TEL AVIV UNIVERSITY

THE RAYMOND AND BEVERLY SACKLER FACULTY OF
EXACT SCIENCES
SCHOOL OF MATHEMATICAL SCIENCES
THE DEPARTMENT OF APPLIED MATHEMATICS

FAST AND ROBUST ALGORITHMS FOR LARGE SCALE STREAMING PCA

A thesis submitted toward a degree of
Master of Science in Applied Mathematics

by

Tal Halpern

May 2017

TEL AVIV UNIVERSITY

THE RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
SCHOOL OF MATHEMATICAL SCIENCES
THE DEPARTMENT OF APPLIED MATHEMATICS

FAST AND ROBUST ALGORITHMS FOR LARGE SCALE STREAMING PCA

A thesis submitted toward a degree of
Master of Science in Applied Mathematics

by

Tal Halpern

This research was carried out in The
School of Mathematical Sciences

This work was carried out under the supervision of
Prof. Sivan Toledo and Prof. Yoel Shkolnisky

May 2017

Abstract

We present a range of new incremental (single-pass streaming) algorithms for principal components analysis (IPCA). IPCA algorithms process the columns of a matrix A one at a time and attempt to build a basis for a low-dimensional subspace that spans the dominant subspace of A. We present a unified framework for IPCA algorithms, show that many existing ones are parameterizations of it, propose new sophisticated algorithms and show that both the new algorithms and many existing ones can be implemented more efficiently than was previously known. There are several ways to measure the quality of an approximate PCA. We focus on the reconstruction of the dominant left subspace of the matrix. We discuss existing ways to measure the quality of PCA algorithms, as well as a new metric that we believe is better suited for high dimensional settings. Unfortunately, we show that existing IPCA algorithms do not satisfy some useful error bounds and that many existing algorithms can fail even in easy cases. We show experimentally that our new algorithms outperform existing ones.

One of the algorithms developed under this thesis was published at SAC '17: Proceedings of the Symposium on Applied Computing. The article is titled "Enhanced situation space mining for data streams" [17] and is the outcome of a collaboration with a group from the Ben Gurion University.

A second paper, that is a summery of this thesis, was recently accepted to PPAM 2017. The article is titled: "Advances in Incremental PCA Algorithms" and is authored by Tal Halpern and Sivan Toledo.

Table of Contents

[1] This equivalence was first suggested by Baker [2]

[2] While our framework is for a 1-by-1, fixed rank updates, it can be extended to block wise and dynamically changing rank

Acknowledgments

I would like to thank Sivan Toledo for his guidance, assistance and support. This research could not have happened without him. Sivan asks difficult questions and offers invaluable suggestions. This work is the product of the two.

List of Figures

List of Tables

1 Introduction

We explore algorithms that approximate the dominant left singular vectors and the associated singular values of a m-by-n matrix A that is close to a rank-$\bar{k}$ for some $\bar{k} \ll m \ll n$. That is, we seek a m-by-k matrix U with orthogonal columns and an k-by-k diagonal matrix S such that $USV^T \approx A$ for some V, where $k \geq \bar{k}$ but also $k \approx \bar{k}$. That is, the degree of the approximation can be a little higher than k but not much higher. We assume that a bound k on $\bar{k}$ is known. These approximate singular vectors and values are commonly known as a *principle components analysis* or PCA. If V is also computed, the problem is essentially finding an approximate singular-value decomposition (SVD). In many applications, PCA suffices; this is helpful, since V is huge.

We are interested in algorithms that construct U and S using data structures with $O(mk)$ words of memory that read every column of A only once. Algorithms that obey the first constraint are called *streaming*, to indicate that the large input data updates a small data structure; there is not enough memory to hold in memory a data structure as large as the input. The *single-pass* constraint makes the problem even more difficult. We refer to algorithms that meet these requirements as *Incremental Principle Component Analysis* (IPCA).

We start by providing a deep review of the related work. Over the years many different papers suggested many different algorithms (some very similar to each over) and we believe putting all of this work under a single scope is very insightful.

We continue by suggesting a general, high level framework, for IPCA. Using the framework we are able to cluster different algorithms together, according to what they do rather then by how they do it. The clustering enables us to perform accuracy analysis for each cluster and not per algorithm. This type of analysis is

also beneficial as it is also relevant for algorithms that will be developed in the future. We use a new error criterion which we believe is better suited for high dimensional settings then previously used ones. Using this new error criterion we show that all covered methods are heuristic, in the sense that all can fail even in very simple cases.

Next we present four new algorithms. The first will be an alternative implantation to a known algorithm called *FrequentDirections* [15] which has the best accuracy guaranties (under a different error criterion then the one used here). The new implantation, *QR-FD* will have the same guaranties but will also have the best known arithmetic complexity. The second is a hybrid of a basic IPCA algorithm and *FrequentDirections* called *TunableShrinkage* that has a control parameter, allowing per-problem tuning. The last two are new type of algorithms using preprocessing and randomization, engineered specifically to meet the failure modes other algorithms had. Both algorithms are without a tuning parameter and with an arithmetic cost bounded by the best known.

We conclude the work with tests on synthetic and real-world data sets.

1.1 Contributions

- We conduct deep literature review of past research, placing work from fields like Latent Semantic Indexing (LSI), Image analysis, Numerical Linear Algebra and Matrix Sketching under a single scope.
- We present a unified framework for IPCA. This novel framework cluster different implementations solely by what they do, not how they do it.
- We propose a new type of error criterion, especially suited for high dimensional data
- We conduct accuracy analysis for each cluster which will also apply to any future algorithm that falls into one of the clusters.
- We show that, under the new error criterion, all of the algorithms that fits the framework and covered in this thesis can fail completely and as such, all should be consider heuristic.

- We propose 4 new algorithms: an alternative implantation to an algorithm called *FrequentDirections*, yielding better complexity / memory usage then originally suggested. A new algorithm called *TunableShrinkage*, equipped with a control parameter, allowing per-problem tuning (scaling between different failure modes of known algorithms) . Two heuristic randomized algorithms called *BIPCA* and *JIT-PCA*. Both were specifically developed in order to achieve a "one fit all" method and with arithmetic complexity bounded by the best known.

2 Related Work

The literature on Incremental PCA (IPCA) is quite rich and spanned over more then two decades. The suggested algorithms are often closely related to one another but the precise relationships are often left implicit and vague. It seems like the researchers, coming from different areas, were not aware of the related work. We believe that the fact that the same algorithms were independently developed in different areas, demonstrate the importance and practical requirement for IPCA. While the general idea is fairly simple, the actual implantation of the algorithms can be very complicated. As example, the most efficient algorithm achieves a linear time update (with regards to the approximation rank) using a very sophisticated method. We will show that this specific implantation was probably not known to researchers working on similar problems, leading to much more complected ways to reach the same linear cost.

There are many different approaches for IPCA. The approximation rank can be fixed during the calculation or can change dynamically. The algorithm may process at each iteration only one column or a block of columns. While our work focuses on a fixed rank, 1-by-1 update, we provide a wider scope in this review. We divide the review into four parts. The first will review SVD based methods. Those methods maintain $U_t \in \mathbb{R}^{m \times k_t}$, an orthogonal rank-$k_t$ approximation to the left singular basis of $A_t = \begin{bmatrix} a_1 & a_2 & \cdots & a_t \end{bmatrix}$ and $S_t \in \mathbb{R}^{k_t \times k_t}$ an approximation to the singular values matrix. They also share the fact that the update process includes performing SVD on a small square matrix of size $O(k_t)$. The second part present an IPCA method based on a *QR* representation, with $Q_t \in \mathbb{R}^{m \times k_t}$ orthogonal and $R_t \in \mathbb{R}^{k_t \times k_t}$ a upper triangular matrix. We continue in the third part with unification of the SVD and QR methods. The forth and final part discusses a relatively new algorithm called *FrequentDirections* and its extensions. *Frequent-*

Directions is similar to SVD based methods with the addition of a function that shrinks the estimation of the singular values after each update.

2.1 SVD Based Methods

First we present a simple method for an exact, rank increasing SVD update to the currant approximation. Let us start with the the 1-by-1 case. at $t = 1$ we set $U_1 = a_1/\|a_1\|$, $S_1 = \|a_1\|$ and $V = 1$. At iteration $t = 2,3...$ we are looking for $\tilde{U}_{t+1}\tilde{S}_{t+1}\tilde{V}_{t+1}^T = \begin{bmatrix} U_t S_t V_t^T & a_{t+1} \end{bmatrix}$ which can be found by:

1. Set $a_{t+1}^{orth} = \left(I - U_t U_t^T\right) a_{t+1} \quad \rho = \|a_{t+1}^{orth}\| \quad p = a_{t+1}^{orth}/\rho$

2. Expend $\begin{bmatrix} U_t S_t V_t^T & a_{t+1} \end{bmatrix} = \begin{bmatrix} U_t & p \end{bmatrix} \begin{bmatrix} S_t & U_t^T a_{t+1} \\ \bar{0} & \rho \end{bmatrix} \begin{bmatrix} V_t & \bar{0} \\ \bar{0} & 1 \end{bmatrix}^T$

3. Calculate the svd of $Z = \begin{bmatrix} S_t & U_t^T a_{t+1} \\ \bar{0} & \rho \end{bmatrix} = U_Z S_Z V_Z^T$

4. Conclude the update with:

$$\tilde{U}_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} U_Z \quad \tilde{S}_{t+1} = S_Z \quad \tilde{V}_{t+1} = \begin{bmatrix} V_t & \bar{0} \\ \bar{0} & 1 \end{bmatrix} V_Z$$

In order to turn this exact, rank increasing SVD update, into a low rank IPCA we need to include two changes. By not maintaining or updating V we turn the SVD into PCA. In order to turn this exact full rank PCA into low rank IPCA we need to perform some sort of rank control. The final step of the IPCA algorithm generates U_{t+1} and S_{t+1} via a truncation decision. If we perform a fixed rank update then we simply drop the last singular pair from $\tilde{U}_{t+1}$ and $\tilde{S}_{t+1}$. If we allow rank increase then the smallest singular value is compared to a threshold value and the singular pair is dropped only if the smallest singular value is smaller then the threshold. We call a 1-by-1 update with fixed rank the *Basic IPCA*. The rotation $\tilde{U}_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} U_Z$ is the most expensive part of the *Basic IPCA* with cost of $O(mk^2)$ arithmetic operations. One way to reduce the total cost of the

algorithm is to batch updates (going from 1-by-1 update to block wise). With a batch-size parameter ℓ, each iteration now costs $\Theta(m(k+\ell)^2)$. If we set ℓ such that $\ell = \Theta(k)$, the per-iteration cost is still $\Theta(mk^2)$, but the number of iterations is ℓ times smaller, only n/ℓ. Therefore, the amortized per-iteration cost reduces to $\Theta(mk)$.

The block wise update is very similar to the 1-by-1, the difference is that we don't have only one new direction but l_{t+1} (size of a block at step $t+1$, assuming the additional part has full rank). Denote the block of new columns by $W \in \Re^{m \times l}$

1. Set $W^{\text{orth}} = (I - U_t U_t^T)W$
2. Find an orthogonal decomposition $QR = W^{orth}$
3. Expend $\begin{bmatrix} U_t S_t V_t^T & W \end{bmatrix} = \begin{bmatrix} U_t & Q \end{bmatrix} \begin{bmatrix} S_t & U^T W \\ \bar{0} & R \end{bmatrix} \begin{bmatrix} V_t & \bar{0} \\ \bar{0} & I \end{bmatrix}^T$
4. Calculate the SVD of $Z = \begin{bmatrix} S_t & U^T W \\ \bar{0} & R \end{bmatrix} = U_Z S_Z V_Z^T$
5. Conclude the update with: $\tilde{U}_{t+1} = \begin{bmatrix} U_t & Q \end{bmatrix} U_Z \quad \tilde{S}_{t+1} = S_Z$

Similar to the 1-by-1 version, the last step is truncation decision. If the approximation is using fixed rank then all of the singular pairs with index $(k+1) : (k+l)$ are dropped. If we allow rank increase then we use a truncation rule (there are many options) to decide which pairs to include and which, if any, to drop. We call a block-wise update with a fixed rank *Block Basic IPCA*.

For each new column / block of columns, the two *Basic* versions finds the k dominant singular pairs of the matrix $\begin{bmatrix} US & W \end{bmatrix}$ and they are equivalent to an even a simpler method we call the *Straightforward IPCA*:

1. Calculate the SVD of $\begin{bmatrix} US & W \end{bmatrix} = \tilde{U}\tilde{S}\tilde{V}^T$
2. Drop $\tilde{V}$ and truncate $\tilde{U}$ and $\tilde{S}$ back to rank-k

Asymptotically, the cost of the *Basic IPCA* and the cost of the 1-by-1 *Straightforward IPCA* is the same but the first has a better constant.

O'Brien [19] and Berry et el [4], both interested in Latent Semantic Indexing (LSI), suggested the *Block Basic IPCA,* without including R, (providing no explanation for it). As a result, their approximation is automatically kept at a fix rank and no truncation is needed. (for them $U_{t+1} = \tilde{U}_{t+1}$ etc). O'Brien, Berry et el also suggested other types of matrix / PCA updates which are not relevant to us.

Zha and Simon [20], also dealing with LSI applications, published a similar article to O'Brien, Berry et el. They claimed that not including R was an error. Their update is exactly the *Block Basic IPCA*. Zha and Simon proved the following theorem: let $B = \begin{bmatrix} A & D \end{bmatrix}$ and $\bar{B} = \begin{bmatrix} A_k & D \end{bmatrix}$, if $B^T B = X + \sigma I$ with $\sigma > 0$ and X symmetric positive semi definite with rank k then $B_k = \bar{B}_k$.

Chandrasekaran et el [16, 9], interested in image analysis, suggested a 1-by-1 update with dynamic rank, using a threshold value δ_i. At each step, the singular values which are bigger then the threshold are kept (along with their matching singular vectors). The matrix

$$Z = \begin{bmatrix} S_t & U_t^T a_{t+1} \\ \bar{0} & \rho \end{bmatrix}$$

is a broken arrowhead matrix. In order to reduce complexity they suggest using the method of Gu and Eisenstat [13] allowing the SVD of Z to be calculated in $O(k^2)$ instead of $O(k^3)$ and reducing the cost of the final rotation $\tilde{U}_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} U_Z$ to $O(mk)$. The technique of Gu and Eisenstat is mentioned in many papers, usually with the same comment, that the overhead is too high to be practical and that it is complicated to use. In their first paper, Chandrasekaran et al. tried to define a value for δ such that $\|a_i - U_n S_n V_n^T e_i\|_2 < \varepsilon$ for $i = 1 : n$ meaning the approximation will approximate up to ε all of the column vectors. They proved that $\delta = \varepsilon/n$ guaranties this bound. In a second paper, they tried to find a value δ such that $\|A_{1:n} - U_n S_n V_n^T\|_2 < \varepsilon$ and again using some assumptions they proved that $\delta = \varepsilon/n$ guaranties this bound. They also mentioned that this rule for δ might cause the approximation rank to be overly high so alternatively (and heuristically) they suggested using $\varepsilon/\sqrt{n}$. This thesis is aiming toward applications on which n is huge. Any bound achieved by using $\delta = \varepsilon/n$ or $\delta = \varepsilon/\sqrt{n}$ will lead to a very

high rank, possibly even a full rank. It appears that Zha, and Simon [20] were not aware of the two articles by Chandrasekaran et al. even though they were published earlier.

Levy and Lindenbaum [14], also interested in image processing, suggested a block based update that was expressed a bit differently than what we wrote but is basically the same. They added several novelties: addition of a fixed decay rate $0 < \lambda \leq 1$ for the singular values from update to update. They suggested setting a threshold value ε. The rank of the update is first reduced to a fixed k . Then they calculate $\sqrt{\varepsilon \sum_{i=1}^{k} \sigma_i^2}$ and remove all singular values which are smaller (ε determines the ratio between the singular value and the total, till-k-sum). Perhaps their greatest contribution was finding the block size which optimize the amortized per-column operation count. They proved that using $k/\sqrt{2}$ minimizes it to $12mk$

Brand suggested an exact full rank incremental SVD [6, 5]. For simplicity we will demonstrate his method for a 1-by-1 update but he also suggested a block-wise version. Brand maintains two separate orthogonal matrices representing the singular basis, U (as usual) and a small rotation matrix U_{small}. Brand starts with $U_1 = a_1/\|a_1\|$, $U_{small} = 1$ and $t = 1$. Then, for each additional column

$$U_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} \quad \text{and} \quad U_{\text{small}} = \begin{bmatrix} U_{\text{small}} & 0 \\ 0 & 1 \end{bmatrix} U_Z$$

U grows by column additions only, avoiding the costly rotations $\begin{bmatrix} U_t & p \end{bmatrix} U_Z$ (which are cached in U_{small} by only performing small matrix multiplication). To re-iterate, this is an exact full rank SVD (in his paper Brand includes V, here, for simplicity and relevance, we describe his method without it). The problem is that his trick only save cost if Z is small, otherwise the calculation of

$$\begin{bmatrix} U_{\text{small}} & 0 \\ 0 & 1 \end{bmatrix} U_Z$$

is no longer cheap. For that reason he targets low rank matrices. If the matrix rank is exactly $k \ll m$ (meaning $\sigma_i = 0$ for $i = k+1 : m$) then most of the updates will have $\rho = 0$, $U_{t+1} = U_t$ (we add no new direction) and $U_{\text{small}} = U_{\text{small}}\, U_{Z,(1:k)}$. In real world data, the rank in never exactly k and you have a long tail of small

singular values. Without some sort of rank control his method will not save cost. He offers several heuristic rank control rules. One is the regular one, if the singular value is smaller then a certain threshold, throw it away. Another method, for the block version, is to calculate $\sqrt{det(R^T R)}$ and if it's under a threshold value throw away all new directions. For the 1-by-1 update (his recommended version), he suggests deciding on a rank and when we reach that rank to set $\rho = 0$ for all future updates. We call this procedure, when dropping V, *Brand IPCA*. Brand defends this heuristic truncation by two reasons: if the rank is higher then k then anyway you need to perform some sort of truncation. Also, because the update is so cheap one can easily use a higher rank then needed (as long as it is still low). Again, it appears that Brand was not aware to the work by Chahlaoui, K. Gallivan, and P. Van Dooren [8, 7] and Baker's thesis (both are discussed shortly)[2].

2.2 A QR Based Method

The per-update cost of the SVD based methods is $O(m(k+l)^2)$ with l the block size (we do not include *Brand IPCA* that ignores new directions). Chahlaoui, Gallivan, and Van Dooren [8, 7] discovered that using a *QR* representation can reduce this cost to $O(mk)$. Interestingly, many authors were not aware of this idea, even in papers that were written more then a decade later. Part of the contributions of this thesis is its incorporation into a recently developed algorithm called *FrequentDirections*. Because this idea is so important we give its details here:

2.2.1 QR-IPCA Specifics

Start with $Q \in \mathbb{R}^{m\times k}$, $R \in \mathbb{R}^{k\times k}$ $QR = A_{1:k}$ [1]. For each additional vector w, update the representation, generating Q_{new} and R_{new} as follows

1. Set $\rho = \|w - QQ^T w\|$ and $p = \|w - QQ^T w\|/\rho$
2. Expend: $\begin{bmatrix} QR & w \end{bmatrix} = \begin{bmatrix} Q & p \end{bmatrix} \begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix}$.

[1] They assume full rank always otherwise their method does not work

3. Find (U_{k+1}, σ_{k+1}), the smallest singular pair of

$$\begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix}$$

4. Find G_u such that $G_u^T U_{k+1} = e_{k+1}$

5. Find orthogonal transformation G_v such that

$$R_{\text{up}} = G_U^T \begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix} G_V$$

   is upper triangular (using RQ decomposition[2])

Denote the right singular basis of R_{up} by $\tilde{V}$ and its left basis by $\tilde{U}$. R_{up} has the same singular values matrix S as

$$\begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix}$$

Let us explore the structure of R_{up}:

$$R_{\text{up}}^T e_{k+1} = R_{\text{up}}^T \tilde{u}_{k+1} = \tilde{V} S \tilde{U}^T \tilde{u}_{k+1} = \tilde{V} S e_{k+1} = \sigma_{k+1} \tilde{v}_{k+1}$$

R_{up} is upper triangle so

$$R_{\text{up}}^T e_{k+1} = \begin{bmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & R_{up_{,(k+1,k+1)}} \end{bmatrix} e_{k+1} = R_{\text{up},(k+1,k+1)} e_{k+1} = \sigma_{k+1} \tilde{v}_{k+1}$$

[2] RQ decomposition is done by combining the regular QR with some algebraic manipulations

implies that $\tilde{v}_{k+1}$ is also e_{k+1} and that $R_{up,(k+1,k+1)} = \sigma_{k+1}$. In addition, we have

$$\begin{aligned} R_{\text{up}}\tilde{v}_{k+1} &= \\ &= \begin{bmatrix} \tilde{U}_{1:k} & e_{k+1} \end{bmatrix} \begin{bmatrix} S_{1:k} & \\ & \sigma_{k+1} \end{bmatrix} \widetilde{V}^T \tilde{v}_{k+1} \\ &= \begin{bmatrix} \tilde{U}_{1:k} & e_{k+1} \end{bmatrix} \begin{bmatrix} S_{1:k} & \\ & \sigma_{k+1} \end{bmatrix} e_{k+1} \\ &= \sigma_{k+1} \begin{bmatrix} \tilde{U}_{1:k} & e_{k+1} \end{bmatrix} e_{k+1} \\ &= \sigma_{k+1} e_{k+1} \end{aligned}$$

So $R_{\text{up}}\tilde{v}_{k+1}$=$R_{\text{up}}e_{k+1} = \sigma_{k+1}e_{k+1}$ and again because R_{up} is upper triangular

$$\begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & \sigma_{k+1} \end{bmatrix} e_{k+1}=\sigma_{k+1}e_{k+1}$$

implies that elements $1:k$ in the last column of R_{up} are all equal zero. We conclude that the structure of R_{up} is

$$\begin{bmatrix} R_{new} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix}$$

With $R_{\text{new}} \in \mathbb{R}^{k\times k}$, an upper triangular matrix. We now continue with the incremental update:

(6) Deflate:

$$\begin{aligned} \begin{bmatrix} Q & p \end{bmatrix} \begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix} &= \\ &= \begin{bmatrix} Q & p \end{bmatrix} G_u \cdot G_u^T \begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix} G_v \cdot G_v^T \\ &= \begin{bmatrix} Q & p \end{bmatrix} G_u \begin{bmatrix} R_{new} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix} G_v^T \end{aligned}$$

7. Set: $\tilde{Q} = \begin{bmatrix} Q & p \end{bmatrix} G_u$ and finish the update by truncating its last column:

$$Q_{\text{new}} = \tilde{Q}(:, 1:k) \in \mathbb{R}^{m \times k}$$

2.2.2 *Equivalence Between QR-IPCA and the SVD Based Methods*[3]

At initialization, $A_{\text{init}} = Q_{init} R_{init}$. Setting $U_R S_R V_R^T = SVD(R_{init})$ we get that $(Q_{init} U_R) S_R V_R^T$ is the exact SVD of A_{init}. At the first step, the matrix R is comprised of three matrices: an orthogonal matrix U_R that rotates Q into the left singular basis of A_{init}, S, the singular values matrix of A_{init} and another orthogonal matrix relevant to the right singular basis which we don't use. Let us denote

$$\tilde{Z} = \begin{bmatrix} R & Q^T w \\ \bar{0} & \rho \end{bmatrix}$$

For each other column,we showed that

$$G_U^T \tilde{Z} G_V = \begin{bmatrix} R_{new} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix}$$

The operation $G_U^T \tilde{Z} G_V$ does not change the singular values of $\tilde{Z}$ (as both G_U and G_V are orthogonal), so the singular values of R_{new} are the same as the top k singular values of $\tilde{Z}$. Before truncating $\tilde{Q} = \begin{bmatrix} Q & p \end{bmatrix} G_u$ and dropping G_V^T the update is still exact. We have:

[3] This equivalence was first suggested by Baker [2]

$$
\begin{aligned}
\begin{bmatrix} QR & w \end{bmatrix} &= \tilde{Q} R_{\text{up}} G_v^T \\
&= \begin{bmatrix} \tilde{Q}_{1:k} & \tilde{q}_{k+1} \end{bmatrix} \begin{bmatrix} R_{new} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix} G_v^T \\
&= \begin{bmatrix} \tilde{Q}_{1:k} & \tilde{q}_{k+1} \end{bmatrix} \begin{bmatrix} U_{R_{new}} S_{R_{new}} V_{R_{new}}^T & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix} G_v^T \\
&= \begin{bmatrix} \tilde{Q}_{1:k} & \tilde{q}_{k+1} \end{bmatrix} \begin{bmatrix} U_{R_{new}} & \bar{0} \\ \bar{0} & 1 \end{bmatrix} \begin{bmatrix} S_{R_{new}} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix} \begin{bmatrix} V_{R_{new}} & \bar{0} \\ \bar{0} & 1 \end{bmatrix}^T G_v^T \\
&= \begin{bmatrix} \tilde{Q}_{1:k} U_{R_{\text{new}}} & \tilde{q}_{k+1} \end{bmatrix} \begin{bmatrix} S_{R_{\text{new}}} & \bar{0} \\ \bar{0} & \sigma_{k+1} \end{bmatrix} \begin{bmatrix} V_{R_{\text{new}}} & \bar{0} \\ \bar{0} & 1 \end{bmatrix}^T G_v^T
\end{aligned}
$$

which is the SVD of $\begin{bmatrix} QR & w \end{bmatrix}$ with $\tilde{Q}_{1:k} U_{R_{\text{new}}}$, the left, rank-$k$, singular basis of $\begin{bmatrix} QR & w \end{bmatrix}$. We conclude that for each new vector, *QR-IPCA* finds in-explicitly the left singular basis and the associated singular values matrix of the expended matrix $\begin{bmatrix} QR & w \end{bmatrix} = \begin{bmatrix} USV^T & w \end{bmatrix}$ with U the left singular basis of the previous step. The presence of V has no effect on the approximation of the left basis. *QR-IPCA* and the *Basic IPCA* are equivalents.

The big advantage of *QR-IPCA* over the *Basic IPCA* is its complexity: $8mk$+$O(k^3)$. Recall that the most expensive part in the *Basic IPCA* was the rotation $\tilde{U}_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} U_Z$ with cost of $O(mk^2)$. QR-IPCA has a similar rotation $\tilde{Q} = \begin{bmatrix} Q & p \end{bmatrix} G_u$. As noted before, *QR-IPCA* update cost is linear, $O(mk)$ and not quadratic. The key to the reduced cost is the fact that G_u is a single Householder reflection, which implies that we can apply it to a m-by-$(k+1)$ matrix at a total cost of only $\Theta(mk)$. To summarize the operation count, we have $4mk$ for the Gram Schmidt, the rotation $\begin{bmatrix} Q & q \end{bmatrix} G_u$ costs another $4mk$ when using Householder transformations (or modified Givens rotations) and the rest of the work: finding G_v (via RQ decomposition), calculation of smallest singular pair and forming $G_u[\cdot]G_v$, all costs $O(k^3)$. Chahlaoui, Gallivan, and Van Dooren also suggested a 2-sided approximation with a cost of $10mk$. They also suggested a way, by using some assumptions, to conduct post-run estimation of the approximation accuracy. Baker (addressed next) showed that the assumptions and accuracy bound do not neces-

sarily hold .

2.3 *Unification of SVD Based and QR Based Methods and Extension to Blocks*

Baker [2] suggested a unified model for 2-sided update showing that the work done from the SVD type updates and the QR based update belong to the same family, specifically addressing the 1-by-1 version of Chandrasekaran, Manjunath et el [9, 16] and the block version of Levy and Lindenbaum [14]. His work mainly continues the work of Chahlaoui, K. Gallivan, and P. Van Dooren [8, 7] and extends it into blocks. He is able to present several different updates with cost of $10mk$ per step (same as the two sided version of Chahlaoui, K. Gallivan, and P. Van Dooren but with variable block size, potentially allowing other type of block wise optimizations). He also showed that the assumptions made by Chandrasekaran, Manjunath et el. regarding the correct value for setting the cutoff threshold is $\varepsilon/\sqrt{n}$ (the value they heuristically recommended) as oppose to ε/n (the value they were able to prove). He also improved the ability to estimate the quality of the approximation after its conclusion. Later, Baker joins Gallivan and P. Van Dooren [3] and together they extended the work to multi-pass algorithms.

2.4 **FrequentDirections***: Error Control by Eigenvalue Annihilation*

Liberty, under the scope of matrix sketching, suggested an algorithm called *FrequentDirections*. This initial paper was followed with another one by Ghashami and Liberty et el. [15, 12]. The algorithm first calculate, in an incremental manner, a sketch B. Performing SVD on B provides the approximated rank-k PCA.

FrequentDirections[4] maintains and outputs a matrix $B \in \Re^{m\times k+1}$ with $b_{k+1} = \bar{0}$, an all zeros column. *FrequentDirections* process each column 1-by-1, according to the following

1. Initialize $B \leftarrow 0^{m\times k}$

[4] original *FrequentDirections* works row wise and generates a sketch with $l > k$ rows, from which the rank k approximation is generated. Here we will work column wise and the sketch will have $k+1$ rows

2. $B_{temp} = \begin{bmatrix} B_{(:,1:k)} & w \end{bmatrix}$

3. $USV^T = SVD(B_{temp})$

4. $\delta = S^2_{k+1,k+1}$

5. $S_{shrunk} = \sqrt{S^2 - \delta I_{k+1}}$

6. $B = US_{shrunk} = \begin{bmatrix} B_{(:,1:k)} & \bar{0} \end{bmatrix}$

Note that *FrequentDirections* is exactly the *Straightforward IPCA* suggested earlier, with the addition of a shrinkage step to the approximate singular values. Recall that Levy and Lindenbaum [14] suggested a similar approach. They suggested applying a fixed decay rate $0 < \lambda \leq 1$ on the singular values from update to update (for them $\tilde{S}_{\text{shrunk}} = \lambda S$). While Levy and Lindenbaum suggestion was heuristic (and was mainly intended to provide more emphasis on recent vectors), Liberty's was not. *FrequentDirections* is the only fixed rank IPCA method that has approximation guarantees. Liberty proved the following bounds

$$\|A^T A - B^T B\|_2 \leqslant \|A - A_{\bar{k}}\|_F^2 / \left(k - \bar{k}\right)$$

$$\|A - UU^T A\|_F^2 \leqslant \left(1 + \frac{\bar{k}}{k - \bar{k}}\right) \|A - A_{\bar{k}}\|_F^2$$

With U the rank k basis found by *FrequentDirections* andA_k, A's best rank-k approximation.

Regarding *FrequentDirections* complexity, we quote from Ghashami and Liberty et el [12](with some minor adaptations to match the notation and column wise operation vs row wise in this thesis): "Each iteration of the Basic *FrequentDirections* is dominated by the computation of SVD(B_{temp}). The standard running time of this operation is $O(mk^2)$. Since this loop is executed once per column in A the total running time would naively be $O(mnk^2)$. However, note that the sketch matrix B actually has a very special form. The first k columns of B are always orthogonal to one another. This is a result of the sketch having been computed by an SVD in the previous iteration. Computing the SVD of this matrix is possible in $O(mk)$ time using the Gu Eisenstat procedure [13]. This requires using the Fast

Multiple Method (FMM) and efficient multiplication of Cauchy matrices by vectors which is, unfortunately, far from being straight forward. It would have been convenient to use a standard SVD implementation and still avoid the quadratic term in k in the running time. We show below that this is indeed possible at the expense of doubling the space used by the algorithm".

In order to reduce the cost, Liberty suggests another algorithm, going from 1-by-1 update to block based. Instead of inserting one column he now inserts a block of size k, he then performs the SVD on the enlarged matrix. δ is still set the same but now $S_{shrunk} = \sqrt{max(S^2 - \delta I_{2k+2}, 0)}$. This alternative algorithm uses double the space and still has a quadratic cost per update but the amortized per-update cost is now linear. In order to avoid the periodic $O\left(mk^2\right)$ cost, Liberty suggested doubling the memory once again, maintaining two separate sketches that will be merged at end of run: "The total running time of the Algorithm is $O(mnk)$ and the amortized running time per row update is $O(mk)$. However, the worst case update time is still $\Omega(mk^2)$ in those cases where the SVD is computed. Using the fact that *FrequentDirections* sketches are mergeable, we can actually use a simple trick to guarantee a worst case $O(mk)$ 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 column in the input, we first add it to the active sketch and then spend $O(mk)$ floating point operations in completing the SVD of the sketch in maintenance mode. After $k+1$ updates, the active sketch runs out of space and must go into maintenance mode. But, in the same time, a total of $O(mk^2)$ floating point operations were invested in the inactive sketch which completed its SVD computation. At this point, we switch the sketch roles and continue. Once the entire matrix was processed, we combine the two sketches using their mergeable property". To summarize, in order to guarantee a worst case $O(mk)$ update time, Liberty suggests doubling the required memory twice.

Ghashami et el [10] conducted comparative accuracy tests for a large number of matrix sketching algorithms (existing and new variants). They mentioned that while *FrequentDirections* is the only algorithm that has bounds, in practice, SVD based methods typically works better. They suggested several variants to *FrequentDirections* with the goal of achieving better accuracy while maintaining *FrequentDirections* bounds. They also suggested an algorithm called *Generic*

FrequentDirections, unifying SVD based methods with *FrequentDirections.* Interestingly, They to did not included the work by Chahlaoui, K. Gallivan, and P. Van Dooren [8, 7] and later by Baker [2].

3 A General Framework for Incremental Low-Rank PCA

We have seen that many of the suggested fixed rank IPCA algorithms are very similar. We also saw that while some may look very different, they are conceptually the same (*QR-IPCA, Basic IPCA* and the *straightforward IPCA*). Baker suggested a framework that unified QR based methods with SVD based methods. Ghashami et el [12] unified *FrequentDirections* with SVD based methods. Non included the work by Brand and no connection between *FrequentDirections* and *QR-IPCA* was made. We suggest a more general framework for Incremental PCA methods with focus on what is done and not how. This high level abstraction allows wider range of unification and an easier performance analysis .

3.1 The Basic Framework[1]

The framework that we propose is a parameterized family of algorithms that take as input a matrix (data set) $A \in \mathbb{R}^{m\times n}$ and a target approximation rank k. The algorithms all output an approximate basis $B \in \mathbb{R}^{m\times k}$ for the dominant subspace of A (the subspace spanned by the left singular vectors associated with the largest singular values). We denote the jth column of A by a_j and similarly for other matrices.

Instantiation of an algorithm from the framework requires the specification of two functions (subroutines), a function $f : (\mathbb{R}^{m\times k}, \mathbb{R}^m) \longrightarrow \mathbb{R}^m$ that we refer to

[1] While our framework is for a 1-by-1, fixed rank updates, it can be extended to block wise and dynamically changing rank

as the *filter* and a function $g : \mathbb{R}^{k+1} \longrightarrow \mathbb{R}^k$ that we refer to as the *reweighter*. Informally, the role of f is to transform a_{t+1} before it is concatenated to B_t and the role of g is to transform the approximate singular values. In some cases, f and g use a set of parameters. Algorithms in the framework operate as follows.

1. Initialize $B = B_t$ to be $B_t = U_t S_t$ where U_t and $S_t = \text{diag}(s_t)$ are the k dominant left singular vectors and diagonal matrix of the k dominant singular values of A_t, where A_t consists of the first t columns of A for some $t \geq k$. This is equivalent to defining U_t and S_t as the singular factors of the best rank-k approximation of A_t.

2. For columns $t+1$ to n of A, perform the following steps:

   (a) Compute $w = f(B_t, a_{t+1})$.

   (b) Generate B_{t+1} with the following properties

      i. $B_{t+1} \in \mathbb{R}^{m \times k}$.

      ii. The left singular vectors of B_{t+1} are the first k left singular vectors of $\begin{bmatrix} B_t & w \end{bmatrix}$

      iii. The singular values of B_{t+1} are the first k singular values of $\begin{bmatrix} B_t & w \end{bmatrix}$ after going threw the *reweighter*.

We will denote U_t and S_t to be the left singular vectors and values of B_t and $\tilde{U}_{t+1}\tilde{S}_{t+1}\tilde{V}_{t+1}^T$ the SVD of $\begin{bmatrix} B_t & w \end{bmatrix}$. Some of the algorithms in the framework represent B_{t+1} implicitly as the product of certain matrices.

3.2 Examples of Filters *and* Reweighers

We now define a few filter functions and reweighing functions that are used by existing incremental PCA methods.

- The filter used in the *Basic IPCA [20, 5]* and in *QR-IPCA [7, 8]* is the *identity filter*

$$f_{\text{ID}}(B_t, a_{t+1}) = a_{t+1} \ .$$

- The filter in Brand's method [6]

$$f_{\text{Brand}}(B_t, a_{t+1}) = \begin{cases} U_t U_t^T a_{t+1} & \text{rank}(U_t) = k \\ a_{t+1} & \text{otherwise .} \end{cases}$$

- The *truncation filter* takes a threshold τ and projects a_{t+1} if the projection is a good approximation of it; Brand uses this filter but with a fixed parameter that is close to some application-dependent noise floor. We define it more generally here,

$$f_{\text{truncate}}(B_t, a_{t+1}) = \begin{cases} U_t U_t^T a_{t+1} & \|a_{t+1} - U_t U_t^T a_{t+1}\| < \tau \\ a_{t+1} & \text{otherwise .} \end{cases}$$

- The *reweighter* used in the *Basic IPCA* and in *QR-PCA* is the *identity reweighter*

$$g_{\text{ID}}(\tilde{S}) = \text{diag}\,(\tilde{s}_{1:k})$$

  (note that this is not exactly an identity function; it is an identity on the dominant singular values and it drops the smallest).

- Liberty defines the *eigenvalue-anihilation reweighter [15]* to be

$$g_{\text{eva}}(\tilde{S}) = \text{diag}\left(\sqrt{\tilde{s}_{1:k}^2 - \tilde{s}_{k+1}^2}\right) .$$

  This function shifts the eigenvalues of $\tilde{S}\tilde{S}^T$ so that the smallest is annihilated and then drops the smallest, which is now zero. Liberty did not name this function.

- The *tracking reweighers* was suggested by Levey and Lindenbaum *[14]* It takes a parameter $0 < \lambda \leq 1$, shrinks the singular values by factor of λ, and drops the smallest,

$$g_{\text{track}}(\tilde{S}) = \lambda \cdot \text{diag}\,(\tilde{s}_{1:k}) .$$

We denote an algorithm that uses f_x and g_y as $\text{IPCA}_{x,y}$.

4 Impossibility Results

We are interested in bounds which measure the quality of the approximation of the dominant subspace. We start this chapter with the presentation of a common error measure and explain its limitation in high dimensional settings. We then introduce a new error criterion we call the *subspace reconstruction bound* (or just *reconstruction bound*) that we believe is better suited for datasets with $k \ll m$. We then show that under this error criterion all of the algorithms presented here that belongs to the suggested framework are heuristic, in the sense that all can fail even in very simple cases.

4.1 A New Error Criterion

A common error measure is the projection norm:

$$E_{\text{proj}}(A,U) = \frac{||A - UU^T A||_F}{||A||_F}$$

The focus of this work is high dimensional data. The following claim explain (why in our eyes) the projection norm is not suited for those scenarios

Claim 1. Denote the singular values of A by σ_i. Assume that $\sigma_{k+1} = \sigma_{k+2} = \cdots = \sigma_m = \sigma_{\text{noise}} \neq 0$. In addition let A_k be A's best rank-k approximation. For any fixed k, the error $E_{\text{proj}} \to 1$ As $m \to \infty$.

Proof. From properties of SVD we know that $\|A - A_k\|_F = \sum_{i=k+1}^{m} \sigma_i^2$. We have

$$E_{\text{pro}\,j} = \sqrt{\frac{(m-k)\sigma_{\text{noise}}^2}{(m-k)\sigma_{\text{noise}}^2 + \sum_{i=1}^{k} \sigma_i^2}} \geqslant \sqrt{\frac{(m-k)\sigma_{\text{noise}}^2}{(m-k)\sigma_{\text{noise}}^2 + C}} \xrightarrow{m\to\infty} 1$$

□

E_{proj} does not provide useful bounds in high-dimensional noisy problems. The reason is simple: $\|A - A_k\|_F^2 = \sum_{k+1}^{m} \sigma_i^2$, so if m is large and if the singular values do not decay quickly to insignificant values, $\|A - A_k\|_F^2$ may be large even for good approximations, say $\text{span}(U) = \text{span}(A_k)$. This means that this bound cannot distinguish between good approximations and bad ones.

Another common error criterion is the *2-norm*

$$E_{2,\text{proj}}(A,U) = \frac{||A - UU^T A||_2}{||A||_2}$$

$E_{2,\text{proj}}$ does not suffer from the problem of E_{proj} as $E_{2,\text{proj}} = \sigma_{k+1}/\sigma_1$. $E_{2,\text{proj}}$ does not accumulate the errors due to the tail of the small, noise related singular values. No bounds that uses $E_{2,\text{proj}}$ with a fixed rank update are available. One interesting but not particularly useful bound that uses E_2 is the work of Chandrasekaran et el., [9, 16]. They show how to keep track of the error and they propose to increment k whenever necessary to preserve the bound. However, their method results in very high a-priori bounds for k; in many interesting cases, their bound is equivalent to maintaining the full rank.

We use an error criterion we call *Reconstruction Error. We* define it as:

$$E_{\text{recon}}(A_{\bar{k}},U) = \frac{||A_{\bar{k}} - UU^T A_{\bar{k}}||_F}{||A_{\bar{k}}||_F}$$

This criterion measures how well U spans the dominant subspace of A. Note that U has rank k and that $A_{\bar{k}}$ has rank $\bar{k} \leq k$. We note that if the gap between $\sigma_{\bar{k}}$ and σ_k is small, the problem of finding a U with a small reconstruction error is highly ill conditioned, because small perturbations in A can cause dramatic changes in $A_{\bar{k}}$; we feel that this is acceptable when the sought-after object is the dominant subspace. Unfortunately, it turns out that guaranteeing a small reconstruction error is impossible for all of the existing algorithms, including *FrequentDirections*, even in easy cases.

4.2 Impossibility Results

We present two groups of theorems, the first will revolve around a matrix with a given size. We will show that for many algorithms, given a matrix with a known size, the user can not be certain that the approximation error will not be $E_{\text{recon}}(A_{\bar{k}}, U) = 1$. The second group will provide similar results but here the focal point is the gap between the singular values and the effects that high dimension has on the approximation. The first group is answering the quotation: " I have a specific matrix at hand, can I be certain to get a proper approximation using an IPCA?" The second type answers a different question: "I know I have a big gap between the dominant subspace and the rest of the noise related singular values, can I be certain to get a proper approximation using an IPCA". Unfortunately, the answer to both, is no.

4.2.1 Building Blocks For Counter Examples

Let us define the following matrices, to be used later:

$$A_1(m,n,x) = \begin{bmatrix} x & \cdots & x \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}_{m\times n} \quad A_2(m,k,x) = \begin{bmatrix} 0 & \cdots & 0 \\ x & & \vdots \\ & \ddots & 0 \\ 0 & & x \\ 0 & \cdots & 0 \end{bmatrix}_{m\times k} \quad A_3 = \begin{bmatrix} 0 & \cdots & 0 \\ 0 & & \vdots \\ & \ddots & 0 \\ 0 & & 0 \end{bmatrix}_{m\times n}$$

and their extensions: $A_{1,i}$ same as A_1 with the difference that the row of the nonzero elements is in the i'th location (so $A_1 = A_{1,1}$) and $A_{2,i}$, which is the same as A_2 but with the difference that the first non zero value starts at row i (so $A_{2,2} = A_2$). In addition let us define

$$A_4(m_1,x_1,m_2,x_2) = \begin{bmatrix} x_1 \cdot I_{m_1} & \\ & x_2 \cdot I_{m_2} \end{bmatrix}_{(m_1+m_2)\times(m_1+m_2)}$$

a diagonal matrix with two groups of values on its diagonal.

We use three very simple properties of SVD. The first is that the SVD decom-

position of $A_{1,i}(m,n,x)$ is

$$U = \bar{e}_i \quad S = \left[x\sqrt{n}\right]_{1\times 1} \quad V^T = \left[\frac{1}{\sqrt{n}} \quad \frac{1}{\sqrt{n}} \quad \cdots \quad \frac{1}{\sqrt{}}\right]_{1\times n}$$

The second property is the extension of the first, where we look at the SVD of the composed matrix

$$A = \left[A_{1,1}(m,n_1,x_1) \quad A_{1,2}(m,n_2,x_2) \quad \cdots \quad A_{1,i}(m,n_i,x_i)\right]$$

which is (up to permutations making sure the diagonal elements of S are non decreasing)

$$\begin{aligned}
U &= \left[\bar{e}_1 \quad \bar{e}_2 \quad \cdots \quad \bar{e}_i\right]_{m\times i} \\
S &= \begin{bmatrix} x_1\sqrt{n_1} & & & \\ & x_2\sqrt{n_2} & & \\ & & \ddots & \\ & & & x_i\sqrt{n_i} \end{bmatrix} \\
&= \begin{bmatrix} S_1 & & \\ & S_2 & \\ & & S_i \end{bmatrix}_{i\times i} \\
V^T &= \begin{bmatrix} \frac{1}{\sqrt{n_1}} & \cdots & \frac{1}{\sqrt{n_1}} & 0 & \cdots & 0 & & & \\ 0 & \cdots & 0 & \frac{1}{\sqrt{n_2}} & \cdots & \frac{1}{\sqrt{n_2}} & 0 & \cdots & 0 \\ \vdots & & & & \vdots & & & & \\ 0 & \cdots & & & \cdots & 0 & \frac{1}{\sqrt{n_i}} & \cdots & \frac{1}{\sqrt{n_i}} \end{bmatrix} \\
&= \begin{bmatrix} V_1^T & 0 & \cdots & 0 \\ 0 & V_2^T & 0 & 0 \\ & & \ddots & \\ 0 & \cdots & 0 & V_i^T \end{bmatrix}_{i\times(n_1+n_2+\cdots+n_i)}
\end{aligned}$$

The last property addresses fixed rank incremental update. If we have a basis U and singular values matrix S and if the vector w is orthogonal to U then the exact

SVD of $\begin{bmatrix} US & w \end{bmatrix}$ is (up to permutation)

$$\begin{bmatrix} US & w \end{bmatrix} = \begin{bmatrix} U & \frac{w}{\|w\|} \end{bmatrix} \begin{bmatrix} S & \\ & \|w\| \end{bmatrix} \begin{bmatrix} I_k & \\ & 1 \end{bmatrix}$$

The size of $\|w\|$ determine the structure of the new basis. If $\|w\| > S_{k,k}$ then the unit vector $w/\|w\|$ is included in the updated U and U's last column will be dropped. If $\|w\| < S_{k,k}$ then U will stay without change.

4.2.2 Impossibility Results for a Given Matrix

Theorem 1. *Denote by U the rank-k basis found by IPCA$_{ID,ID}$ after processing $A \in \mathbb{R}^{m \times \tilde{n}}$. For any given m , $\bar{k} \leq k \leq m - \bar{k}$ and $\tilde{n} \geq k + 2\bar{k}$, there exist a matrix for which $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: Let us start with the case $\bar{k} = 1$. $A = \begin{bmatrix} A_2(m,k,x_2) & A_1(m,n,x_1) \end{bmatrix}$ and $k \leq m - 1$. Let $x_2 \xrightarrow{-} \sqrt{2}$ and $x_1 = 1$. Using the first property above, it is easy to see that if $n \geq 2$ then $\sigma_1 = \sigma_{\bar{k}} = \sqrt{n}$. After the first k columns, the approximated base matrix and the singular value matrix will be:

$$U_k = \begin{bmatrix} \bar{e}_2 & \bar{e}_3 & \cdots & \bar{e}_{k+1} \end{bmatrix}_{m \times k} S_k = \begin{bmatrix} x_2 & \cdots & 0 \\ 0 & \ddots & \vdots \\ 0 & \cdots & x_2 \end{bmatrix}_{k \times k}$$

For each column $i = (k+1) : \tilde{n}$, the left rank-$(k+1)$ SVD basis and the singular values matrix of $\begin{bmatrix} B_{i-1} & f_{\text{ID}}(B_{i-1}, a_i) \end{bmatrix}$ will be

$$\begin{bmatrix} U_k & \bar{e}_1 \end{bmatrix} \quad \textit{and} \quad \begin{bmatrix} S_k & \\ & 1 \end{bmatrix}$$

The identity *reweighter* simply drops the smallest singular value - basis vector pair so $B_i = U_i S_i = U_k S_k$. IPCA$_{\text{ID,ID}}$ will not be able to find the only dominant base vector $\bar{e}_1$ and regardless of the approximation rank used (as long as its smaller them m) $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.

Let us continue the proof for $\bar{k} > 1$. Let $A = \begin{bmatrix} A_{2,\bar{k}+1} & A_{1,1} & A_{1,2} & \cdots & A_{1,\bar{k}} \end{bmatrix}$. If each $A_{1,i}$ has $n \geq 2$ columns then similar to before (using the second property), the first $\bar{k}$ singular values are all equal $\sqrt{n} > \sqrt{2}$. After the first k columns are processed we will have the same U_k and S_k as in the previous case. As long as $k \leq m - \bar{k}$, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.

This Theorem implies that the difference between m and k can be the number of missed dominant directions.

Theorem 2. *Denote by U the rank-k basis found by $IPCA_{ID,eva}$ after processing $A \in m \times \tilde{n}$. For every m, $\bar{n} \geq m$ and $\bar{k} \leq k \leq m - (3k - 2\bar{k} + 3)$ there exist a matrix for which $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof. Let $\tilde{A} = A_4(\bar{k}, 1, m - \bar{k}, 0.6)$ and note that $0.6 > \sqrt{1 - 2 \times 0.6^2}$. After the first k columns we have:

$$U_k = \begin{bmatrix} \bar{e}_1 & \bar{e}_2 & \cdots & \bar{e}_{\bar{k}} & \bar{e}_{\bar{k}+1} & \cdots & \bar{e}_k \end{bmatrix} \quad S_k = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 0.6 & & & \\ & & & & \ddots & & \\ & & & & & 0.6 & \end{bmatrix}$$

The first $\bar{k}$ basis vectors are the correct ones. After the next column, we have

$$S_{k+1} = \begin{bmatrix} \sqrt{1 - 0.6^2} & & & & & \\ & \ddots & & & & \\ & & \sqrt{1 - 0.6^2} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

The next $k-\bar{k}$ columns will fill with 0.6 the zeros on the diagonal

$$S_{2k-\bar{k}+1} = \begin{bmatrix} \sqrt{1-0.6^2} & & & & & \\ & \ddots & & & & \\ & & \sqrt{1-0.6^2} & & & \\ & & & 0.6 & & \\ & & & & \ddots & \\ & & & & & 0.6 \end{bmatrix}$$

The next column will reduce again the size of the first $\bar{k}$ elements and place zeros in all the remaining elements

$$S_{2k-\bar{k}+2} = \begin{bmatrix} \sqrt{1-2\times 0.6^2} & & & & & \\ & \ddots & & & & \\ & & \sqrt{1-2\times 0.6^2} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

The next $k-\bar{k}$ columns will again fill the zeros but now with $0.6 > \sqrt{1-2\times 0.6^2}$

$$S_{3k-2\bar{k}+2} = \begin{bmatrix} 0.6 & & & & & \\ & \ddots & & & & \\ & & 0.6 & & & \\ & & & \sqrt{1-2\times 0.6^2} & & \\ & & & & \ddots & \\ & & & & & \sqrt{1-2\times 0.6^2} \end{bmatrix}$$

And the next column will drop the $\bar{k}$ first direction from the representation

$$S_{3k-2\bar{k}+3} = \begin{bmatrix} 0.6 & & & & & \\ & \ddots & & & & \\ & & 0.6 & & & \\ & & & 0.6 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

If $\tilde{n} > m$ then using $A = \begin{bmatrix} \tilde{A} & A_3 \end{bmatrix}$ (filling the remaining columns with all zeros matrix) completes the proof □

Theorem 3. *Denote by U the rank-k basis found by $IPCA_{Brand,ID}$ after processing $A \in m \times \tilde{n}$. For any given k, $\bar{k}$, $\tilde{n}$ and m such that $\tilde{n} \geq m \geq k + \bar{k}$, there exist a matrix for which $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: use $A_4(k, x_1, \bar{k}, x_2)$ with extra $m - (k + \bar{k})$ all zeros rows. As long as $x_2 > x_1$ we have $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.

All of the above theorems shows that almost in any matrix setting there are infinite number of matrices for which $E_{\text{recon}}(A_{\bar{k}}, U) = 1$ however they do not take notice of the gap $\sigma_{\bar{k}}/\sigma_{\bar{k}+1}$ which indicates if a dominant subspace even exist. The next group of theorems will address this issue specifically.

4.2.3 Impossibility Results for a Given $\sigma_{\bar{k}}/\sigma_{\bar{k}+1}$ Ratio

Theorem 4. *For any positive real number M, and $\bar{k} \leq k$ there exist a matrix $A \in \mathbb{R}^{m \times \tilde{n}}$ with $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$ such that if U is the rank-k basis found by $IPCA_{ID,ID}$ after processing A, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: use $A = \begin{bmatrix} A_{2,\bar{k}+1}(m,k,x_2) & A_{1,1}(m,n,x_1) & \cdots & A_{1,\bar{k}}(m,n,x_1) \end{bmatrix}$ with $m \geq \bar{k} + k$. Choose x_2 to be any real number bigger then zero. Chose x_1 to be any positive real number smaller then x_2. Choose n such that $x_1\sqrt{n}/x_2 > M$.

Theorem 5. *For any positive real number M, and $\bar{k} \leq k$ there exist a matrix $A \in \mathbb{R}^{m \times \tilde{n}}$ with $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$ such that if U is the rank-k basis found by $IPCA_{ID,eva}$ after processing A, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: Let $A = A_4(\bar{k}, 1, m-\bar{k}, 1/M)$[1] . We have seen at theorem 2 that the operation of $\text{IPCA}_{\text{ID,eva}}$ in this case is periodic. First, all the elements in S_k are filled by two groups of values (and the initial basis matrix is generated). The first part with the $\bar{k}$ dominant elements (all equal one) and the remaining $k-\bar{k}$ with value $1/M$. The next column will reduce the first $\bar{k}$ to $\sqrt{1-1/M^2}$ and will zero out all the last $k-\bar{k}$ elements. Next $k-\bar{k}$ columns will fill the zeros and the next column will again reduce the value of the top $\bar{k}$ and zero out the lower $k-\bar{k}$ and so on. For any given ratio $\sigma_{\bar{k}}/\sigma_{\bar{k}+1}$ there exist $\tilde{m}$, the number of columns / rows (each with the singular value $1/M$) needed for reducing the dominant singular values below $1/M$. After another $k-\bar{k}+1$ columns they will be thrown away, along with their matching base vectors yielding $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.

Theorem 6. *For any positive real number M, and $\bar{k} \leq k$ there exist a matrix $A \in \mathbb{R}^{m\times\tilde{n}}$ with $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$ such that if U is the rank-k basis found by $IPCA_{truncate,ID}$ after processing A, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: use $A = \begin{bmatrix} A_{2,\bar{k}+1}(m,k,x_2) & A_{1,1}(m,n,x_1) & \cdots & A_{1,\bar{k}}(m,n,x_1) \end{bmatrix}$ with $m \geq \bar{k}+k$. Choose $x_2 = 1$, $x_1 < \tau$ and n such that $x_1\sqrt{n} > M$.

Theorem 7. *For any positive real number M, and $\bar{k} \leq k$ there exist a matrix $A \in \mathbb{R}^{m\times\tilde{n}}$ with $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$ such that if U is the rank-k basis found by $IPCA_{Brand,ID}$ after processing A, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: f_{Brand} does not allow new direction to enter after the first k were found. As such the result is immediate. Use $A = A_4(k, 1, \bar{k}, M)$.

Theorem 8. *For any positive real number M, and $\bar{k} \leq k$ there exist a matrix $A \in \mathbb{R}^{m\times\tilde{n}}$ with $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$ such that if U is the rank-k basis found by $IPCA_{ID,Track}$ after processing A, $E_{\text{recon}}(A_{\bar{k}}, U) = 1$.*

Proof: Use $A = \begin{bmatrix} A_4(\bar{k}, 1, m-\bar{k}, 0) & A_3(m,i) & A_{2,\bar{k}+1}(m,k,1/M) \end{bmatrix}$ with $m \geq \bar{k}+k$ and i such that $\lambda^i < 1/M$ (in fact i can be smaller due to the second block of A_4) .

[1] For simplicity we use $1/M$ rather then $(1-\varepsilon)/M$ needed for $\sigma_{\bar{k}}/\sigma_{\bar{k}+1} > M$

4.2.4 Discussion

Let us now try to understand why the task before IPCA is so hard. SVD is a global algorithm. As been seen in the counter examples, dominant directions can be comprised of many repetitions, each with locally low norm. For example, the base - singular value pair

$$U = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad S = 1$$

can be the exact rank-1 PCA of any of the following:

$$A = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}_{3\times 1} \qquad B = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{3\times 2} \qquad C = \begin{bmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \cdots & \frac{1}{\sqrt{n}} \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}_{3\times n}$$

As the number of columns n grows, the LOCAL contribution of a meaningful directions can be very low. We call this phenomenon *Mass Spreading*. For batch SVD, *mass spreading* has no effect. Due to its global view, it is able to accumulate all the contributions together. For incremental methods, at each step having to act based on knowledge regarding only the next vector (or small block of vectors), it potentially has huge effect. The goal of both $\text{IPCA}_{\text{Brand,ID}}$ and $\text{IPCA}_{\text{truncate,ID}}$ is to save operation count costs. It is obvious from its definition that $\text{IPCA}_{\text{Brand,ID}}$ does not allow new directions after the initial basis was established. *Mass spreading* explains why it is not possible to use any deterministic value of τ in $\text{IPCA}_{\text{truncate,ID}}$ without knowing it might fail completely. We have not seen any reference to *mass spreading* in any previous work.

In addition to *Mass Spreading*, IPCA faces another problem. It is a known fact that when we update an SVD approximation by adding a vector, the singular values of S are non decreasing. When our dataset includes very large number of columns, even the smallest singular value might get big enough to potentially prevent the incorporation of new dominant directions into the approximation. This is a known phenomenon, sometimes referred to as *Weight of Experience*. A simple illustration to the problem posed by *weight of experience* is as follows. Imagine

a situation where the real singular value of a noise related direction is σ_{k+1}. The dataset may have a very good low rank approximation with a big σ_k/σ_{k+1} ratio but with an average column norm lower then σ_{k+1}. If the noise direction was fully incorporated into U_t, meaning we will have σ_{k+1} in our approximated singular values matrix, it may be very difficult for the IPCA algorithm to replace it with a dominant direction as their local contribution is smaller then σ_{k+1}.

Mass spreading & weight of experience works hand in hand making the task of IPCA that much difficult and they drive the impossibility result for $\text{IPCA}_{\text{ID,ID}}$.

Weight of experience is countered in $\text{IPCA}_{\text{ID,track}}$ by the decay factor. (as it reduces the magnitude of our approximated singular values) however it is not fit as a method to obtain an approximation for A entirely. It's built-in forget factor will do exactly what it meant too. Dominant directions seen early in the dataset will be phased out, potentially clearing the room for noise related directions. For that reason $\text{IPCA}_{\text{ID,track}}$ is mainly mentioned in applications of subspace tracking where recent instances are more important then old ones, but not as a solution to a global approximation problem.

$\text{IPCA}_{\text{ID,eva}}$ uses a similar approach to $\text{IPCA}_{\text{ID,track}}$, also shrinking the singular values from update to update, but in a much more clever way. In fact, $\text{IPCA}_{\text{ID,eva}}$ is so clever it enables Liberty to bound its error (using E_{proj}), an achievement shared by non other fixed rank IPCA algorithm and with a proof much simpler then those arising from matrix sketching work [11]. Clever as it is, the shrinkage mechanism used by $\text{IPCA}_{\text{ID,eva}}$ comes with a cost, which is not evident if the error criterion is E_{proj} (the one used for its error bound). The cost of the shrinkage mechanism is that it makes it sensitive to high dimensional noise. Non-shrinking methods belonging to $\text{IPCA}_{\text{ID,ID}}$ suffer from the increase on columns (instances / observations) but $\text{IPCA}_{\text{ID,eva}}$ suffer from the increase of attributes of each column. While the counter example used in the impossibility results is very non-realistic, we will also demonstrate this drawback in the testing chapter, using much more realistic datasets and on a real world dataset.

5 New Algorithms

5.1 *QR-FD: A New QR Based Implementation of* FrequentDirections

$IPCA_{\text{ID,eva}}$ is an abstraction of *FrequentDirections [15, 12].* We have demonstrated that when using E_{recon} it is not possible to bound its error. Liberty did manage to bound the error using E_{proj}, making *FrequentDirections* the only fixed rank IPCA with error guaranties. The Basic *FrequentDirections* cost is $O\left(mk^2\right)$ per column. Liberty suggested doubling the memory used, going from a 1-by-1 update to block wise updates. This approach still has a quadratic cost per update but the amortized per-update cost is linear. In order to avoid the periodic $O\left(mk^2\right)$ cost, Liberty suggested doubling the memory once again, maintaining two separate sketches, merging them at end of run. We will now show a very simple trick that allows a 1-by-1 update with *FrequentDirections* guaranties and $O\left(mk\right)$ cost without the need to work with blocks and maintaining two different sketches. Recall that when we reviewed work related to $IPCA_{\text{ID,ID}}$ we mentioned that Chahlaoui et al [8] developed an 1-by-1 update algorithm with the best known complexity, only $8mk$ per update. They maintain B_t in-explicitly with a basis Q_t and an upper diagonal matrix R_t. The difference between $IPCA_{\text{ID,ID}}$ and $IPCA_{\text{ID,FD}}$ is the *reweighter* function. Our key observation is that it is possible to add $g_{eva}(\cdot)$ to the *QR* based $IPCA_{\text{ID,ID}}$ as follows:

1. Generate Q_{new} and R_{new} according to the *QR* based $IPCA_{\text{ID,ID}}$ but keep the discarded singular value $\sigma = \sigma_{k+1}$. Denote $R = R_{\text{new}}$

2. Compute the SVD of $R = U_R S_R V_R^T$

3. Expend $\tilde{S}_R = diag(S_R, \sigma)$

4. $S_{\text{shrunk}} = g_{\text{eva}}\left(\tilde{S}_R\right)$

5. Perform RQ decomposition on the reconstructed matrix: $\tilde{R} = U_R S_{\text{shrunk}} V_R^T = R_{\text{new}} Q_{\text{throw\_away}}$

6. Continue with next column

We call this new implantation $QR-FD$. Note that we drop the matrix $Q_{\text{throw\_away}}$ as it is related to the right singular base. The extra cost is $O\left(k^3\right)$ which is negligible in the setting this thesis is aiming at with $k \ll m \ll n$.

5.2 New Instantiations

We now suggest three new instantiations (abstract algorithms). The first, *TunableShrinkage,* tries to balance between the advantages and drawbacks of $\text{IPCA}_{\text{ID,ID}}$ and $\text{IPCA}_{\text{ID,eva}}$ while still enjoining the same guarantees of $\text{IPCA}_{\text{ID,eva}}$ (under E_{proj}). The other two, Boosted-IPCA (*BIPCA*) and Just-In-Time-PCA (*JIT-PCA*) are heuristic algorithms that were specifically engineered, via extensive testing, to be more robust then previously presented algorithms. We justify the use of heuristic algorithms by the fact that we have demonstrated that under E_{recon}, all methods are basically heuristic. For $k \ll m \ll n$ they are both faster then $QR-IPCA$. In low noise setting *JIT-PCA* operation count$\rightarrow 2mk$, the cost of projecting a vector into a subspace.

5.2.1 TunableShrinkage

We have seen that *Mass spreading & weight of experience* poses a problem for algorithms of type $\text{IPCA}_{\text{ID,ID}}$ while it is not apparent that high dimensional noise has any affect on it. On the over side, the reweighter of $\text{IPCA}_{\text{ID,eva}}$ enables it to handle *Mass spreading & weight of experience* but makes it sensitive to high dimensional noise. Let us define the following reweighter

$$g_{\text{r\_eva}}(S,r) = \sqrt{S^2 - I_k \cdot (\sigma^2/r)}$$

The above reweighter is exactly as in $\text{IPCA}_{\text{ID,eva}}$ with one exception, the elements on the diagonal of S^2 are reduced by σ^2/r and not σ^2. We call r the Shrinkage ratio. If we choose $r = 1$ we get g_{eva} and If we choose $r \to \infty$ we get g_{ID} How does the increase of the Shrinkage ratio changes the behavior of the update method? The next theorem will address this explicitly:

Theorem 9. *Let $B \in \mathbb{R}^{m\times k}$ be the sketch produced by TunableShrinkage, for any $\bar{k} < k/r$ and $r \geq 1$ it holds that*

$$\|A - U_{1:\bar{k}}U_{1:\bar{k}}^T A\|_F^2 \leq \left(1 + \frac{\bar{k}r}{k-\bar{k}r}\right)\|A - A_{\bar{k}}\|_F^2$$

Proof. See Appendix A □

TunableShrinkage come with the same guaranty of *FrequentDirections* except the fact that as the Shrinkage ratio increases, the sketch need to be bigger in order to get the same bound. Obviously from a certain value the sketch size will be too big for any practical use, however, we reiterate on the fact that for $k \ll m$ those bounds tell us very little about the performances anyway. As the Shrinkage ratio increases, the update scheme will be less sensitive to increase in dimension but will take longer to incorporate new directions requiring much more repetitions. As m increases, we need to use a bigger value for r which require us to increase n, the number of columns / instances in our dataset. This relation is of no surprise and seen in many different aspect when dealing with high dimensional data. We have no way to pre-define the proper value for the Shrinkage ratio and currently it should be defined by the user based on test on the specific dataset. The arithmetic cost of *TunableShrinkage*, if using the *QR* version, is $8mk$ per step.

5.2.2 Randomized Boosting

We now present two alternative algorithms that were specifically engineered, via extensive testing, to be more robust then previously presented algorithms. The First is called *Boosted IPCA (BIPCA)* and the second *Just In Time PCA (JIT-PCA)*. Both are without a tuning parameter and with a specific implantation, complexity cost varying between $2mk$ to $8mk$ pending on the problem structure (for $k \ll m$).

The new algorithms are using a randomized procedure and instead of manipulating S after each update (via the *Reweigher* function), manipulates the incoming vector prior to processing it (by the *filter* function). Denote $r = \left(I - U_t U_t^T\right) a_{t+1}$ and $\rho = \|r\|$. r can be zeroed out (similar to f_{truncate}), left untouched (using f_{ID}) or boosted (new operation). The low complexity is achieved via alternating between the methods of Chahlaoui et al and of Brand.

5.2.2.1 Boosted Incremental PCA (BIPCA)

This method uses the identity reweighter but with a sophisticated statefull randomized filter. The state that the filter maintains is the average mass of columns $\alpha_t = \|A_{:,1:t}\|_F^2/t$, the smallest singular value of B_t, denoted σ_t and a counter c. We initialize $c = 2$. The filter starts by tossing a biased coin with success probability $1/c$. If the coin toss is successful, the filter simply sets w to the projection $p = U_t U_t^T a_{t+1}$ and it increments c. Otherwise, the filter sets $c = 2$ and computes the projection-residual $r = a_{t+1} - U_t U_t^T a_{t+1}$ and its 2-norm ρ. If $\rho > \sigma_t$, we set $w = a_{t+1}$ and continue. If the residual is small $\rho \leq \sigma_t$, we toss another coin with success probability $1 - \min(1, \rho^2/\alpha_t)$. If the toss is successful, we again set $w = a_{t+1}$. If the coin toss is unsuccessful, we *boost* the residual and set $w = p + \beta r$ where

$$\beta = \begin{cases} \sigma_t/\rho + \varepsilon & p = 0 \\ \min\left(\sigma_t/\rho, \sqrt{(\|a_{t+1}\|^2 + \sigma_t^2)/\|a_{t+1}\|^2}\right) & \text{otherwise,} \end{cases}$$

where ε is infinitesimal (not a significant numeric value). The test $p = 0$ is done in a numerically-robust way (small p's are admitted). The ε term forces w to be retained in B_{t+1} when $p = 0$.

5.2.2.2 JIT-PCA

We have seen that the cost of $\text{IPCA}_{\text{Brand,ID}}$ can be $2mk$ but does not allow new directions once the initial rank-k base is found. *JIT-PCA* tries to narrow down the gap from $8mk$ to $2mk$ but still be able to incorporate new directions as needed. *JIT-PCA*, is closely related to *BIPCA* and uses the same notation. It is a little

simpler, often more efficient, but sometimes a little less accurate. It also uses an identity reweighter and a sophisticated filter. The filter tosses only one coin with probability $(1/c)(1-\min(1,\rho^2/\alpha_t))$ (the product of the probabilities in *BIPCA*). If the coin toss is successful, we set $w = p$ and increment c, otherwise we set $w = p + \gamma r$ and set $c = 2$, with γ defined as

$$\gamma = \begin{cases} 1 & \rho > \sigma_t \\ \beta & \text{otherwise} . \end{cases}$$

5.2.2.3 Complexity of *BIPCA* and *JIT-PCA*

The per step cost of the *Basic IPCA* is dominated by the $O\left(mk^2\right)$ required for updating $U_{t+1} = \begin{bmatrix} U_t & p \end{bmatrix} U_Z$. We have seen two ways to overcome it (when limiting the update to be 1-by-1). Brand maintains two matrices, $U_t \in \mathbb{R}^{m\times k}$, the basis found after the conclusion of step t, which does not change from time $i = t+1$ till end of run. The second is $U_{\text{small},i} \in \mathbb{R}^{k\times k}$ which is being updated by $U_{\text{small,i+1}} = U_{small,i}[U_Z]_{1:k,1:k}$. At end of run the output basis is then $U_t U_{\text{small,n}}$ [6]. Brand's algorithm weakens lie in the fact that it completely ignores new directions that might appear late in the stream. His strength is its cost, merely $2mk$ (for $k \ll m$). We will call this method a *span-only* update.

The second method to reduce the cost was suggested by Chahlaoui et al, using a *QR* updating structure with $U_i \in \mathbb{R}^{m\times k}$ orthonormal and $R_i \in \mathbb{R}^{m\times k}$ upper triangle [8]. Their algorithm takes advantage of efficient multiplication using Householder reflections, achieving the best known arithmetic cost for deterministic IPCA algorithms (besides Brand's, but they do not drop the orthogonal part) - only $8mk$ operations per update. Similarly to Brand's, U_i is not the final base. The SVD of R_i contains both the singular values and a rotation matrix U_R such that $U_t U_R$ is the approximate SVD base. We call this method *full-update*.

Both *BIPCA* and *JIT-PCA* uses coin tosses to decide whether or not to use the orthogonal part of the incoming vector. A small procedure, we call the *complexity trick,* permit *BIPCA* and *JIT-PCA* complexity to vary between $2mk - 8mk$ by alternating between the methods proposed by Brand and Chahlaoui et al.

Table 5.1 provides the details for the conversion needed for all possible [previ-

ous - currant] update pairs. Two cases are straight forward, if the previous vector update type and the current update type are the same, then one directly perform the update operation as originally suggested by Brand or Chahlaoui et al. For the two remaining cases *BIPCA* and *JIT-PCA* switches between the two types of representations either using a RQ decomposition (when the previous update type was *span-only* and the currant type is *full-update*) or a small SVD (in the reverse situation). Both of the operations are $O(k^3)$, negligible in our settings.

The average per-step complexity of BIPCA and JIT-PCA is determined by the ratio between *span-only* and *full-update*. BIPCA complexity is not dependent on the accuracy of the approximation or noise levels as the decision to use the orthogonal part is based on the ratio $1/c$. Its expected per-column cost is $\sim 5mk$. JIT-PCA probability to use the orthogonal part is chained to $(1 - \min(1, \rho^2/\alpha_t)$. On average, it will perform *full-update* less times then *BIPCA*. In cases where the basis found during the run by *JIT-PCA* is close to the real underlying basis, especially in low noise setting (where ρ^2/α_t is low), the per-step cost can get very close to $2mk$. In the testing chapter we will demonstrate that unlike Brand's, the accuracy cost is very small.

5.2.2.4 Discussion

The Randomized Boosting algorithms suggested are design to be more robust then previously suggested methods. They use a boost in order to overcome *Mass spreading & weight of experience*. If a direction is repetitive with low norm it will occasionally win the draw, get a boost and incorporated into the approximation. $\text{IPCA}_{\text{ID,eva}}$ and $\text{IPCA}_{\text{ID,track}}$ both shrinks all of the singular values, the boost is limited by the smallest singular value. This limitation should help preventing non dominant directions to overflow the approximation. We note that classical SVD is more oriented toward local norm then repetition. Both *BIPCA* and *JIT-PCA* are more oriented toward repetition. For example, a matrix made of two mutually orthogonal groups of columns. The first with many small norm columns and the second with very few huge norm columns. If the norm of the small orthogonal group is large enough, they can be the dominant directions for classical SVD. From the design perspective of *BIPCA* and *JIT-PCA*, the huge norm columns are considered as outliers. Neglecting them is a good thing. When we provided the

Previous update	**Currant update**	**Maintained data structures**	**Needed data structures**	**Transformation**
Span only	Span only	U, U_{small}, S	no change	no need
Full update	Full update	Q, R	no change	no need
Span only	Full update	U, U_{small}, S	Q, R	$R\tilde{Q} = U_{small}S$ via RQ decomposition $\tilde{Q}$ is not used and $Q = U$
Full update	Span only	Q, R	U, U_{small}, S	$U_{\text{small}}SV^T = R$ via SVD V is not used

Table 5.1: The Complexity Trick: best known *full-update* cost is $8mk$ and uses a QR structure with $Q \in \mathbb{R}^{m\times k}$ an orthogonal basis and $R \in \mathbb{R}^{m\times k}$ an upper triangular matrix. Best known *span-only* updates cost is $2mk$ and uses 3 matrices: a basis matrix $U \in \mathbb{R}^{m\times k}$, a small rotation matrix $U_{\textbf{small},i} \in \mathbb{R}^{k\times k}$ and a diagonal matrix of estimated singular values S. In order to minimize the cost, *BIPCA* and *JIT-PCA* switches between those two types of representations. The transformation cost is $O(k^3)$ negligible under the setting of this work

impossibility results for $\text{IPCA}_{\text{ID,eva}}$ we used exactly this type of matrix, several large norm vectors and the rest with much smaller norm. This was just for simplification. The same result could have been achieved by small repetitive vectors building up the large singular values. Due to this nature and design of *BIPCA* and *JIT-PCA* we do not believe it is possible to prove anything regarding its accuracy under conventional framework.

6 Experimental Results with Synthetic Datasets

We compare accuracy of several competing algorithms: The *Basic IPCA, FrequentDirections, TunableShrinkage, BIPCA* and *JIT-PCA*. When applicable, the error criterion used is E_{recon}. We conduct different types of tests in order to demonstrate the behavior addressed in the impossibility results. The datasets are all variants of the following generation model: $A = B_{m\times m} \cdot M_{m\times n}$ where B is a square orthogonal matrix (basis). M act as a mixing matrix and potentially divided into blocks $\begin{bmatrix} M_1 & M_2 & \cdots & M_j \end{bmatrix}$, each with optionally different number of columns. The structure of each block is:

$$M_i = \begin{bmatrix} \text{rows with entries from} \sim N(0,\sigma_n) \\ \cdots \\ \text{rows with entries from} \sim N(0,\sigma_d) \\ \cdots \\ \text{rows with entries from} \sim \text{N}(0,\sigma_n) \end{bmatrix}$$

with $\sigma_d > \sigma_n$. Basis column corresponding to rows of M_j with elements from standard deviations σ_n are noise related and those corresponding to the standard deviations σ_d are data (dominant basis) columns. Several blocks can share the same dominant basis (same rows contain elements from standard deviations σ_n and σ_d) or each can have a different one (different rows). The resulting matrix A will have two groups of singular values. A group of strong ones with more or less equal values and a group of weak singular values, again more or less with the same value

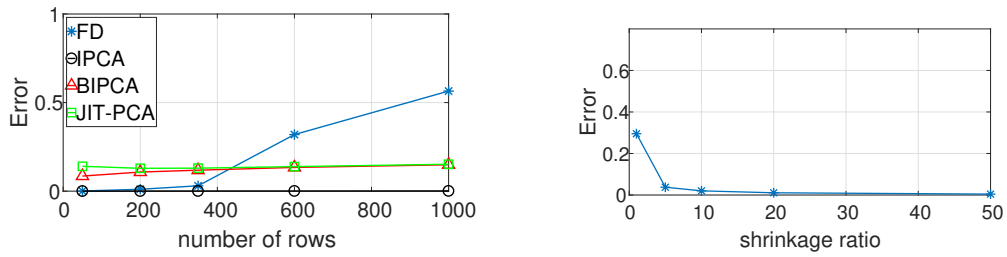

Figure 6.1: The effects of increasing the dimension on the approximation accuracy. Left: The Basic IPCA performs best with $E_{\text{recon}} \approx 0$, BIPCA and JIT-PCA performs equally with ≈ 0.1 and *FrequentDirections* error increase as the dimension (row) increases. Right: the row was fixed at 600. Adjusting the shrinkage ratio allows *TunableShrinkage* to accurately approximate the dominant subspace.

6.1 The Effects of Increasing the Dimension

We randomly generated a dataset $A = B_{\text{row}\times\text{row}} M_{\text{row}\times 3000}$ with row $= 50...1000$. The first five rows of M contain entries drawn from $\sim N(0,1)$ and the remaining rows have entries drawn from $\sim N(0,0.1)$ (so $\sigma_d = 1$, $\sigma_n = 0.1$ and $\bar{k} = 5$). We used $k = 9$ as the approximation rank for all algorithms. The left side of Figure 6.1 demonstrates that on this type of dataset where the underlying basis does not change, the Basic IPCA outperforms the rest with an almost exact approximation. BIPCA and JIT-PCA both have $E_{\text{recon}} \approx 0.1$ and are generally unaffected by dimension increase. *FrequentDirections* demonstrates its sensitivity to high dimension, starting with a very good approximation when A only has 50 rows but reaching 0.8 when A has 1000 rows. The right side of Figure 6.1 demonstrates the ability of *TunableShrinkage* to handle the high dimension. We fixed the row value at 600 and used increasing amount of regulation achieving very accurate results.

6.2 A Block of Outliers

Here A has the following structure

$$\begin{aligned} A &= \begin{bmatrix} \text{Data block 1} & \text{Outlier block} & \text{Data block 2} \end{bmatrix} \\ &= B_{50\times 50} \begin{bmatrix} M_{1,(50\times 10000)} & M_{2,(50\times 200...800)} & M_{3,(50\times 10000)} \end{bmatrix} \end{aligned}$$

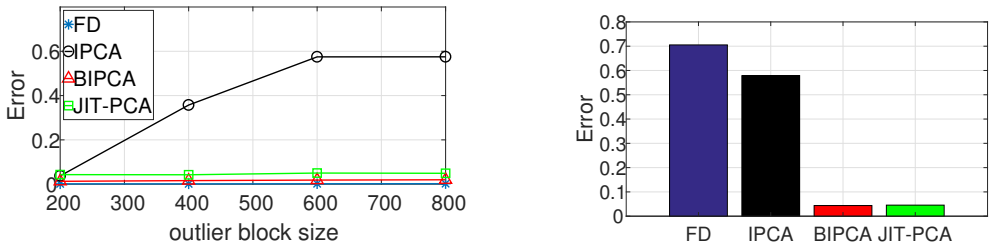

Figure 6.2: Block of Outliers: on the left, E_{recon} as the outlier block increase in size. The dimension of data is 50 and the dominant rank is 6. The Basic IPCA fail completely, BIPCA and JIT-PCA perform equally well but *FrequentDirections* is order of magnitude better then the rest. On the right, outlier block is fixed on 800 and data dimension is increased to 350. Now *FrequentDirections* error is even bigger then the Basic IPCA

Rows $1-3$ of M_1 and rows $10-12$ of M_3 have entries drawn from $\sim N(0,1)$. Rows $4-9$ of M_2 have entries drawn from $\sim N(0,3)$. The remaining entries are drawn from $\sim N(0,0.1)$. The average norm of vectors from the data blocks is lower then the average norm of vectors from the outlier block but the size of the outlier block (number of vectors) is much smaller. Each data block true rank ($\bar{k}$) is 3, the outlier block rank is 6 and the approximation rank (k) used was 10 (trying to approximate subspace spanned by the 6 dominant basis vectors). This test demonstrates the effects of the *weight of experience.* In the left side of figure 6.2 the data dimension (number of rows) is 50 and the outlier block size changes from 200 to 800. As the outlier block size increases so does its associated singular values (but they remain smaller then those associated with the data blocks). Here *FrequentDirections* performed best with $E_{\text{recon}} \approx 0.001$, *BIPCA* error was around 0.01 and *JIT-PCA* error was around 0.04. The results of all of them is steady and not effected by the outlier block size increase. As expected, the *Basic IPCA* failed completely when the block size got larger. The right side uses only one block size, 800 but with a higher data dimension, 350. Now, due to the high dimension of the noise, *FrequentDirections* results is even worse then the *Basic IPCA*. Both *BIPCA* and *JIT-PCA* results remained roughly the same.

6.3 BIPCA and JIT-PCA behavior

Here we take a closer look when and how BIPCA and JIT-PCA uses the orthogonal part of the new vector. A has the following structure

$$A = B_{50\times 50}\begin{bmatrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{bmatrix}$$

with each $M_i \in \mathbb{R}^{50\times 1000}$. Rows 1-5 of blocks (M_1, M_4), Rows 6-10 of blocks (M_2, M_5) and Rows 11-15 of blocks (M_3, M_6) have entries drawn from $\sim N(0,1)$. The approximation rank used is 20. We generated the dataset twice with different noise value $\sigma_n = 0.01$ and $\sigma_n = 0.1$. The left side of figure 6.3 presents the singular values when $\sigma_n = 0.01$ and the right side presents the singular values of the test dataset with a higher noise level $\sigma_n = 0.1$. In each we can see a group of 15 dominant singular values (5 belonging to each basis that generated a pair of blocks), a gap which is determined by the σ_d/σ_n ratio, and the rest of the singular values. Figure 6.4 presents the boost probability for *BIPCA*. We can see two picks right after column index 1000 and 2000, those are the locations where a new part of the generation basis was first witnessed. Note also that the next basis switch column indexes (3001, 4001,5001) no longer have this pick, the basis was already incorporated and kept. Figure 6.5 presents the probability of using the orthogonal part + boost for *JIT-PCA*. The boost probability with *BIPCA* provides feedback regarding the approximation accuracy and the same goes for *JIT-PCA* for which the probability is also indicative for its complexity *(BIPCA* complexity in not effected by the noise level*).* The lower the probability the less the algorithm uses the orthogonal part and its operation count approaches an average cost of *2mk* per step. On the left side, $\sigma_n = 0.01$ and the $O(mk)$ coefficient is 2.03, on the right side, $\sigma_n = 0.1$ and the $O(mk)$ coefficient is 2.45. Even in such high noise settings *JIT-PCA* cost is only 25% more expensive then projecting a vector into a subspace

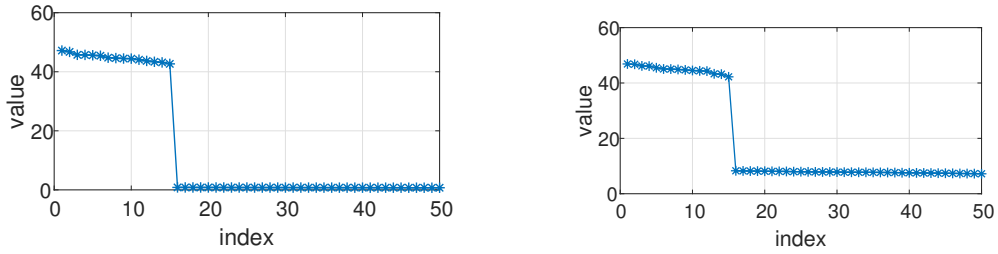

Figure 6.3: Singular Values For *BIPCA* and *JIT-PCA* Analysis: on the left $\sigma_n = 0.01$ and on the right, with a smaller gap between strong and week singular values, $\sigma_n = 0.1$

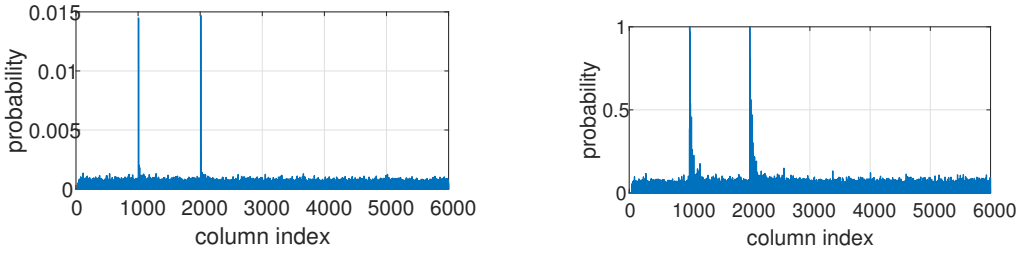

Figure 6.4: *BIPCA* Boost Probability: on the left $\sigma_n = 0.01$ and on the right $\sigma_n = 0.1$. The boost probability in low noise is very low as the basis is captured accurately. The $O(mk)$ coefficient for both is 5.45, independent on the noise level

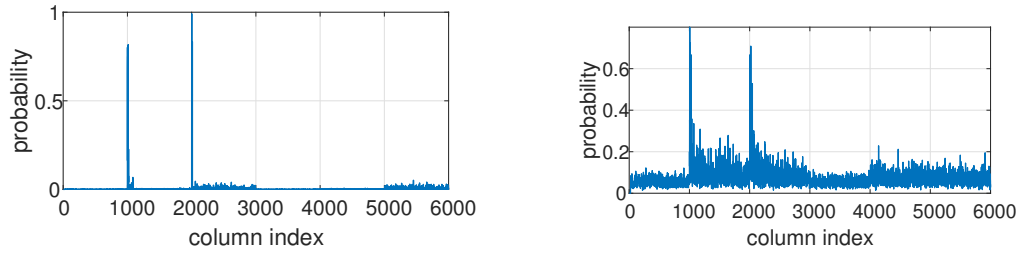

Figure 6.5: *JIT-PCA* Full Update Probability: on the left $\sigma_n = 0.01$ and the $O(mk)$ coefficient is 2.03, on the right $\sigma_n = 0.1$ and the $O(mk)$ coefficient is 2.45. Even in such high noise setting *JIT-PCA* cost is only 25% more expensive then projecting a vector into a subspace

7 Experimental Results with Real World Datasets

7.1 BIRDS Dataset

Inspired by Liberty [12],we consider the real-world dataset BIRDS [1] in which each column represents an image of a bird, and each row a feature, represented via a binary parameter. This dataset has 11788 columns (each an image) of dimension 312. The left side of figure 7.1 shows the singular values of this dataset. There is no real gap distinguishing between the dominant and noise subspace. In our tests we chose the 20th singular value as the cutoff point. It is labeled on the figure by a green circle. Figure 7.2 presents E_{recon} as a function of the approximation rank with values of 25,30,35,40,45. We can see that the *Basic IPCA* performs best (except for rank 25 where *BIPCA* and *JIT-PCA* were slightly better) with *BIPCA* and *JIT-PCA* on the same level. *FrequentDirections*, (again due to high dimension noise) error rate is much higher. The right side of figure 7.1 demonstrates the approximation of *TunableShrinkage*. We can see that with a shrinkage ratio of 100 we get best results, twice as good as the Basic IPCA.

7.2 pcStream

Mirsky et al. [18], interested in Context Mining from streams, suggested an algorithm called *pcStream.* The algorithm maintains models that represents different user states (one model per state), for example, the stream can be generated by the user smartphone accelerometer and the states are running, walking or jumping. Briefly, *pcStream* operates by modeling situation spaces as Gaussian distributions

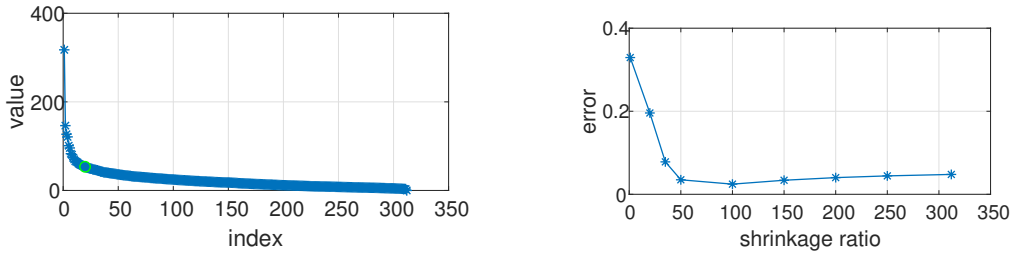

Figure 7.1: BIRDS Dataset: On the left, its singular values, with the 20th labeled in green circle. On the right, E_{recon} for *TunableShrinkage* as a function of the the shrinkage ratio. the edges are *FrequentDirections* and *Basic IPCA*, the best result for tested values is at the value 100

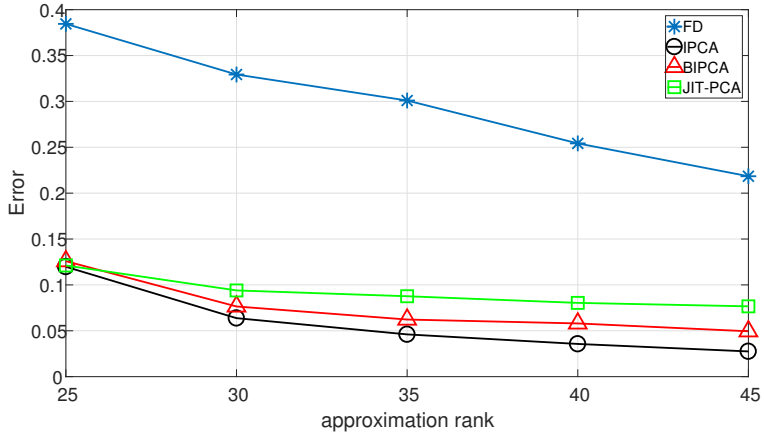

Figure 7.2: BIRDS Dataset: E_{recon} for *FrequentDirections*, *Basic IPCA*, *BIPCA* and *JIT-PCA* as a function of the approximation rank. *Frequent-Directions* error is much higher then the rest due to high noise dimension

while tracking the flow of the stream between known models. When an instance arrives from the stream, it is assigned to the closest model using Mahalanobis distance. Should there be no model close enough then the instance is added to the a buffer B. The purpose of buffer B is to capture the instances that potentially belong to a new situation space. *pcStream* assumes that the actor behind the stream remains in each situation for at least $t_{\min}$ observations at a time. If an observation is ever assigned to an existing model, B is immediately emptied and used to update that model. However, should B ever reach capacity, a new situation space is modeled after the contents of B, and B is reset. For each model / cluster, pcStream maintains a buffer C_i with the last n observations that were assign to this cluster and in addition a rank$-k$ PCA representation for this buffer. Each time a new instance is assigned to a cluster, the oldest instance in C_i is discarded, the new instance is placed as the newest and a rank k PCA is performed of this updated C_i.

On a follow up paper [17], we teamed with Mirsky et al. suggesting several improvements for *pcStream*. Our contribution included the incorporation of *JIT-PCA* into *pcStream,* allowing a much faster run time, much better memory usage (as there was no longer any need to maintain C_i) with results equally good. We encourage interested readers to review this related paper.

References

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

[2] Christopher G Baker. A block incremental algorithm for computing dominant singular subspaces. 2004.

[3] Christopher G Baker, Kyle A Gallivan, and Paul Van Dooren. Low-rank incremental methods for computing dominant singular subspaces. *Linear Algebra and its Applications*, 436(8):2866–2888, 2012.

[4] Michael W Berry, Susan T Dumais, and Todd A Letsche. Computational methods for intelligent information access. In *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, pages 20–20. IEEE, 1995.

[5] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *ECCV (1)*, volume 2350 of *Lecture Notes in Computer Science*, pages 707–720. Springer, 2002.

[6] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and Its Applications*, 415(1):20–30, 2006.

[7] Y. Chahlaoui, Kyle A. Gallivan, and Paul Van Dooren. Recursive calculation of dominant singular subspaces. *SIAM J. Matrix Analysis Applications*, 25(2):445–463, 2003.

[8] Younes Chahlaoui, Kyle A. Gallivan, and Paul Van Dooren. An incremental method for computing dominant singular spaces. In *In Computational Information Retrieval*, pages 53–62, 2001.

[9] Shivkumar Chandrasekaran, BS Manjunath, Yuan-Fang Wang, Jay Winkeler, and Henry Zhang. An eigenspace update algorithm for image analysis. *Graphical Models and Image Processing*, 59(5):321–332, 1997.

[10] Amey Desai, Mina Ghashami, and Jeff M Phillips. Improved practical matrix sketching with guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1678–1690, 2016.

[11] Alan M. Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *FOCS*, pages 370–378. IEEE Computer Society, 1998.

[12] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

[13] Ming Gu and Stanley C Eisenstat. A stable and fast algorithm for updating the singular value decomposition, 1994.

[14] A Levey and Michael Lindenbaum. Sequential karhunen-loeve basis extraction and its application to images. *IEEE Transactions on Image processing*, 9(8):1371–1374, 2000.

[15] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[16] BS Manjunath, Shivkumar Chandrasekaran, and Yuan-Fang Wang. An eigenspace update algorithm for image analysis. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 551–556. IEEE, 1995.

[17] Yisroel Mirsky, Tal Halpern, Rishabh Upadhyay, Sivan Toledo, and Yuval Elovici. Enhanced situation space mining for data streams. In *Proceedings of the Symposium on Applied Computing*, pages 842–849. ACM, 2017.

[18] Yisroel Mirsky, Bracha Shapira, Lior Rokach, and Yuval Elovici. pcstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 119–133. Springer, 2015.

[19] Gavin W O'Brien. Information management tools for updating an svd-encoded indexing scheme. Master's thesis, University of Tennessee, Knoxville, 1994.

[20] Hongyuan Zha and Horst D Simon. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing*, 21(2):782–791, 1999.

A *TunableShrinkage Error Bounds*

We now provide the proof to theorem 9. The provided proof is an adaptation to the proofs provided by Liberty [12]. In order to make the comparison easier, we will work row-wise, same as in the original article. First, let us review *Frequent-directions* again. B_l indicates row l of B and a_i is row i of A.

1. Inputs: l, A$\in \mathbb{R}^{n\times d}$
2. $B \leftarrow 0^{l\times d}$
3. For $i = 1 : n$
   (a) $B_l \leftarrow a_i$
   (b) $\begin{bmatrix} U & \Sigma & V \end{bmatrix} \leftarrow \text{SVD}(B)$
   (c) $C \leftarrow \Sigma V^T$
   (d) $\delta \leftarrow \sigma_l^2$
   (e) $B \leftarrow \sqrt{\Sigma^2 - \delta I_l} \cdot V^T$
4. Return B

TunableShrinkage works the same as *Frequentdirections* with three differences. First we also get as input a number $r \geq 1$. Second, we use $\delta \leftarrow \sigma_l^2/r$ and third, we calculate $\sqrt{\Sigma^2 - \delta I_l}$, we then zero out $\Sigma_{l,l}$ and only then reconstruct B. In the proofs ahead, we denote $\delta_i = \sigma_l^2/r$, B_i, C_i the values of δ, B, and C respectively after the i'th row of A was processed, $\Delta = \sum_{i=1}^{n} \delta_i$, B_0 an all zeros matrix and $B_n = B$ the final sketch. In addition let $\widetilde{B}_i$ denote the first $l-1$ rows of B_i. We

denote by y_j the right singular vectors (in descending order) of A and v_j of B and B_i (we drop the i notation from v for simplicity, it will be clear from the context to which it refers) . Finally, we denote $\pi_B^k(A) = AV_kV_k^T$ the right projection of A on the top k right singular vectors of B and A_k to be A's best rank k approximation.

Claim 2. For any vector x we have $\langle a_i, x\rangle^2 + \|B_{i-1}x\|^2 = \|C_i x\|^2$

Proof. At each iteration we start by placing the new vector into the (all zeros) last row of B_{i-1} and we then perform SVD on the augmented matrix

$$U_i\Sigma_i V_i^T = \begin{bmatrix} \widetilde{B}_{i-1} \\ a_i \end{bmatrix} \Rightarrow \|U_i\Sigma_i V_i^T x\|^2 = \left\| \begin{bmatrix} \widetilde{B}_{i-1} \\ a_i \end{bmatrix} x \right\|^2$$

The L.H.S is

$$\|U_i\Sigma_i V_i^T x\|^2 = \|\Sigma_i V_i^T x\|^2 = \|C_i x\|^2$$

And the R.H.S

$$\left\| \begin{bmatrix} \widetilde{B}_{i-1} \\ a_i \end{bmatrix} x \right\|^2 = \langle a_i, x\rangle^2 + \|B_{i-1}x\|^2$$

□

Claim 3. For any vector x we have $\|C_i x\|^2 - \|B_i x\|^2 \geq 0$

Proof.

$$\|C_i x\|^2 = \|\Sigma_i V_i^T x\|^2 = \sum_{j=1}^{l} \sigma_j^2 \langle v_j, x\rangle^2 \geq$$
$$\sum_{j=1}^{l} \left[(\sigma_j^2 - \delta_i) \langle v_j, x\rangle^2 \right] \geq \sum_{j=1}^{l-1} \left[(\sigma_j^2 - \delta_i) \langle v_j, x\rangle^2 \right] = \|B_i x\|^2$$

□

Lemma 1. *(Property 1) For any vector x, we have* $\|Ax\|^2 - \|Bx\|^2 \geq 0$

Proof.

$$\|Ax\|^2 - \|Bx\|^2 = \sum_{i=1}^{n} \left[\langle a_i, x \rangle^2 + \|B_{i-1}x\|^2 - \|B_i x\|^2 \right] \overset{\text{claim 2}}{=} \sum_{i=1}^{n} \left[\|C_i x\|^2 - \|B_i x\|^2 \right] \overset{\text{claim 3}}{\geq} 0$$

□

Claim 4. For any unit vector x we have $\|C_i x\|^2 - \|B_i x\|^2 \leq r\delta_i$

Proof.

$$\begin{aligned} \|C_i x\|^2 - \|B_i x\|^2 &= \sum_{j=1}^{l} \sigma_j^2 \langle v_j, x \rangle^2 - \sum_{j=1}^{l-1} \left[\left(\sigma_j^2 - \delta_i \right) \langle v_j, x \rangle^2 \right] = \sigma_l^2 \langle v_l, x \rangle^2 + \delta_i \sum_{j=1}^{l-1} \langle v_j, x \rangle^2 \\ &= r\delta_i \langle v_l, x \rangle^2 + \delta_i \sum_{j=1}^{l-1} \langle v_j, x \rangle^2 = (r-1)\, \delta_i \langle v_l, x \rangle^2 + \delta_i \|V^T x\|^2 \\ &\leq (r-1)\, \delta_i \langle v_l, x \rangle^2 + \delta_i \|V^T\|^2 \leq (r-1)\, \delta_i + \delta_i = r\delta_i \end{aligned}$$

□

Lemma 2. *(property 2) For any unit vector x we have* $\|Ax\|^2 - \|Bx\|^2 \leq r\Delta$

Proof.

$$\|Ax\|^2 - \|Bx\|^2 = \sum_{i=1}^{n} \left[\|C_i x\|^2 - \|B_i x\|^2 \right] \overset{\text{claim 4}}{\leq} \sum_{i=1}^{n} r\delta_i = r\Delta$$

□

Claim 5. $\|C_i\|_F^2 \geq \|B_i\|_F^2 + l\delta_i$

Proof.

$$\|C_i\|_F^2 - \|B_i\|_F^2 = \sum_{j=1}^{l} \sigma_j^2 - \sum_{j=1}^{l-1} (\sigma_j^2 - \delta_i) = \sigma_l^2 + (l-1)\, \delta_i = (l + r - 1)\, \delta_i \geq l\delta_i$$

□

Claim 6. $\|C_i\|_F^2 = \|B_{i-1}\|_F^2 + \|a_i\|^2$

Proof.

$$\|C_i\|_F^2 = \left\| \begin{bmatrix} \widetilde{B}_{i-1} \\ a_i \end{bmatrix} \right\|_F^2 = \|B_{i-1}\|_F^2 + \|a_i\|^2$$

□

Lemma 3. *(property 3)* $\Delta l \leq \|A\|_F^2 - \|B\|_F^2$

Proof.

$$\|A\|_F^2 = \sum_{i=1}^{n} \|a_i\|^2 \overset{\text{claim 6}}{=} \sum_{i=1}^{n} \|C_i\|_F^2 - \|B_{i-1}\|_F^2 \overset{\text{claim 5}}{\geq} \sum_{i=1}^{n} \left[\|B_i\|_F^2 + l\delta_i - \|B_{i-1}\|_F^2\right] = \|B\|_F^2 + \Delta l$$

□

Lemma 4. $\Delta \leq \frac{\|A-A_k\|_F^2}{l-kr}$

Proof.

$$\begin{aligned}
\Delta l &\overset{\text{lemma 3}}{\leq} \|A\|_F^2 - \|B\|_F^2 \\
&= \sum_{i=1}^{k} \|Ay_i\|_F^2 + \sum_{i=k+1}^{d} \|Ay_i\|_F^2 - \|B\|_F^2 \\
&= \sum_{i=1}^{k} \|Ay_i\|^2 + \|A-A_k\|_F^2 - \|B\|_F^2 \\
\text{with} \quad \|B\|_F^2 &\geq \textstyle\sum_{i=1}^{k} \|By_i\|^2 \\
&\leq \|A-A_k\|_F^2 + \sum_{i=1}^{k} \left[\|Ay_i\|^2 - \|By_i\|^2\right] \\
&\overset{\text{lemma 2}}{\leq} \|A-A_k\|_F^2 + kr\Delta
\end{aligned}$$

Now solving for Δ

$$\begin{aligned}
\Delta l &\leq \|A-A_k\|_F^2 + kr\Delta \\
\Delta(l-rk) &\leq \|A-A_k\|_F^2 \\
\Delta &\leq \frac{\|A-A_k\|_F^2}{l-kr}
\end{aligned}$$

□

Now we can prove the theorem provided in the text body: Let $B \in \mathbb{R}^{l \times d}$ be the sketch produced by TunableShrinkage, for any $k < l/r$ and $r \geq 1$ it holds that

Theorem 10.

$$\|A - \pi_B^k(A)\|_F^2 \leq \left(1 + \frac{kr}{l-kr}\right) \|A - A_k\|_F^2$$

Proof.

$$\begin{aligned}
\|A - \pi_B^k(A)\|_F^2 &\overset{\text{Pythgorean Th.}}{=} \|A\|_F^2 - \|\pi_B^k(A)\|_F^2 = \|A\|_F^2 - \sum_{i=1}^{k} \|Av_i\|^2 \\
&\overset{\text{lemma 1}}{\leq} \|A\|_F^2 - \sum_{i=1}^{k} \|Bv_i\|^2 \\
&\leq \|A\|_F^2 - \sum_{i=1}^{k} \|By_i\|^2 \\
&\overset{\text{lemma 2}}{\leq} \|A\|_F^2 - \sum_{i=1}^{k} \left[\|Ay_i\|^2 - r\Delta\right] \\
&= \|A\|_F^2 - \|A_k\|_F^2 + kr\Delta \\
&\overset{\text{lemma 4}}{\leq} \|A - A_k\|_F^2 + \frac{kr\|A - A_k\|_F^2}{l-kr} \\
&= \left(1 + \frac{kr}{l-kr}\right) \|A - A_k\|_F^2
\end{aligned}$$

□

תקציר

במסגרת מחקר זה פותחו אלגוריתמים חדשים לביצוע PCA על מאגרי מידע גדולים ובמתארי הזרמת מידע. אנו מציגים מסגרת אחודה לאלגוריתמים המבצעים PCA מתעדכן ומראים כי חלק נרחב מהאלגוריתמים הקיימים משתייך למסגרת זו. אנו מציגים קרטריון שגיאה חדש המתאים למידע ממימד גבוה ומראים כי תחת קריטריון זה, מגוון רחב של אלגוריתמים מקובלים עלולים להכשל, אפילו עבור מאגרי מידע פשוטים. אנו מציגים אלגוריתמים שפותחו על מנת להיות חסינים יותר לכשל ומציגים שיטה המאפשרת מימוש מהיר הן של האלגוריתמים החדשים והן של האלגוריתמים המוכרים.

אוניברסיטת תל-אביב

הפקולטה למדעים מדויקים

ע״ש ריימונד ובברלי סאקלר

בית הספר למדעי המתמטיקה

אלגוריתמי PCA מהירים וחסינים עבור מאגרי מידע גדולים והזרמת מידע

חיבור זה הוגש כעבודת מחקר לקראת התואר "מוסמך אוניברסיטה" במתמטיקה שימושית

על ידי

טל הלפרן

העבודה נעשתה בבית הספר למדעי המתמטיקה

בהנחיית פרופ' סיון טולדו ופרופ' יואל שקולניצקי

אייר תשע"ז

אוניברסיטת תל-אביב

הפקולטה למדעים מדויקים

ע״ש ריימונד ובברלי סאקלר

בית הספר למדעי המתמטיקה

אלגוריתמי PCA מהירים וחסינים עבור מאגרי מידע גדולים והזרמת מידע

חיבור זה הוגש כעבודת מחקר לקראת התואר "מוסמך אוניברסיטה" במתמטיקה שימושית

על ידי

טל הלפרן

אייר תשע"ז