Simple and Deterministic Matrix Sketching

Edo Liberty
Yahoo! Labs
Haifa, Israel
edo.liberty@ymail.com

ABSTRACT

A sketch of a matrix A is another matrix B which is significantly smaller than A, but still approximates it well. Finding such sketches efficiently is an important building block in modern algorithms for approximating, for example, the PCA of massive matrices. This task is made more challenging in the streaming model, where each row of the input matrix can be processed only once and storage is severely limited.

In this paper, we adapt a well known streaming algorithm for approximating item frequencies to the matrix sketching setting. The algorithm receives n rows of a large matrix $A \in \mathbb{R}^{n \times m}$ one after the other, in a streaming fashion. It maintains a sketch $B \in \mathbb{R}^{\ell \times m}$ containing only $\ell \ll n$ rows but still guarantees that $A^T A \approx B^T B$. More accurately,

$$\forall x, \|x\| = 1 \qquad 0 \leq \|Ax\|^2 - \|Bx\|^2 \leq 2\|A\|_f^2/\ell$$

Or

$$B^T B \prec A^T A \text{ and } \|A^T A - B^T B\| \leq 2\|A\|_f^2/\ell \ .$$

This algorithm's error decays proportionally to $1/\ell$ using $O(m\ell)$ space. In comparison, random-projection, hashing or sampling based algorithms produce convergence bounds proportional to $1/\sqrt{\ell}$. Sketch updates per row in A require amortized $O(m\ell)$ operations and the algorithm is perfectly parallelizable. Our experiments corroborate the algorithm's scalability and improved convergence rate. The presented algorithm also stands out in that it is deterministic, simple to implement, and elementary to prove.

Categories and Subject Descriptors

G.1.2 [**Numerical Analysis**]: Approximation

Keywords

Streaming, matrix sketching, frequent items

1. INTRODUCTION

Modern large data sets are often viewed as large matrices. For example, textual data in the bag-of-words model is represented by a matrix whose rows correspond to documents. In large scale image analysis, each row in the matrix corresponds to one image and contains either pixel values or other derived feature values. Other large scale machine learning systems generate such matrices by converting each example into a list of numeric features. Low rank approximations for such matrices are used in common data mining tasks such as Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), and k-means clustering. Regardless of the data source, the optimal low rank approximation for any matrix is obtained by its truncated Singular Value Decompositions (SVD).

Data matrices, as above, are often extremely large and distributed across many machines. This renders standard SVD algorithms infeasible. Given a very large matrix A, a common approach is to compute a sketch matrix B that is significantly smaller than the original. A good sketch matrix B is such that computations can be performed on B rather than on A without much loss in precision.

Matrix sketching methods are, therefore, designed to be pass-efficient, i.e., the data is read at most a constant number of times. If only one pass is required, the computational model is also referred to as the streaming model. The streaming model is especially attractive since a sketch can be obtained while the data is collected. In that case, storing the original matrix becomes superfluous, see [17] for more motivations and examples of streaming algorithms for data mining applications.

There are three main matrix sketching approaches, presented here in an arbitrary order. The first generates a sparser version of the matrix. Sparser matrices are stored more efficiently and can be multiplied faster by other matrices [4][2][15]. The second approach is to randomly combine matrix rows [25][28][27][21]. The proofs for these rely on subspace embedding techniques and strong concentration of measure phenomena. The above methods will be collectively referred to as *random-projection* in the experimental section. A recent result along these lines [8], gives simple and efficient subspace embeddings that can be applied in time $O(nnz(A))$ for any matrix A. We will refer to this result as *hashing* in the experimental section. While our algorithm requires more computation than *hashing*, it will produce more accurate sketches given a fixed sketch size. The third sketching approach is to find a small subset of matrix rows (or columns) that approximate the entire matrix. This problem is known as the Column Subset Selection Problem and has

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

been thoroughly investigated, [16][12][6][11][14][5]. Recent results offer algorithms with almost matching lower bounds, [11][5][7]. Alas, it is not immediately clear how to compare some of these methods' results to ours since their objectives are different. They aim to recover a low rank matrix whose column space contains most of the space spanned by the matrix top k singular vectors. Moreover, most of the above algorithms are quite intricate and require several passes over the input matrix.

A simple streaming solution to the Column Subset Selection problem is obtained by sampling rows from the input matrix. The rows are sampled with probability proportional to their squared ℓ_2 norm. Despite this algorithm's apparent simplicity, providing tight bounds for its performance required over a decade of research [16][3][12][26][29][24][14]. We will refer to this algorithm as *sampling*. Algorithms such as CUR utilize the leverage scores of the rows [13] and not their squared ℓ_2 norms. The discussion on matrix leverage scores goes beyond the scope of this paper, see [22] for more information and references.

This manuscript proposes a fourth approach. It draws on the similarity between the matrix sketching problem and the item frequency estimation problem. In the following, we shortly describe the item frequency approximation problem, as well as a well known algorithm for it.

1.1 Frequent-items

In the item frequency approximation problem there is a universe of m items $a_1, \ldots, a_m$ and a stream $A_1, \ldots, A_n$ of item appearances. The frequency f_i of item a_i stands for the number of times a_i appears in the stream. It is trivial to produce all item frequencies using $O(m)$ space simply by keeping a counter for each item. Our goal is to use $O(\ell)$ space and produce approximate frequencies g_j such that $|f_j - g_j| \leq n/\ell$ for all j simultaneously.

This problem received an incredibly simple and elegant solution in [23]. It was later independently rediscovered by [10] and [19], who also improved its update complexity. The algorithm simulates a process of 'deleting' from the stream ℓ appearances of *different* items. This is performed repeatedly for as long as possible, namely, until there are less than ℓ unique items left. This trimmed stream is stored concisely in $O(\ell)$ space. The claim is that, if item a_j appears in the final trimmed stream g_j times, then g_j is a good approximation for its true frequency f_j (even if $g_j = 0$). This is because $f_j - g_j \leq t$, where t is the number of times items were deleted. Each item type is deleted at most once in each deletion batch. Moreover, we delete ℓ items in every batch and at most n items can be deleted altogether. Thus, $t\ell \leq n$ or $t \leq n/\ell$, which completes the proof. The reader is referred to [19] for an efficient streaming implementation. From this point on, we refer to this algorithm as *Frequent-items*.

The following is a description of the item frequency problem as a matrix sketching problem. Let A be a matrix that is given to the algorithm as a stream of its rows. For now, let us constrain the rows of A to be indicator vectors. In other words, we have $A_i \in \{e_1, \ldots, e_m\}$, where e_j is the j'th standard basis vector. Note that such a matrix can encode a stream of items (as above). If the i'th element in the stream is a_j, then the i'th row of the matrix is set to $A_i = e_j$. The frequency f_j can be expressed as $f_j = \|Ae_j\|^2$. Moreover, a good sketch B would be one such that $g_j = \|Be_j\|^2$ is a good approximation to f_j. Replacing $n = \|A\|_f^2$, we get that the condition $|f_j - g_j| \leq n/\ell$ is equivalent to $|\|Ae_j\|^2 - \|Be_j\|^2| \leq \|A\|_f^2/\ell$. From the above, it is clear that for 'item indicator' matrices, a sketch $B \in \mathbb{R}^{\ell \times m}$ can be obtained by the *Frequent-items* algorithm.

1.2 Frequent-directions

In this paper we describe *Frequent-directions*, an extension of *Frequent-items* to general matrices. Given any matrix $A \in \mathbb{R}^{n \times m}$, the algorithm processes the rows of A one by one and produces a sketch matrix $B \in \mathbb{R}^{\ell \times m}$, such that

$$B^T B \prec A^T A \text{ and } \|A^T A - B^T B\| \leq 2\|A\|_f^2/\ell \ .$$

The intuition behind *Frequent-directions* is surprisingly similar to that of *Frequent-items*: In the same way that *Frequent-items* periodically deletes ℓ different elements, *Frequent-directions* 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". This intuition is made exact below. As a remark, when presented with an item indicator matrix, *Frequent-directions* exactly mimics a variant of *Frequent-items*.

As its name suggests, the *Frequent-items* algorithm is often used to uncover frequent items in an item stream. Namely, if one sets $\ell > 1/\varepsilon$, then any item that appears more than εn times in the stream must appear in the final sketch. Similarly, *Frequent-directions* can be used to uncover any unit vector (direction) in space x for which $\|Ax\|^2 \geq \varepsilon\|A\|_2^2$ by taking $\ell > 2r/\varepsilon$.[1]

This property makes *Frequent-directions* very useful in practice. In data-mining, it is common to represent data matrices by low rank matrices. Typically, one computes the SVD of A and approximates it using the first k singular vectors and values. The value k is such that the k'th singular value is larger than some threshold value t. In other words, we only "care about" unit vectors such that $\|Ax\| \geq t$. Using *Frequent-directions* we can invert this process. We can prescribe t in advance and find the space of all vectors x such that $\|Ax\| \geq t$ directly while circumventing the SVD computation altogether.

1.3 Connection to sampling

Here we point out another similarity between item frequency estimation and matrix sketching. It is simple to see that all item frequencies can be approximated from a uniform sample of the stream. Using Chernoff's bound (and then applying the union bound carefully) one concludes that $O(r\log(r)/\varepsilon^2)$ samples suffice to ensure that $|f_i - g_i| \leq \varepsilon f_{max}$. In this context we define $r = n/f_{\max}$. Similarly, matrix sketching by row sampling [24][14] requires $O(r\log(r)/\varepsilon^2)$ row samples where $r = \|A\|_f^2/\|A\|_2^2$ to ensure that $\|A^T A - B^T B\| \leq \varepsilon\|A^T A\|$. From the above discussion, it is evident that the matrix sampling result implies the item sampling algorithm. This is because running the matrix sampling algorithm on an item indicator matrix (as before) produces uniform random samples. Moreover, for such matri-

[1]Here $r = \|A\|_f^2/\|A\|_2^2$ denotes the numeric rank of A. For any matrix $A \in \mathbb{R}^{n \times m}$ the numeric rank $r = \|A\|_f^2/\|A\|_2^2$ is a smooth relaxation of the algebraic rank $Rank(A)$.

ces, $r = \|A\|_f^2/\|A\|_2^2 = f_{max}/n$ and $f_{max} = \|A^T A\|$. We argue that *Frequent-directions* improves on matrix sampling in the same way that *Frequent-items* improves on item sampling.

1.4 Connection to low rank approximation

Low rank approximation of matrices is a well studied problem. The goal is to obtain a small matrix B, containing ℓ rows, that contains in its row space a projection matrix $\Pi^A_{B,\xi}$ of rank k such that $\|A - A\Pi^A_{B,\xi}\|_\xi \leq (1+\varepsilon)\|A - A_k\|_\xi$. Here, A_k is the best rank k approximation of A and ξ is either 2 (spectral norm) or f (Frobenius norm). It is difficult to compare our algorithm to this line of work since the types of bounds obtained are qualitatively different. We note, however, that it is possible to use *Frequent-directions* to produce a low rank approximation result.

Lemma 4 from a version of [12] (modified). Let P_k^B denote the projection matrix on the right k singular vectors of B corresponding to its largest singular values. Then the following holds: $\|A - AP_k^B\|^2 \leq \sigma_{k+1}^2 + 2\|A^T A - B^T B\|$, where σ_{k+1} is the $(k+1)$'th singular value of A.

Let $r = \|A\|_f^2/\|A\|_2^2$ denote the numeric rank of A and let $\ell \geq 4r\sigma_1^2/\varepsilon\sigma_{k+1}^2$. Using *Frequent-directions* and letting the sketch B maintain ℓ columns, we get that $2\|A^T A - B^T B\| \leq \varepsilon\sigma_{k+1}^2$ and therefore $\|A - AP_k^B\| \leq \sigma_{k+1}(1+\varepsilon)$, which is a $1+\varepsilon$ approximation to the optimal solution. Since $r\sigma_1^2/\sigma_{k+1}^2 \in \Omega(k)$, this is asymptotically inferior to the space requirement of [5]. That said, if $r\sigma_1^2/\sigma_{k+1}^2 \in O(k)$, *Frequent-directions* is also optimal due to [7].

2. FREQUENT-DIRECTIONS

The algorithm keeps an $\ell \times m$ sketch matrix B that is updated every time a new row from the input matrix A is added. Rows from A simply replace all-zero valued rows of the sketch B. The algorithm maintains the invariant that all-zero valued rows always exist. Otherwise, half the rows in the sketch are nullified by a two-stage process. First, the sketch is rotated (from the left) using its SVD such that its rows are orthogonal and in descending magnitude order. Then, the sketch rows norms are "shrunk" so that at least half of them are set to zero. In the algorithm, we denote by $[U, \Sigma, V] \leftarrow \text{SVD}(B)$ the Singular Value Decomposition of B. We use the convention that $U\Sigma V^T = B$, $U^T U = V^T V = VV^T = I_\ell$, where I_ℓ stands for the $\ell \times \ell$ identity matrix. Moreover, Σ is a non-negative diagonal matrix such $\Sigma = \text{diag}([\sigma_1, \ldots, \sigma_\ell])$, $\sigma_1 \geq \ldots \geq \sigma_\ell \geq 0$. We also assume that $\ell/2$ is an integer.

Algorithm 1 *Frequent-directions*

Input: ℓ, $A \in \mathbb{R}^{n\times m}$
$B \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell\times m}$
for $i \in [n]$ **do**
  Insert A_i into a zero valued row of B
  if B has no zero valued rows **then**
    $[U, \Sigma, V] \leftarrow \text{SVD}(B)$
    $C \leftarrow \Sigma V^T$ # Only needed for proof notation
    $\delta \leftarrow \sigma_{\ell/2}^2$
    $\check{\Sigma} \leftarrow \sqrt{\max(\Sigma^2 - I_\ell\delta, 0)}$
    $B \leftarrow \check{\Sigma}V^T$ # At least half the rows of B are all zero
  end if
end for
Return: B

Claim 1. *If B is the result of applying Algorithm 1 to matrix A, then:*

$$0 \preceq B^T B \preceq A^T A$$

Proof. First, $0 \preceq B^T B$ because $B^T B$ is positive semidefinite for any matrix B. Second, $B^T B \preceq A^T A$ is a consequence of the fact that $\forall x \ \|Ax\|^2 - \|Bx\|^2 \geq 0$. Let B^i and C^i denote the values of B and C after the main loop in the algorithm has been executed i times. For example, B^0 is an all zeros matrix and $B^n = B$ is the returned sketch.

$$\begin{aligned}\|Ax\|^2 - \|Bx\|^2 &= \sum_{i=1}^{n}[\langle A_i, x\rangle^2 + \|B^{i-1}x\|^2 - \|B^i x\|^2] \\ &= \sum_{i=1}^{n}[\|C^i x\|^2 - \|B^i x\|^2] \geq 0\end{aligned}$$

The statement that $\langle A_i, x\rangle^2 + \|B^{i-1}x\|^2 = \|C^i x\|^2$ holds true because A_i is inserted in a zero valued row of B^{i-1}. Also, note that C^i is an isometric left rotation of a matrix containing the rows of B^{i-1} and the new row A_i. Finally, $\|C^i x\|^2 - \|B^i x\|^2 \geq 0$ because ${C^i}^T C^i \succeq {B^i}^T B^i$ by the definition of the shrinking step. □

Claim 2. *If B is the result of applying Algorithm 1 to matrix A with prescribed sketch size ℓ, then:*

$$\|A^T A - B^T B\| \leq 2\|A\|_f^2/\ell$$

Proof. Let δ_i denote the value of δ at time step i. If the algorithm does not enter the 'if' section in the i'th step, then $\delta_i = 0$. Similarly, let B^i, C^i, and V^i be the values of B, C and V after the main loop in the algorithm is executed i times.

We start by bounding the value of $\|A^T A - B^T B\|$ as a function of δ_i. In what follows, x is the eigenvector of $A^T A - B^T B$ corresponding to its largest eigenvalue.

$$\begin{aligned}\|A^T A - B^T B\| &= \|Ax\|^2 - \|Bx\|^2 \\ &= \sum_{i=1}^{n}[\langle A_i, x\rangle^2 + \|B^{i-1}x\|^2 - \|B^i x\|^2] \\ &= \sum_{i=1}^{n}[\|C^i x\|^2 - \|B^i x\|^2] \\ &\leq \sum_{i=1}^{n}\|{C^i}^T C^i - {B^i}^T B^i\| \\ &= \sum_{i=1}^{n}\|(\Sigma^i)^2 - (\check{\Sigma}^i)^2\| = \sum_{i=1}^{n}\delta_i\end{aligned}$$

The second transition is correct because $\langle A_i, x\rangle^2 + \|B^{i-1}x\|^2 = \|C^i x\|^2$ which is explained in the proof of Claim 1. The last step is obtained by replacing C^i and B^i by their definitions.

We now turn to bounding the value of $\sum_{i=1}^{n}\delta_i$ by computing the Frobenius norm of the resulting sketch B:

$$\begin{aligned}
\|B^n\|_f^2 &= \sum_{i=1}^{n}[\|B^i\|_f^2 - \|B^{i-1}\|_f^2] \\
&= \sum_{i=1}^{n}[(\|C^i\|_f^2 - \|B^{i-1}\|_f^2) - (\|C^i\|_f^2 - \|B^i\|_f^2)] \\
&= \sum_{i=1}^{n}\|A_i\|^2 - \mathrm{tr}(C^{i^T}C^i - B^{i^T}B^i) \\
&= \|A\|_f^2 - \sum_{i=1}^{n}\mathrm{tr}((\Sigma^i)^2 - (\check{\Sigma}^i)^2) \\
&\leq \|A\|_f^2 - (\ell/2)\cdot\sum_{i=1}^{n}\delta_i \ .
\end{aligned}$$

The reason that $\|C^i\|_f^2 - \|B^{i-1}\|_f^2 = \|A_i\|^2$ is, again, because C^i is, up to a unitary left rotation, a matrix that contains both B^{i-1} and A_i. The last transition is correct because $\mathrm{tr}((\Sigma^i)^2 - (\check{\Sigma}^i)^2) \geq (\ell/2)\delta_i$, which in turn is true because the matrix $((\Sigma^i)^2 - (\check{\Sigma}^i)^2)$ contains ℓ non-negative elements on its diagonal at least half of which are equal to δ_i. We conclude that $\sum_{i=1}^{n}\delta_i \leq 2(\|A\|_f^2 - \|B\|_f^2)/\ell$. Combining this with our earlier observation that $\|A^TA - B^TB\| \leq \sum_{i=1}^{n}\delta_i$, we obtain that $\|A^TA - B^TB\| \leq 2(\|A\|_f^2 - \|B\|_f^2)/\ell$. This fact will be used in Section 2.2. By the simple fact that $\|B\|_f^2 \geq 0$, we obtain $\|A^TA - B^TB\| \leq 2\|A\|_f^2/\ell$. This completes the proof of the claim. □

2.1 Running time

Let $T_{\mathrm{SVD}}(\ell, m)$ stand for the number of operations required to obtain the Singular Value Decomposition of an ℓ by m matrix. The worst case update time of *Frequent-directions* is therefore $O(T_{\mathrm{SVD}}(\ell, m))$, which is also $O(m\ell^2)$. That said, the SVD of the sketch is computed only once every $\ell/2$ iterations. This is because the shrinking step in the algorithm nullifies at least $\ell/2$ rows in B. When the SVD is not computed, the addition running time is $O(m)$. The total running time is therefore bounded by $O(nm\ell)$. This gives an amortized update time of $O(m\ell)$ per row in A.

2.2 Parallelization and sketching sketches

A convenient property of this sketching technique is that it allows for combining sketches. Let $A = [A_1; A_2]$ such that A consists of the rows of A_1 and A_2 stacked on top of one another. Also, let B_1 and B_2 be the sketches computed by the above technique for A_1 and A_2 respectively. Now let the final sketch, C, be the sketch of a matrix $B = [B_1; B_2]$ that contains the two sketches B_1 and B_2 vertically stacked. We show below that $\|A^TA - C^TC\| \leq 2\|A\|_f^2/\ell$. This means that sketching each half of A separately and then sketching the resulting sketches is as good as sketching A directly. To see this, we compute $\|Cx\|^2$ for a test vector $\|x\| = 1$:

$$\begin{aligned}
\|Cx\|^2 &\geq \|Bx\|^2 - (2/\ell)(\|B\|_f^2 - \|C\|_f^2) \\
&= \|B_1x\|^2 + \|B_2x\|^2 \\
&\quad -(2/\ell)(\|B_1\|_f^2 + \|B_2\|_f^2) + (2/\ell)\|C\|_f^2 \\
&\geq \|A_1x\|^2 - (2/\ell)(\|A_1\|_f^2 - \|B_1\|_f^2) \\
&\quad +\|A_2x\|^2 - (2/\ell)(\|A_2\|_f^2 - \|B_2\|_f^2) \\
&\quad -(2/\ell)(\|B_1\|_f^2 + \|B_2\|_f^2) + (2/\ell)\|C\|_f^2 \\
&= \|A_1x\|^2 + \|A_2x\|^2 \\
&\quad -(2/\ell)(\|A_1\|_f^2 + \|A_2\|_f^2) + (2/\ell)\|C\|_f^2 \\
&= \|Ax\|^2 - (2/\ell)(\|A\|_f^2 - \|C\|_f^2) \ .
\end{aligned}$$

Here we use the fact that $\|B_1x\|^2 \geq \|A_1x\|^2 - \varepsilon(\|A_1\|_f^2 - \|B_1\|_f^2)$ for $\|x\| = 1$, which is shown in the proof of Claim 2. 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 *Frequent-directions*.

3. EXPERIMENTS

We compare *Frequent-directions* to five different algorithms. The first two constitute brute force and naïve baselines. The other three are common algorithms that are used in practice: *sampling*, *hashing*, and *random-projection*. References can be found in the introduction. All tested methods receive the rows of an $n \times m$ matrix A one by one. They are all limited in storage to an $\ell \times m$ sketch matrix B and additional $o(\ell m)$ space for any auxiliary variables. This is with the exception of the brute force algorithm that requires $\Theta(m^2)$ space. For a given input matrix A we compare the computational efficiency of the different methods and their resulting sketch accuracy. The computational efficiency is taken as the time required to produce B from the stream of A's rows. The accuracy of a sketch matrix B is measured by $\|A^TA - B^TB\|$. Since some of the algorithms below are randomized, each algorithm was executed 5 times for each input parameter setting. The reported results are median values of these independent executions.

The experiments were conducted on a FreeBSD machine with 48GB of RAM and a 12MB cache using a single Intel(R) Xeon(R) X5650 CPU. All experimental results, from which the plots below are obtained, are available for download as *json* formatted records at [20].

3.1 Competing algorithms

Brute Force: The brute force approach produces the optimal rank ℓ approximation of A. It explicitly computes the matrix $A^TA = \sum_i^n A_i^TA_i$ by aggregating the outer products of the rows of A. The final 'sketch' consists of the top ℓ right singular vectors and values (square rooted) of A^TA which are obtained by computing its SVD. The update time of Brute Force is $\Theta(m^2)$ and its space requirement is $\Theta(m^2)$.
Naïve: Upon receiving a row in A the naïve method does nothing. The sketch it returns is an all zeros ℓ by m matrix. This baseline is important for two reasons: First, it can actually be more accurate than random methods due to under sampling scaling issues. Second, although it does not per-

form any computation, it does incur computation overheads such as I/O exactly like the other methods.

Sampling: Each row in the sketch matrix B^{samp} is chosen i.i.d. from A and rescaled. More accurately, each row B_j^{samp} takes the value $A_i/\sqrt{\ell p_i}$ with probability $p_i = \|A_i\|^2/\|A\|_f^2$. The space it requires is $O(m\ell)$ 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 $O(m)$ per row in A. For a theoretical analysis of this algorithm the reader is referred to [12][26][24].

Hashing: The matrix B^{hash} is generated by adding or subtracting the rows of A from random rows of B^{hash}. More accurately, B^{hash} is initialized to be an $\ell \times m$ all zeros matrix. Then, when processing A_i we perform $B^{hash}_{h(i)} \leftarrow B^{hash}_{h(i)} + s(i)A_i$. Here $h : [n] \to [\ell]$ and $s : [n] \to \{-1, 1\}$ 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" or "hashing trick" [31]. For a surprising new analysis of this method see [8].

Random-projection: The matrix B^{proj} is equivalent to the matrix RA where R is an $\ell \times n$ matrix such that $R_{i,j} \in \{-1/\sqrt{\ell}, 1/\sqrt{\ell}\}$ uniformly. Since R is a random projection matrix [1], B^{proj} contains the m columns of A randomly projected from dimension n to dimension ℓ. This is easily computed in a streaming fashion, while requiring at most $O(m\ell)$ space and $O(m\ell)$ operation per row updated. For proofs of correctness and usage see [25][28][27][21]. Sparser constructions of random projection matrices are known to exist [9][18]. These, however, were not implemented since the running time of applying random projection matrices is not the focus of this experiment.

3.2 Synthetic data

Each row of the generated input matrices, A, consists of a d dimensional signal and m dimensional noise ($d \ll m$). More accurately, $A = SDU + N/\zeta$. The signal coefficients matrix $S \in \mathbb{R}^{n \times d}$ is such that $S_{i,j} \sim \mathcal{N}(0,1)$ i.i.d. The diagonal matrix D is $D_{i,i} = 1 - (i-1)/d$, which gives linearly diminishing signal singular values. The signal row space matrix $U \in \mathbb{R}^{d \times m}$ contains a random d dimensional subspace in $\mathbb{R}^m$, for clarity, $UU^T = I_d$. The matrix SDU is exactly rank d and constitutes the signal we wish to recover. The matrix $N \in \mathbb{R}^{n \times m}$ contributes additive Gaussian noise $N_{i,j} \sim \mathcal{N}(0,1)$. Due to [30], the spectral norms of SDU and N are expected to be the same up to some universal constant c_1. Experimentally, $c_1 \approx 1$. Therefore, when $\zeta \le c_1$ we cannot expect to recover the signal because the noise spectrally dominates it. On the other hand, when $\zeta \ge c_1$ the spectral norm is dominated by the signal which is therefore recoverable. Note that the Frobenius norm of A is dominated by the noise for any $\zeta \le c_2\sqrt{m/d}$, for another constant close to 1, c_2. Therefore, in the typical case where $c_1 \le \zeta \le c_2\sqrt{m/d}$, the signal is recoverable by spectral methods even though the vast majority of the energy in each row is due to noise.

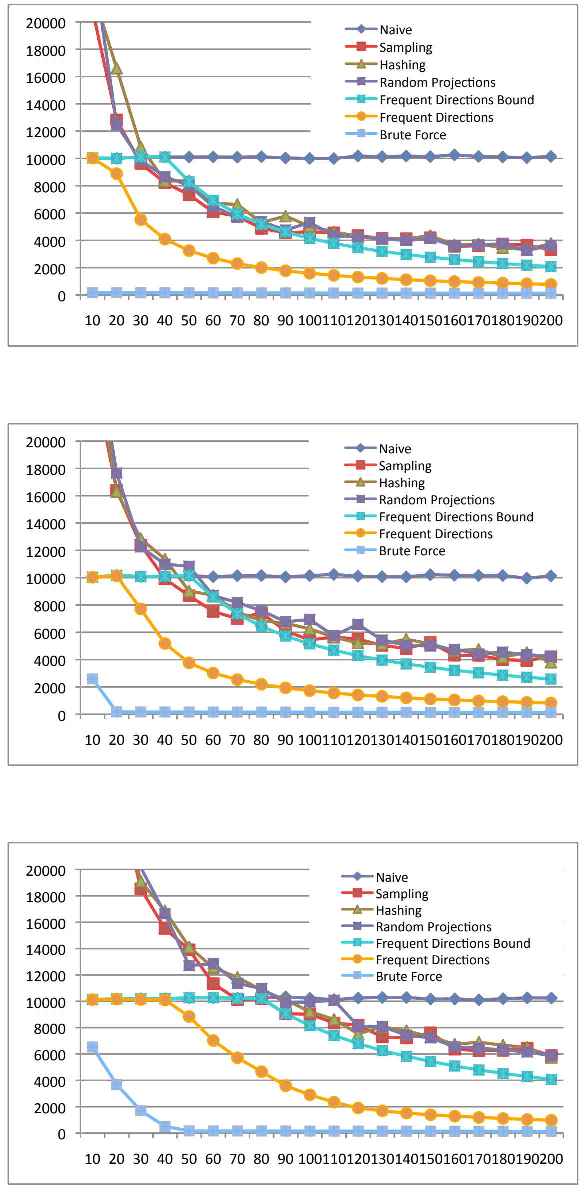

Figure 1: Accuracy vs. sketch size. The y-axis indicates the accuracy of the sketches. If a method returns a sketch matrix B, the accuracy is measured by $\|A^TA - B^TB\|$. The size of the sketch is fixed for all methods and is $B \in \mathbb{R}^{\ell \times m}$. The value of ℓ is indicated on the x-axis. The form of the input matrix is explained in Section 3.2. Here the signal dimensions are $d = 10, 20, 50$ ordered from top to bottom. The signal to noise ratio is kept constant at $\zeta = 10$. Each plot line corresponds to one of the sketching techniques explained in Section 3.1. The only plot line that does not correspond to an algorithm is denoted by *Frequent-directions* bound. This is the theoretical worst case performance guaranteed by *Frequent-directions*.

3.3 Results

The performance of *Frequent-directions* was measured both in terms of accuracy and running time compared to the above algorithms. In the first experiment, a moderately sized matrix $(10,000 \times 1,000)$ was approximated by each algorithm. The moderate input matrix size is needed to accommodate the brute force algorithm and to enable the exact error measure. The results are shown in Figure 1 and give rise to a few interesting observations. First, all three random techniques actually perform worse than naïve for small sketch sizes. This is a side effect of under-sampling which causes overcorrection. This is not the case with *Frequent-directions*. Second, the three random techniques perform equally well. This might be a result of the chosen input. Nevertheless, practitioners should consider these as potentially comparable alternatives. Third, the curve indicated by "Frequent Direction Bound" plots the accuracy *guaranteed* by *Frequent-directions*. Note that "Frequent Direction Bound" is consistently lower than the random methods. This means that the worst case performance guarantee is lower than the actual performance of the competing algorithms. Finally, *Frequent-directions* produces significantly more accurate sketches than predicted by its worst case analysis.

The running time of *Frequent-directions*, however, is not better than its competitors. This is clearly predicted by their asymptotic running times. In Figure 2, the running times (in seconds) of the sketching algorithms are plotted as a function of their sketch sizes. Clearly, the larger the sketch, the higher the running time. Note that *hashing* is extremely fast. In fact, it is almost as fast as *naïve*, which does nothing! *Sampling* is also faster than *Frequent-directions* but only by a factor of roughly 2. This is surprising because it should be faster by a factor of ℓ. *Frequent-directions*, however, executes faster than *random-projection* although they share the same asymptotic running time $(O(nd\ell))$. It is important to stress that the implementations above are not very well optimized. Different implementations might lead to different results.

Nevertheless, we will claim that *Frequent-directions* scales well. Its running time is $O(nm\ell)$, which is linear in each of the three terms. In Figure 3, we fix the sketch size to be $\ell = 100$ and increase n and m. Note that the running time is indeed linear in both n and m as predicted. Moreover, sketching an input matrix of size $10^5 \times 10^4$ requires roughly 3 minutes. Assuming 4 byte floating point numbers, this matrix occupies roughly $4Gb$ of disk space. More importantly though, *Frequent-directions* is a streaming algorithm. Thus, its memory footprint is fixed and its running time is exactly linear in n. For example, sketching a $40Gb$ matrix of size $10^6 \times 10^4$ terminates in half an hour. The fact that *Frequent-directions* is also perfectly parallelizable (Section 2.2) makes *Frequent-directions* applicable to truly massive and distributed matrices.

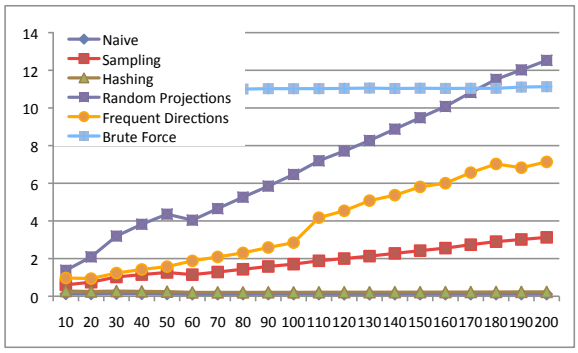

Figure 2: Running time in seconds vs. sketch size. Each method produces a sketch matrix B of size $\ell \times m$ for a dense $n \times m$ matrix. Here, $n = 10,000$, $m = 1,000$ and the value of ℓ is indicated on the x-axis. The total amount of computation time required to produce the sketch is indicated on the y-axis in seconds. The brute force method computes the complete SVD of A, and therefore its running time is independent of ℓ. Note that *hashing* is almost as fast as the naïve method and independent of ℓ which is expected. The rest of the methods exhibit a linear dependence on ℓ which is also expected. Surprisingly though, *Frequent-directions* is more computationally efficient than *random-projection* although both asymptotically require $O(nm\ell)$ operations.

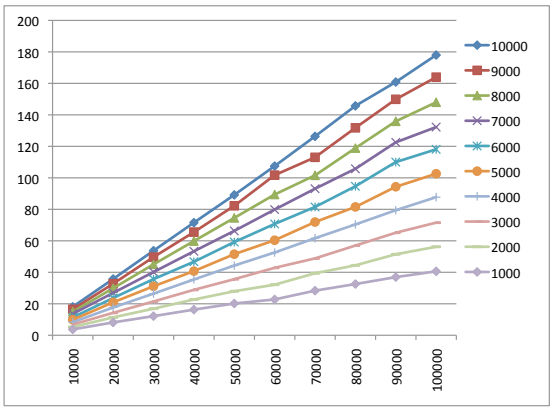

Figure 3: Running time in seconds vs. input matrix size. Here, we measure only the running time of *Frequent-directions*. The sketch size is kept fixed at $\ell = 100$. The size of the input matrix is $n \times m$. The value of n is indicated on the x-axis. Different plot lines correspond to different values of m (indicated in the legend box). The running time is measured in seconds and is indicated on the y-axis. It is clear from this plot that the running time of *Frequent-directions* is linear in both n and m. Note also that sketching a $10^5 \times 10^4$ dense matrix terminates in roughly 3 minutes.

4. FUTURE WORK

Note that *Frequent-directions* does not take advantage of any possible sparsity of the input matrix. Designing a better version of this algorithm that utilizes the input matrix sparsity should be possible. One possible direction is to replace the SVD step with a light-weight orhogonalization step. Another improvement might enable a *Frequent-directions*-like algorithm that can process the entries of the matrix in an arbitrary order and not only row by row. This is impor-

tant for recommendation systems. In this setting, user actions correspond to single non-zeros in the matrix and are presented to the algorithm one by one in an arbitrary order. New concentration results show that sampling entries (correctly) yields good matrix sketches, but no deterministic streaming algorithm is known.

Acknowledgments: The author truly thanks Petros Drineas, Jelani Nelson, Nir Ailon, Zohar Karnin, Yoel Shkolnisky and Amit Singer for very helpful discussions and pointers.

5. REFERENCES

[1] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 274–281, New York, NY, USA, 2001. ACM.

[2] Dimitris Achlioptas and Frank Mcsherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), 2007.

[3] Rudolf Ahlswede and Andreas Winter. Strong converse for identification via quantum channels. *IEEE Transactions on Information Theory*, 48(3):569–579, 2002.

[4] Sanjeev Arora, Elad Hazan, and Satyen Kale. A fast random sampling algorithm for sparsifying matrices. In *Proceedings of the 9th international conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th international conference on Randomization and Computation*, APPROX'06/RANDOM'06, pages 272–279, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near optimal column-based matrix reconstruction. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 305–314, Washington, DC, USA, 2011. IEEE Computer Society.

[6] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 968–977, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[7] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 205–214, New York, NY, USA, 2009. ACM.

[8] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 81–90, New York, NY, USA, 2013. ACM.

[9] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse johnson: Lindenstrauss transform. In *STOC*, pages 341–350, 2010.

[10] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 348–360, London, UK, UK, 2002. Springer-Verlag.

[11] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *APPROX-RANDOM*, pages 292–303, 2006.

[12] Petros Drineas and Ravi Kannan. Pass efficient algorithms for approximating large matrices. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 223–232, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[13] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error cur matrix decompositions. *SIAM J. Matrix Analysis Applications*, 30(2):844–881, 2008.

[14] Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamas Sarlos. Faster least squares approximation. *Numer. Math.*, 117(2):219–249, February 2011.

[15] Petros Drineas and Anastasios Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Inf. Process. Lett.*, 111(8):385–389, March 2011.

[16] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 370–, Washington, DC, USA, 1998. IEEE Computer Society.

[17] Phillip B. Gibbons and Yossi Matias. External memory algorithms, 1999.

[18] Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. In *SODA*, pages 1195–1206, 2012.

[19] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, March 2003.

[20] Edo Liberty. www.cs.yale.edu/homes/el327/public/experimentalresults/.

[21] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences,*, 104(51):20167–20172, December 2007.

[22] Michael W. Mahoney, Petros Drineas, Malik Magdon-Ismail, and David P. Woodruff. Fast approximation of matrix coherence and statistical leverage. In *ICML*, 2012.

[23] Jayadev Misra and David Gries. Finding repeated elements. Technical report, Ithaca, NY, USA, 1982.

[24] Roberto Imbuzeiro Oliveira. Sums of random hermitian matrices and an inequality by rudelson. arXiv:1004.3821v1, April 2010.

[25] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: a probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 159–168, New York, NY, USA, 1998. ACM.

[26] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM*, 54(4), July 2007.

[27] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *FOCS*, pages 143–152, 2006.

[28] S. S. Vempala. *The Random Projection Method*. American Mathematical Society, 2004.

[29] Roman Vershynin. A note on sums of independent random matrices after ahlswede-winter. *Lecture Notes*.

[30] Roman Vershynin. Spectral norm of products of random and deterministic matrices. *Probability Theory and Related Fields*, 150(3-4):471–509, 2011.

[31] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM.