along certain singular vectors), provide an algorithm to do so, and then show that this is not sufficient for any approximation guarantees.

7.2.4.1 Replicated Algorithm for Matrices

Each site can keep track of A_j the exact matrix describing all of its data, and an approximate matrix $\hat{A}_j$. The matrix $\hat{A}_j$ will also be kept on the coordinator for each site. So the coordinator's full approximation $\hat{A} = [\hat{A}_1; \hat{A}_2; \ldots; \hat{A}_m]$ is just the stacking of the approximation from each site. Since the coordinator can keep track of the contribution from each site separately, the sites can maintain $\hat{A}_j$ under the same process as the coordinator.

In more detail, both the site and the coordinator can maintain the $[U, \Sigma, V] = \text{SVD}(\hat{A}_j)$, where $V = [v_1, v_2, \ldots, v_d]$ stores the right singular vectors and $\Sigma = \text{diag}(s_1, s_2, \ldots, s_d)$ are the singular values. (Recall, U is just an orthogonal rotation, and does not change the squared norm.) Thus $\|\hat{A}_j x\|^2 = \sum_{i=1}^{d} s_i^2 \langle v_i, x\rangle^2$. Now if we can consider setting $A' = [\hat{A}_j; r]$ where $\|r\| = \langle v_{i'}, r\rangle$, so it is along the direction of a singular vector $v_{i'}$, then

$$\|A'x\|^2 = \|r\|^2 \langle v_{i'}, x\rangle^2 + \sum_{i=1}^{d} s_i^2 \langle v_i, x\rangle^2.$$

Thus if we update the singular value $s_{i'}$ to $\bar{s}_{i'} = \sqrt{s_{i'}^2 + \|r\|^2}$, (and to simplify notation $\bar{s}_i = s_i$ for $i \neq i'$) then $\|A'x\|^2 = \sum_{i=1}^{d}(s_i')^2\langle v_i, x\rangle^2$. Hence, we can update the squared norm of $\hat{A}_j$ in a particular direction, as long as that direction is one of its right singular values. But unfortunately, in general, for arbitrary direction x (if not along a right singular vector), we cannot do this update while also preserving or controlling the orthogonal components.

We can now explain how to use this form of update in a full protocol on both the site and the coordinator. The algorithm for the site is outlined in Algorithm 23. On an incoming row a, we updated $A_j = [A_j; a]$ and send a message with probability $1 - e^{-p\|a\|^2}$ where $p = 2\sqrt{m}/(\varepsilon\hat{F})$. If we are sending a message, we first set $z_i = \sqrt{\|A_j v_i\|^2 + 1/p}$ for all $i \in [d]$, and send a vector $z = (z_1, z_2, \ldots, z_d)$ to the coordinator. We next produce the new $\hat{A}_j$ on both site and coordinator as follows. Set $Z = \text{diag}(z_1, z_2, \ldots, z_d)$ and update $\hat{A}_j = ZV^T$. Now along any right singular vector v_i of $\hat{A}_j$ we have $\|\hat{A}_j v_i\|^2 = \|A_j v_i\|^2 + 1/p$. Importantly note that the right singular vectors of $\hat{A}_j$ do not change; although their singular values and hence ordering may change, the basis does not.

Algorithm 23 P4: Site j process new row a

```
Given $\hat{F}$ from coordinator, set $p = 2\sqrt{m}/(\varepsilon\hat{F})$.
The site also has maintained $[U, \Sigma, V] = \text{SVD}(\hat{A}_j)$.
Update $A_j = [A_j; a]$.
Set $\hat{p} = 1 - e^{-p\|a\|^2}$. Generate $u \in \text{Unif}[0,1]$.
if ($u \leq \hat{p}$) then
  for $i \in [d]$ do $z_i = \sqrt{\|A_j v_i\|^2 + 1/p}$.
  Send vector $z = (z_1, \ldots, z_d)$.
  Set $Z = \text{diag}(z_1, \ldots, z_d)$; update $\hat{A}_j = ZV^T$.
```

7.2.4.2 Error Analysis

To understand the error analysis, we first consider a similar protocol, except where instead of each z_i, we set $\bar{z}_i = \|A_j v_i\|$ (without the $1/p$). Let $\bar{Z} = \text{diag}(\bar{z}_1, \ldots, \bar{z}_d)$ and $\bar{A}_j = \bar{Z}V^T$. Now for all right singular vectors $\|A_j v_i\|^2 = \|\bar{A}_j v_i\|^2$ (this is not true for general x in place of v_i), and since $\bar{z}_i \geq s_i$ for all $i \in [d]$, then for all x we have $\|\bar{A}_j x\|^2 \geq \|\hat{A}_j^{\text{old}} x\|^2$, where $\hat{A}_j^{\text{old}}$ is the approximation before the update. See Figure 7.1 to illustrate these properties.

For directions x that are right singular values of $\hat{A}_j$, this analysis should work, since $\|\bar{A}_j x\|^2 = \|A_j x\|^2$. But two problems exist in other directions. First in any other direction x the norms $\|A_j x\|$ and $\|\bar{A}_j x\|$ are incomparable; in some directions each is larger than the other. Second, there is no utility to change the right singular vectors of $\hat{A}_j$ to align with those of A_j. The skew between the two can be arbitrarily large, again see Figure 7.1, and without being able to adjust these, this error cannot be bounded.

One option would be to after every $\sqrt{m}$ rounds send a Frequent Directions sketch B_j of A_j of size $O(1/\varepsilon)$ rows from each site to the coordinator. Then we use this B_j as the new $\hat{A}_j$. This has two problems. First it only has $O(1/\varepsilon)$ singular vectors that are well-defined, so if there is increased squared norm in its null space, it is not well-defined how to update it. And second, still in between these updates within a round, there is no way to maintain the error.

One can also try to shorten a round to update when $\hat{F}$ increases by a $(1+\varepsilon)$ factor, to bound the change within a round. But this causes $O((1/\varepsilon)\log(\beta N))$ rounds, as in Section 7.2.2, and leads to $O((\sqrt{m}/\varepsilon^2)\log(\beta N))$ total messages, which is as bad as the very conservative and deterministic algorithm P1.

Thus, for direction x that is a right singular value as analyzed above, we can get a Protocol 4 with $O((\sqrt{m}/\varepsilon)\log(\beta N))$ communication. But in the general case, how to, or if it is possible at all to, get $O((\sqrt{m}/\varepsilon)\log(\beta N))$ communication, as Protocol 4 does for weighted heavy hitters, in arbitrary distributed matrix tracking is an intriguing open problem.

7.2.4.3 Experiments With P4

In order to give a taste on why P4 does not work, we compared it with other protocols. Figures 7.2 and 7.3 show the the error this protocol incurs on PAMAP and MSD datasets. Not only does it tend to accumulate error for smaller values of ε, but for the PAMAP dataset and small ε, the returned answer is almost all error.

7.3 Experimental Evaluation

We denote our three protocols by P1, P2, and P3 in all plots. *As a baseline*, we consider two algorithms: they both send all data to the coordinator. One calls Frequent-Directions

(FD) [91], and second calls SVD which is optimal but not streaming. In all remaining experiments, we have used default value $\varepsilon = 0.1$ and $m = 50$, unless specified. Otherwise ε varied in range $\{5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}, 5 \times 10^{-1}\}$, and m varied in range $[10, 100]$.

7.3.1 Datasets

We used two large real datasets "PAMAP" and "YearPredictionMSD", from the machine learning repository of UCI.

PAMAP is a Physical Activity Monitoring dataset and contains data of 18 different physical activities (such as walking, cycling, playing soccer, etc.), performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor. The dataset contains 54 columns including a timestamp, an activity label (the ground truth) and 52 attributes of raw sensory data. In our experiments, we used a subset with $N = 629,250$ rows and $d = 44$ columns (removing columns containing missing values), giving a $N \times d$ matrix (when running to the end). This matrix is low-rank.

YearPredictionMSD is a subset from the "Million Songs Dataset" [20] and contains the prediction of the release year of songs from their audio features. It has over 500,000 rows and $d = 90$ columns. We used a subset with $N = 300,000$ rows, representing a $N \times d$ matrix (when running to the end). This matrix has high rank.

7.3.2 Metrics

We compare efficiency and accuracy of our matrix tracking protocols on the following metrics:

- err: Defined as $\|A^TA - B^TB\|_2 / \|A\|_F^2$, where A is the input matrix and B is the constructed low rank approximation to A. It is equivalent to: $\max_{\{x,\, \|x\|=1\}}(\|Ax\|^2 - \|Bx\|^2)/\|A\|_F^2$.
- msg: Number of messages (scalar-form and vector-form) sent during a protocol.

Similar to experiments in Section 2.5.7, we observed that both the approximation errors and communication costs of all methods *are very stable with respect to query time*, by executing estimations at the coordinator at randomly selected time instances. Hence, we only report the average err from queries in the very end of the stream (i.e., results of our methods on really large streams).

Table 7.1 compares all algorithms, including SVD and FD to compute rank k approximations of the matrices, with $k = 30$ and $k = 50$ on PAMAP and MSD, respectively. Since err values for the two offline algorithms are minuscule for PAMAP, it indicates it is a low rank matrix (less than 30), whereas MSD is high rank, since error remains, even with the best rank 50 approximation from the SVD method.

Note that P3WOR and P3WR refer to Protocol 3, *without replacement* and *with replacement* sampling strategies, respectively. As predicted by the theoretical analysis, we see that P3WOR outperforms P3WR in both settings, always having much less error and many fewer messages. Moreover, P3WOR will gracefully shift to sending all data deterministically with no error as ε becomes very small. Hence we only use P3WOR elsewhere, labeled as just P3.

Also note that P1 in the matrix scenario is far less effective; although it achieves very small error, it sends as many messages (or more) as the naive algorithms. Little compression is taking place by FD at distributed sites before the squared norm threshold is reached.

Figures 7.4(a) and 7.5(a) show as ε increases, error of protocols increases too. In case of P3 this observation is justified by the fact P3 samples $O((1/\varepsilon^2)\log(1/\varepsilon))$ elements, and as ε increases, it samples fewer elements, hence results in a weaker estimation of true heavy directions. In case of P2, as ε increases, they allocate a larger error slack to each site and sites communicate less with the coordinator, leading to a coarse estimation. Note that again P1 vastly outperforms its error guarantees, this time likely explained via the improved analysis of Frequent-Directions [69].

Figures 7.4(b) and 7.5(b) show number of messages of each protocol vs. error guarantee ε. As we see, in large values of ε (say for $\varepsilon > 1/m = 0.02$), P2 typically uses slightly more messages than P3. But as ε decreases, P3 surpasses P2 in number of messages. This confirms the dependency of their asymptotic bound on ε ($1/\varepsilon^2$ vs. $1/\varepsilon$). P1 generally sends much more messages than both P2 and P3.

Next, we examined the number of sites (m). Figures 7.4(c) and 7.5(c) show that P2 and P3 used more communication as m increases, showing a linear trend with respect to m. P1 shows no trend since its communication depends solely on the total weight of the stream. Note that P1 sends its whole sketch, hence fix number of messages, whenever it reaches the threshold. As expected, the number of sites does not have significant impact on the measured approximation error in any protocol; see Figures 7.4(d) and 7.5(d).

We also compared the performance of protocols by tuning the ε parameter to achieve (roughly) the same measured error. Figure 7.6 shows their communication cost (#msg) vs the err. As shown, protocols P1, P2, and P3 incur less error with more communication and each works better in various regimes of the err versus msg trade-off. P1 works the best when the smallest error is required, but more communication is permitted. Even though its communication is the same as the naive algorithms in these examples, it allows each site and the coordinator to run small space algorithms. For smaller communication requirements (several of orders of magnitude smaller than the naive methods), then either P2 or P3 is recommended. P2 is deterministic, but P3 is slightly easier to implement. Note that since MSD is high rank, and even the naive SVD or FD do not achieve really small error (e.g., 10^{-3}), it is not surprising that our algorithms do not either.

Table 7.1: Raw Numbers of PAMAP and MSD.

DataSet	**PAMAP**, $k = 30$		**MSD**, $k = 50$	
Method	err	msg	err	msg
P1	7.5859e-06	628537	0.0057	300195
P2	0.0265	10178	0.0695	6362
P3WOR	0.0057	3962	0.0189	3181
P3WR	0.0323	25555	0.0255	22964
FD	2.1207e-004	629250	0.0976	300000
SVD	1.9552e-006	629250	0.0057	300000

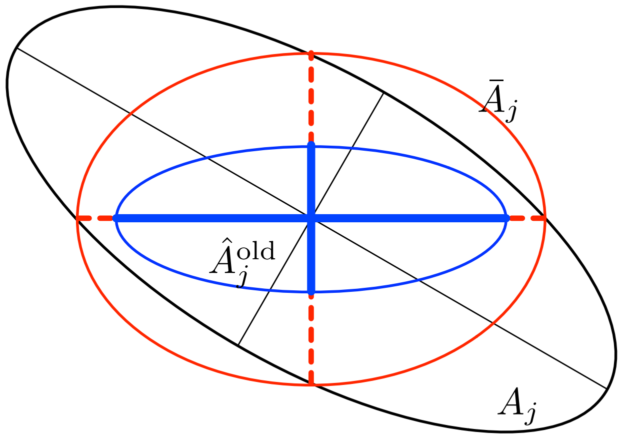


Figure 7.1: Possible update of $\hat{A}_j^{\text{old}}$ to $\bar{A}_j$ with respect to A_j for $d = 2$. Note the norm of a matrix along each direction x is an ellipse, with the axes of the ellipse corresponding to the right singular vectors. Thus $\hat{A}_j^{\text{old}}$ and $\bar{A}_j$ have the same axes, and along those axes both $\bar{A}_j$ and A_j have the same norm, but otherwise are incomparable.

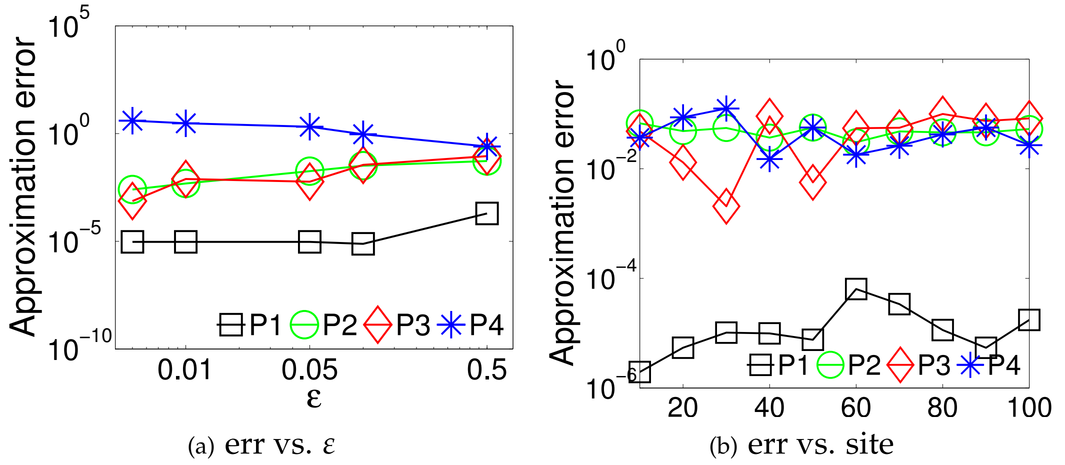


Figure 7.2: P4 vs. other protocols on PAMAP

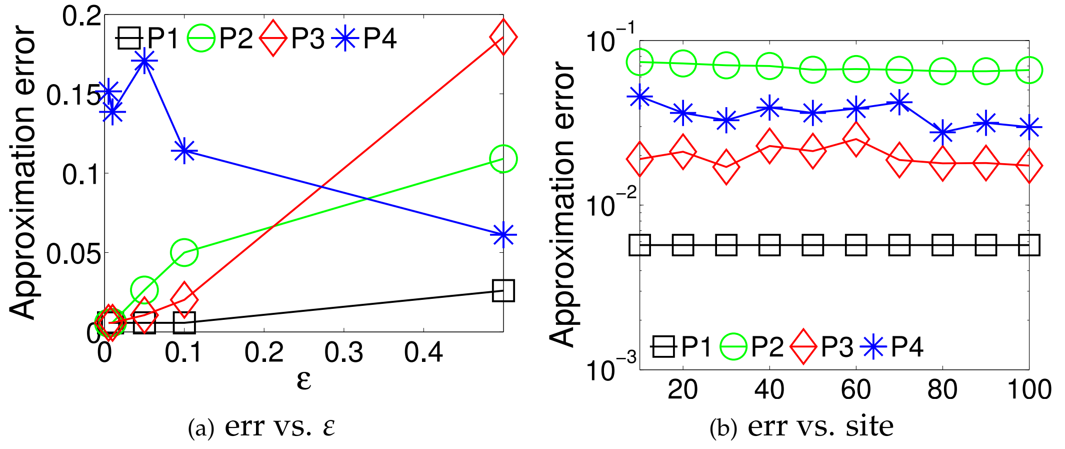
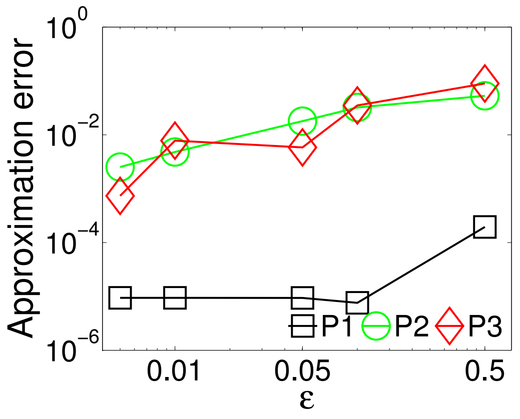
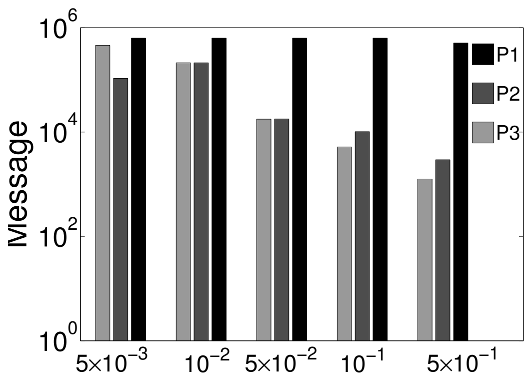


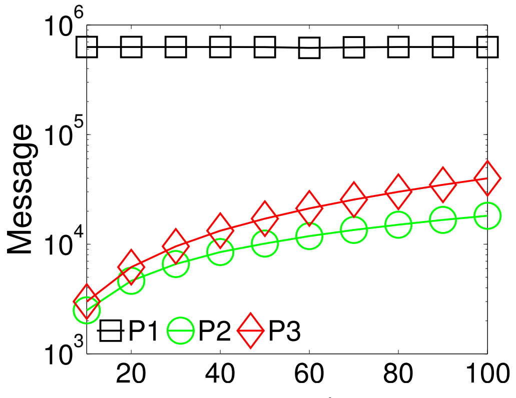
Figure 7.3: P4 vs. other protocols on MSD



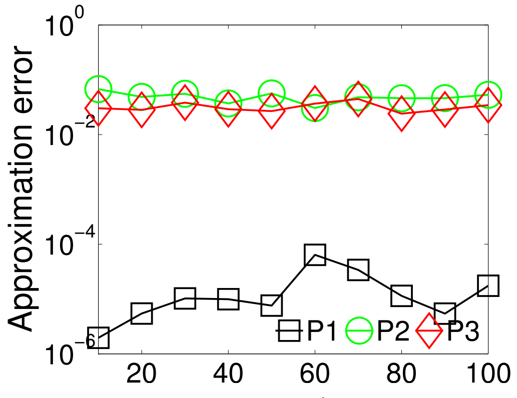
(a) err vs. ε



(b) msg vs. ε

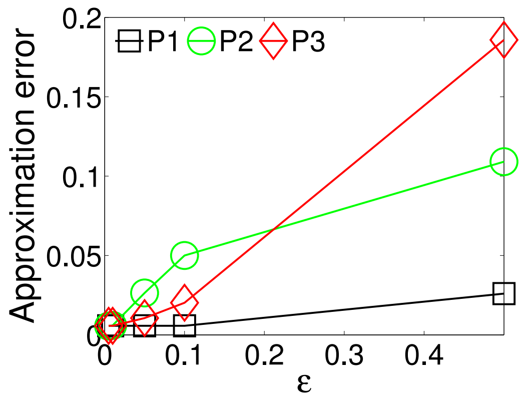


(c) msg vs. site

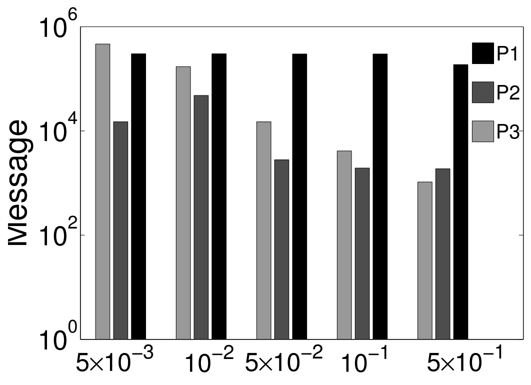


(d) err vs. site

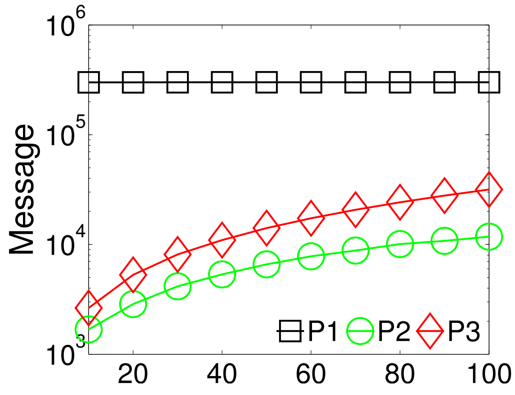
Figure 7.4: Experiments for PAMAP dataset



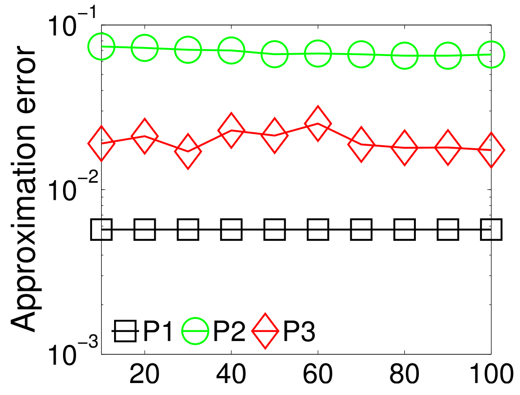
(a) err vs. ε



(b) msg vs. ε

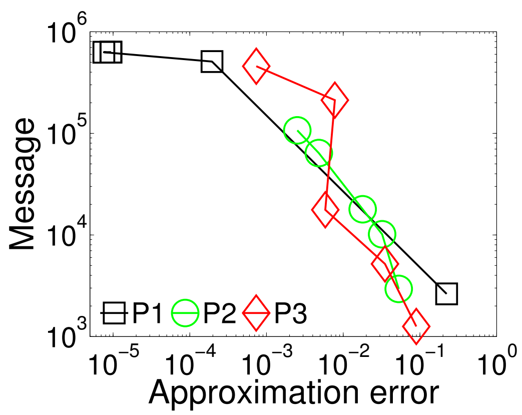


(c) msg vs. site

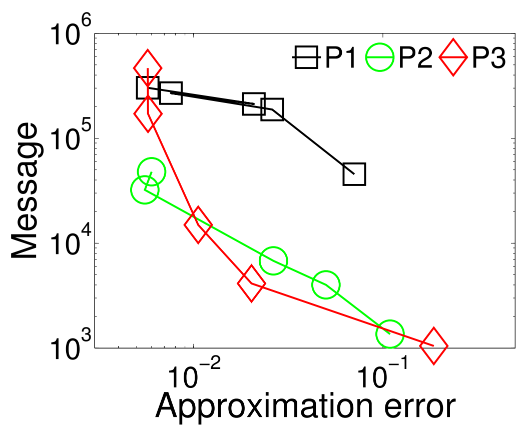


(d) err vs. site

Figure 7.5: Experiments for MSD dataset



(a) PAMAP



(b) MSD

Figure 7.6: Comparing the two protocols: msg vs. err

CHAPTER 8

CONCLUSION

Matrix sketching is becoming an important tool within large scale learning and mining. This dissertation develops FrequentDirections as a general and fundamental tool within this landscape. FrequentDirections has a lot of applications in the data reduction step of other algorithms where it serves as a concise summary. For example, in [68] we used FrequentDirections for approximating Kernel Principal Component Analysis (KPCA) of a streaming datasest. FrequentDirections has also been used for streaming anomaly detection [77] which is a fundamental problem in any data analytics system. New online hashing techniques (i.e., online sketching hashing [89]) are developed based on Frequent-Directions to approximate nearest neighbors results in search queries.

Currently, deep learning is one of the most popular areas of research. Deep learning has led to unprecedented breakthroughs for various problems in imaging and text analysis. Sketching could be used to speed up development of neural network models. Various learning algorithms need to learn hyperparameters, for tuning and performing multiple cycles of learning algorithms is too expensive. Instead one could learn on a sketched dataset for hyper parameters. Applications of FrequentDirections in this area are yet to be explored.

REFERENCES

[1] *Concept drift in machine learning and knowledge discovery group*, http://mlkd.csd.auth.gr/concept_drift.html.

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

[3] *Vision.caltech*, http://www.Vision.caltech.edu/visipedia/CUB-200-2011.html.

[4] D. Achlioptas, *Database-friendly random projections*, in Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2001, pp. 274–281.

[5] ——, *Database-friendly random projections: Johnson-Lindenstrauss with binary coins*, Journal of Computer and System Sciences, 66 (2003), pp. 671–687.

[6] D. Achlioptas, Z. S. Karnin, and E. Liberty, *Near-optimal entrywise sampling for data matrices*, in 27th Annual Conference on Neural Information Processing Systems 2013, 2013, pp. 1565–1573.

[7] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, in STOC, 2001.

[8] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, *Mergeable summaries*, in PODS, 2012.

[9] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, *Mergeable summaries*, ACM Transactions on Database Systems, 38 (2013), p. 26.

[10] R. Ahlswede and A. Winter, *Strong converse for identification via quantum channels*, IEEE Transactions on Information Theory, 48 (2002), pp. 569–579.

[11] N. Ailon and B. Chazelle, *Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform*, in Proceedings of 38th ACM Symposium on Theory of Computing, 2006.

[12] N. Ailon and E. Liberty, *An almost optimal unrestricted fast Johnson-Lindenstrauss transform*, in Proceedings of 22nd ACM-SIAM Symposium on Discrete Algorithms, 2011.

[13] A. Arasu, S. Babu, and J. Widom, *An abstract semantics and concrete language for continuous queries over streams and relations*, Stanford, 2002.

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

[15] N. Asendorf, M. McGaffin, M. Prelee, and B. Schwartz, *Algorithms for completing a user ratings matrix*.

[16] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and issues in data stream systems*, in PODS, 2002.

[17] B. Babcock and C. Olston, *Distributed top-k monitoring*, in SIGMOD, 2003.

[18] R. Berinde, G. Cormode, P. Indyk, and M. Strauss, *Space-optimal heavy hitters with strong error bounds*, ACM Trans. Database Syst., 35 (2010).

[19] R. Berinde, G. Cormode, P. Indyk, and M. J. Strauss, *Space-optimal heavy hitters with strong error bounds*, in Proceedings ACM Symposium on Principles of Database Systems, 2009.

[20] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, *The million song dataset*, in ISMIR, 2011.

[21] P. Bonnet, J. Gehrke, and P. Seshadri, *Towards sensor database systems*, Lecture Notes in Computer Science, (2001), pp. 3–14.

[22] C. Boutsidis, P. Drineas, and M. Magdon-Ismail, *Near optimal column-based matrix reconstruction*, in FOCS, 2011, pp. 305–314.

[23] C. Boutsidis, M. W. Mahoney, and P. Drineas, *An improved approximation algorithm for the column subset selection problem*, in Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2009, pp. 968–977.

[24] C. Boutsidis, M. W. Mahoney, and P. Drineas, *An improved approximation algorithm for the column subset selection problem*, in Proceedings of 20th ACM-SIAM Symposium on Discrete Algorithms, 2009.

[25] C. Boutsidis and D. P. Woodruff, *Optimal cur matrix decompositions*, in Proceedings of the 46th Annual ACM Symposium on Theory of Computing, ACM, 2014, pp. 353–362.

[26] M. Brand, *Incremental singular value decomposition of uncertain data with missing values*, in Computer Vision—ECCV 2002, Springer, 2002, pp. 707–720.

[27] S. Buss, *Connectus data set – Florida sparse matrix collection*. `http://www.cise.ufl.edu/research/sparse/matrices/Buss/connectus.html`.

[28] M. Charikar, K. Chen, and M. Farach-Colton, *Finding frequent items in data streams*, in Automata, Languages and Programming, Springer, 2002.

[29] M. Charikar, K. Chen, and M. Farach-Colton, *Finding frequent items in data streams*, in Proceedings of 29th International Colloquium on Automata, Languages and Programming, 2002.

[30] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, *Niagaracq: A scalable continuous query system for internet databases*, ACM SIGMOD Record, 29 (2000), pp. 379–390.

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

[32] K. L. Clarkson and D. P. Woodruff, *Numerical linear algebra in the streaming model*, in STOC, 2009.

[33] ——, *Low rank approximation and regression in input sparsity time*, in STOC, 2013, pp. 81–90.

[34] E. Cohen, N. Duffield, H. Kaplan, C. Lund, and M. Thorup, *Stream sampling for variance-optimal estimation of subset sums*, in Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, 2009.

[35] G. Cormode and M. Garofalakis, *Sketching streams through the net: Distributed approximate query tracking*, in VLDB, 2005.

[36] G. Cormode and S. Muthukrishnan, *An improved data stream summary: The count-min sketch and its applications*, Journal of Algorithms, 55 (2005), pp. 58–75.

[37] G. Cormode, S. Muthukrishnan, and K. Yi, *Algorithms for distributed functional monitoring*, TALG, 7 (2011), p. 21.

[38] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, *Continuous sampling from distributed streams*, JACM, 59 (2012).

[39] C. Cortes, K. Fisher, D. Pregibon, and A. Rogers, *Hancock: A language for extracting signatures from data streams*, in Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2000.

[40] A. Dasgupta, R. Kumar, and T. Sarlós, *A sparse Johnson-Lindenstrauss transform*, in Proceedings of the 42nd ACM Symposium on Theory of Computing, ACM, 2010, pp. 341–350.

[41] E. D. Demaine, A. López-Ortiz, and J. I. Munro, *Frequency estimation of internet packet streams with limited space*, in ESA, 2002.

[42] A. Desai, M. Ghashami, and J. M. Phillips, *Improved practical matrix sketching with guarantees*, arXiv preprint arXiv:1501.06561, (2015).

[43] ——, *Improved practical matrix sketching with guarantees*, IEEE Transactions on Knowledge and Data Engineering, 28 (2016), pp. 1678–1690.

[44] A. Deshpande and L. Radamacher, *Efficient volume sampling for row/column subset selection*, in Proceedings of 51st IEEE Symposium on Foundations of Computer Science, 2010.

[45] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, *Matrix approximation and projective clustering via volume sampling*, in SODA, 2006.

[46] A. Deshpande and S. Vempala, *Adaptive sampling and fast low-rank matrix approximation*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 2006, pp. 292–303.

[47] I. S. Dhillon and D. S. Modha, *Concept decompositions for large sparse text data using clustering*, Machine Learning, 42 (2001), pp. 143–175.

[48] P. Drineas and R. Kannan, *Pass efficient algorithms for approximating large matrices*, in SODA, 2003.

[49] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices i: Approximating matrix multiplication*, SIAM Journal on Computing, 36 (2006), pp. 132–157.

[50] ——, *Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix*, SICOMP, 36 (2006), pp. 158–183.

[51] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition*, SIAM Journal on Computing, 36 (2006), pp. 184–206.

[52] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, *Fast approximation of statistical leverage*, Journal of Machine Learning Research, 13 (2012), pp. 3475–3506.

[53] P. Drineas and M. W. Mahoney, *Effective resistances, statistical leverage, and applications to linear equation solving*, in arXiv:1005.3097, 2010.

[54] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Relative-error cur matrix decompositions*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 844–881.

[55] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numerische Mathematik, 117 (2011), pp. 219–249.

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

[57] N. Duffield, C. Lund, and M. Thorup, *Priority sampling for estimation of arbitrary subset sums*, JACM, 54 (2007).

[58] N. Duffield, C. Lund, and M. Thorup, *Priority sampling for estimation of arbitrary subset sums*, Journal of the ACM, 54 (2007), p. 32.

[59] J. C. C. et al., *Spanner: Google's globally distributed database*, ACM TOCS, 31 (2013).

[60] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, in FOCS, 1998.

[61] ——, *Fast Monte-Carlo algorithms for finding low-rank approximations*, JACM, 51 (2004), pp. 1025–1041.

[62] R. Fuller and M. Kantardzic, *FIDS: Monitoring frequent items over distributed data streams*, in MLDM, 2007.

[63] M. Ghashami, A. Desai, and J. M. Phillips, *Improved practical matrix sketching with guarantees*, in Proceedings of the 22nd Annual Symposium on Algorithms, 2014.

[64] M. Ghashami, F. Li, and J. M. Phillips, *Continuous matrix approximation on distributed data*, in Proceedings of the 40th International Conference on Very Large Data Bases, 2014.

[65] M. Ghashami, E. Liberty, and J. M. Phillips, *Efficient frequent directions algorithm for sparse matrices*, arXiv preprint arXiv:1602.00412, (2016).

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

[67] ——, *Frequent directions: Simple and deterministic matrix sketching*, arXiv preprint arXiv:1501.01711, (2015).

[68] M. Ghashami, D. J. Perry, and J. Phillips, *Streaming kernel principal component analysis*, in Artificial Intelligence and Statistics, 2016, pp. 1365–1374.

[69] M. Ghashami and J. M. Phillips, *Relative errors for deterministic low-rank matrix approximation*, in SODA, 2014.

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

[71] M. Ghashami and J. M. Phillips, *Relative errors for deterministic low-rank matrix approximations*, in Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms, 2014.

[72] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Quicksand: Quick summary and analysis of network data*, tech. rep., Technical Report, Dec. 2001. citeseer. nj. nec. com/gilbert01quicksand. html, 2001.

[73] L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro, *Identifying frequent items in sliding windows over on-line packet streams*, in SIGCOMM IMC, 2003.

[74] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3, JHUP, 2012.

[75] N. Halko, P. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[76] P. Hall, D. Marshall, and R. Martin, *Incremental eigenanalysis for classification*, in British Machine Vision Conference, 1998.

[77] H. Huang and S. Kasiviswanathan, *Streaming anomaly detection using online matrix sketching*.

[78] Z. Huang, K. Yi, and Q. Zhang, *Randomized algorithms for tracking distributed count, frequencies, and ranks*, in PODS, 2012.

[79] G. Hulten, L. Spencer, and P. Domingos, *Mining time-changing data streams*, in KDD, 2001.

[80] R. Y. Hung and H.-F. Ting, *Finding heavy hitters over the sliding window of a weighted data stream*, in LATIN, 2008.

[81] W. B. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, Contemporary Mathematics, 26 (1984), p. 1.

[82] D. M. Kane and J. Nelson, *Sparser Johnson-Lindenstrauss transforms*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2012, pp. 1195–1206.

[83] R. M. Karp, S. Shenker, and C. H. Papadimitriou, *A simple algorithm for finding frequent elements in streams and bags*, ACM TODS, 28 (2003), pp. 51–55.

[84] R. Keralapura, G. Cormode, and J. Ramamirtham, *Communication-efficient distributed monitoring of thresholded counts*, in SIGMOD, 2006.

[85] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, Computer Science Department, University of Toronto, Technical Report, (2009).

[86] A. Lakhina, M. Crovella, and C. Diot, *Diagnosing network-wide traffic anomalies*, in SIGCOMM, 2004.

[87] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, *Structural analysis of network traffic flows*, in SIGMETRICS, 2004.

[88] K. Lang, *Newsweeder: Learning to filter netnews*, in Proceedings of the Twelfth International Conference on Machine Learning, 1995, pp. 331–339.

[89] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, *Online sketching hashing*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2503–2511.

[90] A. Levey and M. Lindenbaum, *Sequential Karhunen-Loeve basis extraction and its application to images*, IEEE Transactions on Image Processing, 9 (2000), pp. 1371–1374.

[91] E. Liberty, *Simple and deterministic matrix sketching*, in Proceedings 19th ACM Conference on Knowledge Discovery and Data Mining, (arXiv:1206.0594 in June 2012), 2013.

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

[93] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, PNAS, 104 (2007), pp. 20167–20172.

[94] M. W. Mahoney and P. Drineas, *Cur matrix decompositions for improved data analysis*, Proceedings of the National Academy of Sciences, 106 (2009), pp. 697–702.

[95] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, *Finding (recently) frequent items in distributed data streams*, in ICDE, 2005.

[96] J. Matoušek, *On variants of the Johnson-Lindenstrauss lemma*, Random Structures & Algorithms, 33 (2008), pp. 142–156.

[97] A. Metwally, D. Agrawal, and A. E. Abbadi, *An integrated efficient solution for computing frequent and top-k elements in data streams*, ACM Transactions on Database Systems (TODS), 31 (2006), pp. 1095–1133.

[98] J. M. P. Mina Ghashami, Edo Liberty and D. P. Woodruff, *Frequent directions: Simple and deterministic matrix sketching*, arXiv preprint.

[99] J. Misra and D. Gries, *Finding repeated elements*, Science of Computer Programming, 2 (1982), pp. 143–152.

[100] C. Musco and C. Musco, *Stronger approximate singular value decomposition via the block Lanczos and power methods*, arXiv preprint arXiv:1504.05477, (2015).

[101] S. Muthukrishnan, *Data streams: Algorithms and applications*, Now Publishers Inc, 2005.

[102] J. Nelson and H. L. Nguyen, *OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings*, in Proceedings of 54th IEEE Symposium on Foundations of Computer Science, 2013.

[103] R. I. Oliveira, *Sums of random hermitian matrices and an inequality by rudelson*, Electron. Commun. Probab., 15 (2010), pp. 203–212.

[104] A. Panconesi and A. Srinivasan, *Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds*, SICOMP, 26 (1997), pp. 350–368.

[105] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, *Latent semantic indexing: A probabilistic analysis*, in PODS, 1998.

[106] S. Papadimitriou, J. Sun, and C. Faloutsos, *Streaming pattern discovery in multiple time-series*, in VLDB, 2005.

[107] K. Papagiannaki, N. Taft, and A. Lakhina, *A distributed approach to measure ip traffic matrices*, in SIGCOMM IMC, 2004.

[108] D. Papapailiopoulos, A. Kyrillidis, and C. Boutsidis, *Provable deterministic leverage score sampling*, in Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2014.

[109] Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, and Z. Zhang, *MadLINQ: Large-scale distributed matrix computation for the cloud*, in EuroSys, 2012.

[110] V. Rokhlin, A. Szlam, and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 1100–1124.

[111] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, *Incremental learning for robust visual tracking*, International Journal of Computer Vision, 77 (2008), pp. 125–141.

[112] M. Rudelson and R. Vershynin, *Sampling from large matrices: An approach through geometric functional analysis*, JACM, 54 (2007), p. 21.

[113] T. Sarlos, *Improved approximation algorithms for large matrices via random projections*, in FOCS, 2006.

[114] M. Sullivan and A. Heybey, *A system for managing large databases of network traffic*, in USENIX, 1998.

[115] M. Szegedy, *The DLT priority sampling is essentially optimal*, STOC, (2006).

[116] S. Tirthapura and D. P. Woodruff, *Optimal random sampling in distributed streams revisited*, in PODC, 2011.

[117] S. S. Vempala, *The random projection method*, vol. 65, American Mathematical Soc., 2004.

[118] S. Venkatasubramanian and Q. Wang, *The Johnson-Lindenstrauss transform: An empirical study*, in Proceedings of ALENEX Workshop on Algorithms Engineering and Experimentation, 2011.

[119] R. Vershynin, *A note on sums of independent random matrices after Ahlswede-Winter*, Lecture Notes, (2009).

[120] J. S. Vitter, *Random sampling with a reservoir*, ACM Transactions on Mathematical Software, 11 (1985), pp. 37–57.

[121] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, *Feature hashing for large scale multitask learning*, in Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 1113–1120.

[122] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, *Feature hashing for large scale multitask learning*, in Proceedings of 26th Interantional Conference on Machine Learning, 2009.

[123] R. Witten and E. Candès, *Randomized algorithms for low-rank matrix factorizations: Sharp performance bounds*, Algorithmica, 72 (2013), pp. 264–281.

[124] D. P. Woodruff, *Low rank approximation lower bounds in row-update streams*, in Proceedings of the 27th Annual Conference on Advances in Neural Information Processing Systems, 2014.

[125] ——, *Sketching as a tool for numerical linear algebra*, Foundations and Trends in Theoretical Computer Science, 10 (2014), pp. 1–157.

[126] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Applied and Computational Harmonic Analysis, 25 (2008), pp. 335–366.

[127] A. C.-C. Yao, *Some complexity questions related to distributive computing*, in STOC, 1979.

[128] K. Yi and Q. Zhang, *Optimal tracking of distributed heavy hitters and quantiles*, Algorithmica, 65 (2013), pp. 206–223.

[129] Y. Zhu and D. Shasha, *Statstream: Statistical monitoring of thousands of data streams in real time*, in VLDB, 2002.