Sparse PCA through Low-rank Approximations

Dimitris S. Papailiopoulos DIMITRIS@UTEXAS.EDU
Alexandros G. Dimakis DIMAKIS@AUSTIN.UTEXAS.EDU
Department of Electrical and Computer Engineering, the University of Texas at Austin, TX, USA

Stavros Korokythakis STAVROS@STOCHASTICTECHNOLOGIES.COM
Stochastic Technologies

Abstract

We introduce a novel algorithm that computes the k-sparse principal component of a positive semidefinite matrix A. Our algorithm is combinatorial and operates by examining a discrete set of special vectors lying in a low-dimensional eigen-subspace of A. We obtain provable approximation guarantees that depend on the spectral profile of the matrix: the faster the eigenvalue decay, the better the quality of our approximation. For example, if the eigenvalues of A follow a power-law decay, we obtain a polynomial-time approximation algorithm for any desired accuracy. We implement our algorithm and test it on multiple artificial and real data sets. Due to a feature elimination step, it is possible to perform sparse PCA on data sets consisting of millions of entries in a few minutes. Our experimental evaluation shows that our scheme is nearly optimal while finding very sparse vectors. We compare to the prior state of the art and show that our scheme matches or outperforms previous algorithms in all tested data sets.

1. Introduction

Principal component analysis (PCA) reduces the dimensionality of a data set by projecting it onto principal subspaces spanned by the leading eigenvectors of the sample covariance matrix. The statistical significance of PCA partially lies in the fact that the principal components capture the largest possible data variance. The first principal component (i.e., the first eigenvector) of an $n \times n$ matrix A is the solution to

$$\arg\max_{\|x\|_2=1} x^T A x$$

where $A = SS^T$ and S is the $n \times m$ data set matrix consisting of m data-points, or entries, each evaluated on n features, and $\|x\|_2$ is the ℓ_2-norm of x. PCA can be efficiently computed using the singular value decomposition (SVD). The statistical properties and computational tractability of PCA renders it one of the most used tools in data analysis and clustering applications.

A drawback of PCA is that the generated vectors typically have very few zero entries, i.e., they are not *sparse*. Sparsity is desirable when we aim for *interpretability* in the analysis of principal components. An example where sparsity implies interpretability is document analysis, where principal components can be used to cluster documents and detect trends. When the principal components are sparse, they can be easily mapped to topics (e.g., newspaper article classification into politics, sports, etc.) using the few keywords in their support (Gawalt et al., 2010; Zhang & El Ghaoui, 2011). For that reason it is desirable to find sparse eigenvectors.

Sparse PCA. Sparsity can be directly enforced in the principal components. The sparse principal component x_* is defined as

$$x_* = \arg\max_{\|x\|_2=1, \|x\|_0=k} x^T A x. \qquad (1)$$

The ℓ_0 cardinality constraint limits the optimization over vectors with k non-zero entries. As expected, sparsity comes at a cost since the optimization in (1) is NP-hard (Moghaddam et al., 2006a) and hence computationally intractable in general.

Our Contribution. We introduce a novel algorithm for sparse PCA that has a provable approximation guarantee. Our algorithm generates a k-sparse,

Proceedings of the 30

unit length vector x_d that gives an objective provably within a $1-\epsilon_d$ factor from the optimal:

$$x_d^T A x_d \geq (1-\epsilon_d)\, x_*^T A x_*$$

with

$$\epsilon_d \leq \min\left\{\frac{n}{k}\cdot\frac{\lambda_{d+1}}{\lambda_1},\ \frac{\lambda_{d+1}}{\lambda_1^{(1)}}\right\}, \tag{2}$$

where λ_i is the ith largest eigenvalue of A and $\lambda_1^{(1)}$ is the maximum diagonal element of A. For any desired value of the parameter d, our algorithm runs in time $O(n^{d+1}\log n)$. Our approximation guarantee is directly related to the spectrum of A: the greater the eigenvalue decay, the better the approximation. Equation (2) contains two bounds: one that uses the largest eigenvalue λ_1 and one that uses the largest diagonal element of A, $\lambda_1^{(1)}$. Either bound can be tighter, depending on the structure of the A matrix.

We subsequently rely on our approximation result to establish guarantees for considerably general families of matrices.

Constant-factor approximation. If we only assume that there is an arbitrary decay in the eigenvalues of A, *i.e.*, there exists a constant $d = O(1)$ such that $\lambda_1 > \lambda_{d+1}$, then we can obtain a constant-factor approximation guarantee for the linear sparsity regime. Specifically, we find a constant δ_0 such that for all sparsity levels $k > \delta_0\, n$ we obtain a constant approximation ratio for sparse PCA, partially solving the open problem discussed in (Zhang et al., 2012; d'Aspremont et al., 2012). This result easily follows from our main theorem.

Eigenvalue Power-law Decay. When the data matrix spectrum exhibits a power-law decay, we can obtain a much stronger performance guarantee: we can solve sparse PCA for any desired accuracy ϵ in time polynomial in n, k (but not in $\frac{1}{\epsilon}$). This is sometimes called a polynomial-time approximation scheme (PTAS). Further, the power-law decay is not necessary: the spectrum does not have to follow exactly that decay, but only exhibit a substantial spectral drop after a few eigenvalues.

Our algorithm operates by scanning a low-dimensional subspace of A. There, it examines a polynomial number of special vectors, that lead to a sparse principal component which admits provable performance. A key conceptual innovation that we employ is a hyperspherical transformation on our problem space to reduce its dimensionality. Another important component of our scheme is a safe feature elimination step that allows the scalability of our algorithm for data sets with millions of entries. We introduce a test that discards features that are provably not in the support of the sparse PC, in a similar manner as (Zhang & El Ghaoui, 2011), but using a different combinatorial criterion.

Experimental Evaluation. We evaluate and compare our algorithm against state of the art sparse PCA approaches on synthetic and real data sets. Our real data set is a large Twitter collection of more than 10 million tweets spanning approximately six months. We executed several experiments on various subsets of our data set: collections of tweets during a specific time-window, tweets that contained a specific word, etc. Our implementation executes in less than one second for $50k - 100k$ documents and in a few minutes for millions of documents. Our scheme typically comes closer than 90% of the optimal performance, even for $d \leq 3$, and empirically outperforms previously proposed sparse PCA algorithms.

1.1. Related Work

There has been a substantial volume of prior work on sparse PCA. Initial heuristic approaches used factor rotation techniques and thresholding of eigenvectors to obtain sparsity (Kaiser, 1958; Jolliffe, 1995; Cadima & Jolliffe, 1995). Then, a modified PCA technique based on the LASSO (SCoTLASS) was introduced in (Jolliffe et al., 2003). In (Zou et al., 2006), a nonconvex regression-type approximation, penalized à la LASSO was used to produce sparse PCs. A nonconvex technique was presented in (Sriperumbudur et al., 2007). In (Moghaddam et al., 2006b), the authors used spectral arguments to motivate a greedy branch-and-bound approach, further explored in (Moghaddam et al., 2007). In (Shen & Huang, 2008), a similar technique to SVD was used employing sparsity penalties on each round of projections. A significant body of work based on semidefinite programming (SDP) approaches was established in (d'Aspremont et al., 2007a; Zhang et al., 2012; d'Aspremont et al., 2008). A variation of the power method was used in (Journée et al., 2010). When computing multiple PCs, the issue of deflation arises as discussed in (Mackey, 2009). In (Amini & Wainwright, 2008), the first theoretical optimality guarantees were established for thresholding and the SDP relaxation of (d'Aspremont et al., 2007a), in the high-dimensional setting of a generative model where the covariance has one sparse eigenvector. In (Yuan & Zhang, 2011), the authors introduced a very efficient sparse PCA approximation based on truncating the well-known power method to obtain the exact level of sparsity desired, which came along with peformance guarantees for a specific data model. In (Asteris et al., 2011), the authors present an algorithm that solves sparse PCA exactly and in polynomial time for matrices of constant rank. The main algorithmic differences from (Asteris et al., 2011) are *i)* our solver speeds up calculations for matrices with nonnegative entries by a 2^{d-1} factor in running complexity and *ii)* a safe feature elimination step is introduced that is fundamental in implementing the algorithm for large data sets. Despite this extensive literature, to the best of our knowledge, there are very few provable approximation guarantees for sparse PCA algorithms and usually under limited data models (Amini & Wainwright, 2008; Yuan & Zhang, 2011; d'Aspremont et al., 2012; Ma, 2011).

2. Sparse PCA through Low-rank Approximations

2.1. Proposed Algorithm

Our algorithm is technically involved and for that reason we start with a high-level informal description. For any given accuracy parameter d we follow the following steps:

Step 1: *Obtain A_d, a rank-d approximation of A.* We obtain A_d, the best-fit rank-d approximation of A, by keeping the first d terms in its eigen-decomposition:

$$A_d = \sum_{i=1}^{d} \lambda_i v_i v_i^T,$$

where λ_i is the i-th largest eigenvalue of A and v_i the corresponding eigenvector.

Step 2: *Use A_d to obtain $O(n^d)$ candidate supports.* For any matrix A, we can exhaustively search for the optimal x_* by checking all $\binom{n}{k}$ possible $k \times k$ submatrices of A: x_* is the k-sparse vector with the same support as the sub-matrix of A with the maximum largest eigenvalue. However, we show how sparse PCA can be efficiently solved on A_d if the rank d is constant with respect to n. The key technical fact that we prove is that there are *only* $O(n^d)$ candidate supports that need to be examined. Specifically, we show that a *set of candidate supports* $\mathcal{S}_d = \{\mathcal{I}_1, \ldots, \mathcal{I}_T\}$, where $\mathcal{I}_t$ is a subset of k indices from $\{1, \ldots, n\}$, contains the optimal support. We prove that the number of these supports is[1]

$$|\mathcal{S}_d| \leq 2^{2d} \binom{n}{d}.$$

The above set $\mathcal{S}_d$ is efficiently created by our *Spannogram algorithm* described in the next subsection.

Step 3: *Check each candidate support from $\mathcal{S}_d$ on A.* For a given support $\mathcal{I}$ it is easy to find the best vector supported on $\mathcal{I}$: it is the leading eigenvector of the principal sub-matrix of A, with rows and columns indexed by $\mathcal{I}$. In this step, we check all the supports in $\mathcal{S}_d$ *on the original matrix A* and output the best. Specifically, define $A_\mathcal{I}$ to be the zeroed-out version of A, except on the support $\mathcal{I}$. That is, $A_\mathcal{I}$ is an $n \times n$ matrix with zeros everywhere except for the principal sub-matrix indexed by $\mathcal{I}$. If $i \in \mathcal{I}$ and $j \in \mathcal{I}$, then $A_\mathcal{I} = A_{ij}$, else it is 0. Then, for any $A_\mathcal{I}$ matrix, with $\mathcal{I} \in \mathcal{S}_d$, we compute its largest eigenvalue and corresponding eigenvector.

Algorithm 1 Sparse PCA via a rank-d approximation

```
Input: $k$, $d$, $A$
$p \leftarrow 1$ if $A$ has nonnegative entries, 0 if mixed
$A_d \leftarrow \sum_{i=1}^{d} \lambda_i v_i v_i^T$
$\hat{A}_d \leftarrow$ feature_elimination$(A_d)$
$\mathcal{S}_d \leftarrow$ Spannogram$\left(k, p, \hat{A}_d\right)$
for each $\mathcal{I} \in \mathcal{S}_d$ do
    Calculate $\lambda_1(A_\mathcal{I})$
end for
$\mathcal{I}_d^{\text{opt}} = \arg\max_{I \in \mathcal{S}_d} \lambda_1(A_\mathcal{I})$
$\text{OPT}_d = \lambda_1(A_{\mathcal{I}_d^{\text{opt}}})$
$x_d^{\text{opt}} \leftarrow$ the principal eigenvector of $A_{\mathcal{I}_d^{\text{opt}}}$.
Output: $x_d^{\text{opt}}$
```

Output: Finally, we output the k-sparse vector x_d that is the principal eigenvector of the $A_\mathcal{I}$ matrix, $\mathcal{I} \in \mathcal{S}_d$, with the largest maximum eigenvalue. We refer to this approximate sparse PC solution as the *rank-d optimal solution.*

The exact steps of our algorithm are given in the pseudo-code tables denoted as Algorithm 1 and 2. The spannogram subroutine, i.e., Algorithm 2, computes the T candidate supports in $\mathcal{S}_d$, and is presented and explained in Section 3. The complexity of our algorithm is equal to calculating d leading eigenvectors of A ($\mathcal{O}(dn^2)$), running our spannogram algorithm ($\mathcal{O}(n^{d+1} \log n)$), and finding the leading eigenvector of $O(n^d)$ matrices of size $k \times k$ ($\mathcal{O}(n^d k^2)$). Hence, the total complexity is $O(n^{d+1} \log n + n^d k^2 + dn^2)$.

Elimination Step: By using a feature elimination subroutine we can identify that certain variables provably cannot be in the support of x_d, the rank-d optimal sparse PC. We have a test which is related to the norms of the rows of V_d that identifies which of the n rows cannot be in the optimal support. We use this step to further reduce the number of candidate supports $|\mathcal{S}_d|$. The elimination

[1] In fact, in our proof we show a better dependency on d, which however has a more complicated expression.

algorithm is very important when it comes to large-scale data sets. For example, for some of our Twitter experiments, the elimination was reducing n from $100,000$ down to only 100, or fewer candidate features. This subroutine is presented in detail in the extended version of the manuscript (Papailiopoulos et al., 2013).

2.2. Approximation Guarantees

The desired sparse PC is $x_* = \underset{\|x\|_2=1, \|x\|_0=k}{\arg\max} \; x^T A x$. We instead obtain the k-sparse, unit length vector x_d which gives an objective

$$x_d^T A x_d = \max_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_{\mathcal{I}}).$$

We measure the quality of our approximation using the standard approximation factor:

$$\rho_d = \frac{x_d^T A x_d}{x_*^T A x_*} = \frac{\max\limits_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_{\mathcal{I}})}{\lambda_1^{(k)}},$$

where $\lambda_1^{(k)} = x_*^T A x_*$ is the k-sparse largest eigenvalue of A.[2] Clearly, $\rho_d \leq 1$ and as it approaches 1, the approximation becomes tighter. Our main result follows:

Theorem 1. *For any d, our algorithm outputs x_d, where $||x_d||_0 = k$, $||x_d||_2 = 1$ and*

$$x_d^T A x_d \geq (1 - \epsilon)\, x_*^T A x_*,$$

with an error bound $\epsilon_d \leq \min\left\{ \frac{n}{k} \frac{\lambda_{d+1}}{\lambda_1}, \frac{\lambda_{d+1}}{\lambda_1^{(1)}} \right\}$.

Proof. The proof can be found in (Papailiopoulos et al., 2013). The main idea is that we obtain *i)* an upper bound on the performance loss using A_d instead of A and *ii)* a lower bound for $\lambda_1^{(k)}$. □

We now use our main theorem to provide the following model specific approximation results.

Corollary 1. *Assume that for some constant value d, there is an eigenvalue decay $\lambda_1 > \lambda_{d+1}$ in A. Then there exists a constant δ_0 such that for all sparsity levels $k > \delta_0 n$ we obtain a constant approximation ratio.*

Corollary 2. *Assume that the first $d + 1$ eigenvalues of A follow a power-law decay, i.e., $\lambda_i = Ci^{-\alpha}$, for some $C, \alpha > 0$. Then, for any $k = \delta n$ and any $\epsilon > 0$ we can get a $(1 - \epsilon)$-approximate solution x_d in time $O\left(n^{1/(\epsilon\delta)^\alpha + 1} \log n\right)$.*

The above corollaries can be established by plugging in the values for λ_i in the error bound. We find the above families of matrices interesting, because in practical data sets (like the ones we tested), we observe a significant decay in the first eigenvalues of A which in many cases follows a power law. The main point of the above approximability result is that any matrix with decent decay in the spectrum endows a good sparse PCA approximation.

3. The Spannogram Algorithm

In this section, we describe how to construct the candidate supports in $\mathcal{S}_d$ and explain why this set has tractable size. We build up to the general algorithm by explaining special cases that are easier to understand.

3.1. Rank-1 case

Let us start with the rank 1 case, i.e., when $d = 1$. For this case

$$A_1 = \lambda_1 v_1 v_1^T.$$

Assume, for now, that all the eigenvector entries are unique. This simplifies tie-breaking issues that are formally addressed by a perturbation lemma in (Papailiopoulos et al., 2013). For the rank-1 matrix A_1, a simple thresholding procedure solves sparse PCA: Simply keep the k largest entries of the eigenvector v_1. Hence, in this simple case $\mathcal{S}_1$ consists of only 1 set. To show this, we can rewrite (1) as

$$\max_{x \in \mathbb{S}_k} x^T A_1 x = \lambda_1 \cdot \max_{x \in \mathbb{S}_k} \left(v_1^T x\right)^2, \tag{3}$$

where $\mathbb{S}_k$ is the set of all vectors $x \in \mathbb{R}^n$ with $||x||_2 = 1$ and $||x||_0 = k$. Thus, we are trying to find a k-sparse vector x that maximizes the inner product with a given vector v_1. This problem is solved by sorting the absolute elements of the eigenvector v_1 and keeping the support of the k entries in v_1 with the largest absolute value.

Definition 1. *Let $\mathcal{I}_k(v)$ denote the set of indices of the top k largest absolute entries of a vector v.*

We can conclude that for the rank-1 case, the optimal k-sparse PC for A_1 will simply be the k-sparse vector that is co-linear to the k-sparse vector induced on this single candidate support: $\mathcal{S}_1 = \{\, \mathcal{I}_k(v_1) \,\}$.

3.2. Rank-2 case

Now we describe how to compute $\mathcal{S}_2$. This is the first nontrivial d which exhibits the details of the Spannogram algorithm. Here, we have the rank 2 matrix $A_2 = \sum_{i=1}^{2} \lambda_i v_i v_i^T = V_2 V_2^T$, where $V_2 =$

[2] Notice that the k-sparse largest eigenvalue of A for $k = 1$, denoted by $\lambda_1^{(1)}$, is simply the largest element on the diagonal of A.

$\left[\sqrt{\lambda_1}\cdot v_1 \;\; \sqrt{\lambda_2}\cdot v_2\right]$. We can rewrite (1) on A_2 as

$$\max_{x\in\mathbb{S}_k} x^T A_2 x = \max_{x\in\mathbb{S}_k} \left\|V_2^T x\right\|_2^2. \qquad (4)$$

In the rank-1 case we could write the quadratic form maximization as a simple maximization of a dot product: $\max_{x\in\mathbb{S}_k} x^T A_1 x = \max_{x\in\mathbb{S}_k} \left(v_1^T x\right)^2$. Similarly, we will prove that in the rank-2 case we can write

$$\max_{x\in\mathbb{S}_k} x^T A_2 x = \max_{x\in\mathbb{S}_k} \left(v_c^T x\right)^2,$$

for some *specific* vector v_c in the span of the eigenvectors v_1, v_2; this will be very helpful in solving the problem efficiently.

To see this, let c be a 2×1 unit length vector, i.e., $\|c\|_2 = 1$. Using the Cauchy-Schwartz inequality for the inner product of c and $V_2^T x$ we obtain $\left(c^T V_2^T x\right)^2 \le \|V_2^T x\|_2^2$, where equality holds, if and only if, c is co-linear to $V_2^T x$. By the previous fact, we have a variational characterization of the ℓ_2-norm:

$$\|V_2^T x\|_2^2 = \max_{\|c\|_2=1} \left(c^T V_2^T x\right)^2. \qquad (5)$$

We can use (5) to rewrite (4) as

$$\max_{x\in\mathbb{S}_k} \max_{\|c\|_2=1} \left(c^T V_2^T x\right)^2 = \max_{\|c\|_2=1} \max_{x\in\mathbb{S}_k} \left(v_c^T x\right)^2, \qquad (6)$$

where $v_c = V_2 c$. We would like to note two important facts here. The first is that for all unit vectors c, $v_c = V_2 c$ generates all vectors in the span of V_2 (up to scaling factors). The second fact is that if we fix c, then the maximization $\max_{x\in\mathbb{S}_k} \left(v_c^T x\right)^2$ is a rank-1 instance, similar to (3). Therefore, for each fixed unit vector c there will be one candidate support (denote it by $\mathcal{I}_k(V_2 c)$) to be added in $\mathcal{S}_2$.

If we could collect all possible candidate supports $\mathcal{I}_k(V_2 c)$ in

$$\mathcal{S}_2 = \bigcup_{c\in\mathbb{R}^{2\times 1}, \|c\|_2=1} \left\{\mathcal{I}_k(V_2 c)\right\}, \qquad (7)$$

then we could solve exactly the sparse PCA problem on A_2: we would simply need to test all locally optimal solutions obtained from each support in $\mathcal{S}_2$ and keep the one with the maximum metric. The issue is that there are infinitely many v_c vectors to check. Naively, one could think that all possible k-supports could appear for some v_c vector. The key combinatorial fact is that if a vector v_c lives in a two dimensional subspace, there are tremendously fewer possible supports [3]: $|\mathcal{S}_2| \le 4\binom{n}{2}$.

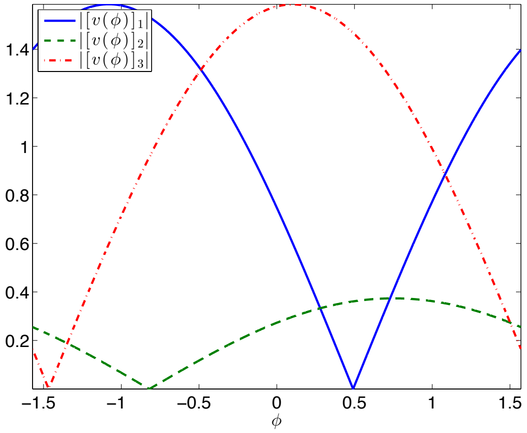

Figure 1. A rank-2 spannogram for a V_2 matrix with $n = 3$.

Spherical variables. Here we use a transformation of our problem space into a 2-dimensional space. The transformation is performed through spherical variables that enable us to visualize the 2-dimensional span of V_2. For the rank-2 case, we have a single phase variable $\phi \in \Phi = \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and use it to rewrite c, without loss of generality, as

$$c = \begin{bmatrix}\sin\phi \\ \cos\phi\end{bmatrix},$$

which is again unit norm and for all ϕ it scans all[4] 2×1 unit vectors. Under this characterization, we can express v_c in terms of ϕ as

$$v(\phi) = V_2 c = \sin\phi\cdot\sqrt{\lambda_1}v_1 + \cos\phi\cdot\sqrt{\lambda_2}v_2. \qquad (8)$$

Observe that each element of $v(\phi)$ is a continuous curve in ϕ: $[v(\phi)]_i = \left[\sqrt{\lambda_1}v_1\right]_i \sin(\phi) + \left[\sqrt{\lambda_2}v_2\right]_2 \cos(\phi)$, for all $i = 1, \ldots, n$. Therefore, the support set of the k largest absolute elements of $v(\phi)$ (i.e., $\mathcal{I}_k(v(\phi))$) is itself a function of ϕ.

The Spannogram. In Fig. 1, we draw an example plot of 3 (absolute) curves $|[v(\phi)]_i|$, $i = 1, 2, 3$, from a randomly generated matrix V_2. We call this a *spannogram*, because at each ϕ, the values of curves correspond to the absolute values of the elements in the column span of V_2. Computing $[v(\phi)]_i$ for all i, ϕ is equivalent to computing the span of V_2. From the spannogram in Fig. 1, we can see that the continuity of the curves implies a local invariance property of

[3] This is a special case of our general d dimensional lemma, but we prove the special case to simplify the presentation.

[4] Note that we restrict ourselves to $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$, instead of the whole $(-\pi, \pi]$ angle region. First observe that the vectors in the complement of Φ are opposite to the ones evaluated on Φ. Omitting the opposite vectors poses no issue due to the squaring in (4), i.e., vectors c and $-c$ map to the same solutions.

the support sets $\mathcal{I}(v(\phi))$, around a given ϕ. That is, we expect that $\mathcal{I}_k(v(\phi \pm \epsilon)) = \mathcal{I}_k(v(\phi))$, for a sufficiently small $\epsilon > 0$. As a matter of fact, a support set $\mathcal{I}_k(v(\phi))$ changes, *if and only if*, the respective sorting of two absolute elements $|[v(\phi)]_i|$ and $|[v(\phi)]_j|$ changes. Finding these intersection points $|[v(\phi)]_i| = |[v(\phi)]_j|$ is the key to find all possible support sets.

There are n curves and each pair intersects on exactly two points.[5] Therefore, there are exactly $2\binom{n}{2}$ intersection points. The intersection of two absolute curves are *exactly* two points ϕ that are a solution to $[v(\phi)]_i = [v(\phi)]_j$ and $[v(\phi)]_i = -[v(\phi)]_j$. These are the *only* points where local support sets might change. These $2\binom{n}{2}$ intersection points partition Φ in $2\binom{n}{2}+1$ regions within which the top k support sets remain invariant.

Building $\mathcal{S}_2$. To build $\mathcal{S}_2$, we need to *i)* determine all c intersection vectors that are defined at intersection points on the ϕ-axis and *ii)* compute all distinct locally optimal support sets $\mathcal{I}_k(v_c)$. To determine an intersection vector we need to solve all $2\binom{n}{2}$ equations $[v(\phi)]_i = \pm[v(\phi)]_j$ for all pairs $i, j \in [n]$. This yields $[v(\phi)]_i = \pm[v(\phi)]_j \Rightarrow e_i^T Vc = \pm e_j^T Vc$, that is

$$\left(e_i^T \pm e_j^T\right)Vc = 0 \Rightarrow c = \text{nullspace}\left(\left(e_i^T \pm e_j^T\right)V\right). \quad (9)$$

Since c needs to be unit norm, we simply need to normalize the solution c. We will refer to the intersection vector calculated on the ϕ of the intersection of two curves i and j as $c^+_{i,j}$ and $c^-_{i,j}$, depending on the corresponding sign in (9). For the intersection vectors $c^+_{i,j}$ and $c^-_{i,j}$ we compute $\mathcal{I}_k(V_2 c^+_{i,j})$ and $\mathcal{I}_k(V_2 c^-_{i,j})$. Observe that since the i and j curves are equal on the intersection points, there is no prevailing sorting among the two corresponding elements i and j of $V_2 c^+_{i,j}$ or $V_2 c^-_{i,j}$. Hence, for each intersection vector $c^+_{i,j}$ and $c^-_{i,j}$, we create two candidate support sets, one where element i is larger than j, and vice versa. This is done to secure that both support sets, left and right of the ϕ of the intersection, are included in $\mathcal{S}_2$. With the above methodology, we can compute all possible $\mathcal{I}_k(V_2 c)$ rank-2 optimal candidate sets and we obtain

$$|\mathcal{S}_2| \leq 4\binom{n}{2} = O(n^2).$$

The time complexity to build $\mathcal{S}_2$ is then equal to sorting $\binom{n}{2}$ vectors and solving $2\binom{n}{2}$ equations in the 2 unknowns of $c^+_{i,j}$ and $c^+_{i,j}$. That is, the total complexity is equal to $\binom{n}{2} n \log n + \binom{n}{2} 2^3 = O\left(n^3 \log n\right)$.

Algorithm 2 Spannogram Algorithm for $\mathcal{S}_d$.

```
Input: $k$, $p$, $V_d = \left[\sqrt{\lambda_1} v_1 \dots \sqrt{\lambda_1} v_1\right]$
Initialize $\mathcal{S}_d \leftarrow \emptyset$, $\mathcal{B} \leftarrow \{b_1, \dots, b_{d-1}\} \in \{\pm 1\}^{d-1}$
if $p = 1$ then
    $\mathcal{B} \leftarrow \{1, \dots, 1\}$, $V_d \leftarrow [V_d^T \; 0_{d\times 1}^T]^T$, $n \leftarrow n+1$
end if
for all $\binom{n}{d}$ subsets $(i_1, \dots, i_d)$ from $\{1, \dots, n\}$ do
    for all sequences $(b_1, \dots, b_{d-1}) \in \mathcal{B}$ do
        $c \leftarrow \text{nullspace}\left(\begin{bmatrix} e_{i_1}^T - b_1 \cdot e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} \cdot e_{i_d}^T \end{bmatrix} V_d\right)$
        if $p = 1$ then
            $\mathcal{I} \leftarrow$ {indices of the $k$-top elements of $Vc$}∪
                {indices of the $k$-top elements of $-Vc$}
        else
            $\mathcal{I} \leftarrow$ indices of the $k$-top elements of abs($Vc$)
        end if
        $l \leftarrow 1$
        $\mathcal{J}_1 \leftarrow \mathcal{I}_{1:k}$
        $r \leftarrow |\mathcal{J}_1 \cap (i_1, \dots, i_d)|$
        if $r < d$ then
            for all $r$-subsets $\mathcal{M}$ from $(i_1, \dots, i_d)$ do
                $l \leftarrow l + 1$
                $\mathcal{J}_l \leftarrow \mathcal{I}_{1:k-r} \cup \mathcal{M}$
            end for
        end if
        $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup \mathcal{J}_1 \ldots \cup \mathcal{J}_l$.
    end for
end for
Output: $\mathcal{S}_d$.
```

Remark 1. *The spannogram algorithm operates by simply solving systems of equations and sorting vectors. It is not iterative nor does it attempt to solve a convex optimization problem. Further, it computes solutions that are exactly k-sparse, where the desired sparsity can be set a-priori.*

The spannogram algorithm presented here is a subroutine that can be used to find the leading sparse PC of A_d in polynomial time. The general rank-d case is given as Algorithm 2. The details of our algorithm, the elimination step, and tune-ups for matrices with non-negative entries can be found in (Papailiopoulos et al., 2013).

4. Experimental Evaluation and Conclusions

We now empirically evaluate the performance of our algorithm and compare it to the full regularization path greedy approach (FullPath) of (d'Aspremont et al., 2007b), the generalized power method (GPower)

[5] As we mentioned, we assume that the curves are in "general position," i.e., no three curves intersect at the same point and this can be enforced by a small perturbation argument.

of (Journée et al., 2010), and the truncated power method (TPower) of (Yuan & Zhang, 2011). We omit the DSPCA semidefinite approximation of (d'Aspremont et al., 2007a), since the FullPath algorithm is experimentally shown to have similar or better performance (d'Aspremont et al., 2008). We begin with a synthetic experiment: we seek to estimate the support of the first two sparse eigenvectors of a covariance matrix from sample vectors. We continue with testing our algorithm on gene expression data sets. Finally, we run experiments on a large-scale document-term data set, comprising of millions of Twitter posts.

4.1. Spiked Covariance Recovery

We first test our approximation algorithm on an artificial data set generated in the same manner as in (Shen & Huang, 2008; Yuan & Zhang, 2011). We consider a covariance matrix Σ, which has two sparse eigenvectors with large eigenvalues; the remaining eigenvectors correspond to small eigenvalues. Here, we consider $\Sigma = \sum_{i=1}^{n} \lambda_i v_i v_i^T$ with $\lambda_1 = 400, \lambda_2 = 300, \lambda_3 = 1, \ldots, \lambda_{500} = 1$, where v_1, v_2 are sparse and each has 10 nonzero entries and non-overlapping supports.

We have two sets of experiments, one for few samples and one for extremely few. First, we generate $m = 50$ samples of length $n = 500$, distributed as zero mean Gaussian with covariance matrix Σ and repeat the experiment 5000 times. We repeat the same experiment for $m = 5$. We compare our rank-1 and rank-2 algorithms against FullPath, GPower with ℓ_1 penalization and ℓ_0 penalization, and TPower. After estimating the first eigenvector with $\tilde{v}_1$, we deflate A to obtain A'. We use the projection deflation method (Mackey, 2009) to obtain $A' = (I - \tilde{v}_1\tilde{v}_1^T)A(I - \tilde{v}_1\tilde{v}_1^T)$ and work on it to obtain $\tilde{v}_2$, the second estimated eigenvector of Σ. In the following table, we report the probability of correctly recovering the supports of v_1 and v_2: if both estimates $\tilde{v}_1$ and $\tilde{v}_2$ have matching supports with the true eigenvectors, then the recovery is considered successful. In our experiments for $m = 50$, all algorithms were comparable and performed near-optimally, apart from the rank-1 approximation (PCA+thresholding). For $m = 5$ samples we observe that the performance of the rank-1 and GPower methods decay and FullPath, TPower, and rank-2 find the correct support with probability approximately equal to 96%. This overall decay in performance of all schemes is due to the fact that 5 samples are not sufficient for a perfect estimate.

		500×50	500×5
	k	$p_{\text{rec.}}$	$p_{\text{rec.}}$
PCA+thresh.	10	.98	0.85
GPower-ℓ_0 ($\gamma = 0.8$)	10	1	0.33
GPower-ℓ_1 ($\gamma = 0.8$)	10	1	0.33
FullPath	10	1	0.96
TPower	10	1	0.96
Rank-2 approx.	10	1	0.96

4.2. Gene Expression Data Set

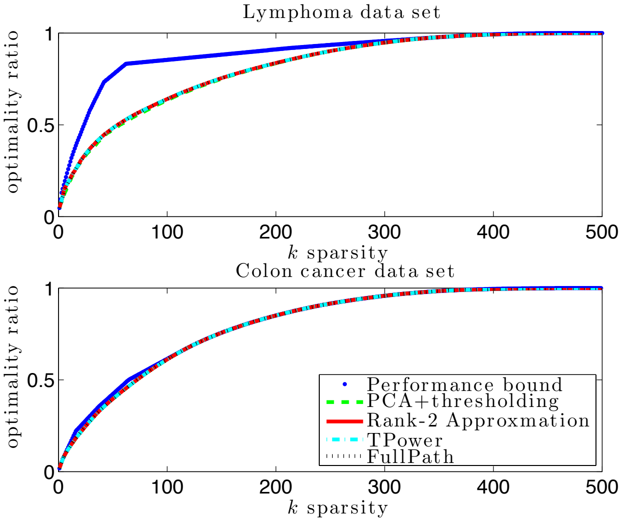

Figure 2. Results on gene expression data sets.

In the same manner as in the relevant sparse PCA literature, we evaluate our approximation on two gene expression data sets used in (d'Aspremont et al., 2007b; 2008; Yuan & Zhang, 2011). We plot the ratio of the explained variance coming from the first sparse PC to the explained variance of the first eigenvector (which is equal to the first eigenvalue). We also plot the performance outer bound derived in (d'Aspremont et al., 2008). We observe that our approximation follows the same optimality pattern as most previous methods, for many values of sparsity k. In these experiments we did not test the GPower method since the output sparsity cannot be explicitly predetermined. However, previous literature indicates that GPower is also near-optimal in this scenario.

4.3. Large-scale Twitter data set

Here, we evaluate our algorithm on a large-scale data set. Our data set comprises of millions of tweets coming from Greek Twitter users. Each tweet corresponds to a list of words and has a character limit of 140 per tweet. Although each tweet was associated with metadata, such us hyperlinks, user id, hash tags etc, we strip these features out and just use the word list. We use a simple Python script to normalize each Tweet. Words that are not contextual are discarded in an ad-

Rank-1	**TPower**	**Rank-2**	**Rank-3**	**fullPath**	**Rank-1**	**TPower**	**Rank-2**	**Rank-3**	**fullPath**
		sparse PC-1					sparse PC-3		
skype	~~eurovision~~	skype	skype	eurovision	~~downtown~~G	~~twitter~~	love	love	love
microsoft	skype	microsoft	microsoft	finalG	censusG	censusG	received	received	received
billion	microsoft	billion	acquisitionG	greeceG	~~athensG~~	homeG	greek	twitter	damon
acquisitionG	billion	acquisitionG	billion	greece	homeG	~~google~~	~~know~~	~~know~~	greek
~~eurovision~~	acquisitionG	acquiredG	acquiredG	lucasG	~~twitter~~	yearG	damon	greek	hate
acquiredG	buying	acquiresG	acquiresG	semifinalG	yearG	greek	amazing	damon	~~know~~
acquiresG	acquiredG	buying	buying	final	~~murderG~~	mayG	hate	hate	amazing
buying	acquiresG	dollarsG	dollarsG	contest	~~songG~~	~~facebook~~	twitter	amazing	sweet
google	dollarsG	acquisition	acquisition	stereo	mayG	startsG	great	great	great
dollarsG	acquisition	google	google	watching	yearsG	populationG	sweet	sweet	~~songs~~
		var./OPT					var./OPT		
0.9863	0.9861	**0.9870**	**0.9870**	0.9283	0.7875	0.7877	0.8993	**0.8994**	**0.8994**
		sparse PC-2					sparse PC-4		
greece	greece	eurovision	eurovision	skype	thanouG	downtownG	downtownG	downtownG	~~twitter~~
greeceG	greeceG	greece	greece	microsoft	kenterisG	athensG	athensG	athensG	~~facebook~~
love	love	greeceG	lucasG	billion	guiltyG	yearG	murderG	murderG	~~welcome~~
lucasG	loukas	finalG	finalG	acquisitionG	kenteris	year'sG	yearsG	yearsG	~~account~~
final	finalsG	lucasG	final	acquiresG	tzekosG	murderG	brutalG	brutalG	~~goodG~~
greek	~~athens~~	final	stereo	acquiredG	monthsG	cameraG	stabbedG	stabbedG	~~followers~~
~~athens~~	final	stereo	semifinalG	buying	tzekos	crimeG	bad_eventsG	bad_eventsG	censusG
finalG	stereo	semifinalG	contest	dollarsG	~~facebook~~	crime	yearG	cameraG	populationG
stereo	country	contest	greeceG	official	imprisonmentG	stabbedG	turmoilG	yearG	homeG
country	~~sailing~~	songG	watching	google	penaltiesG	brutalG	cameraG	crimeG	startsG
		var./OPT					var./OPT		
0.8851	0.8850	0.9850	**0.9852**	**0.9852**	0.7174	0.7520	0.8419	**0.8420**	0.8412

Table 1. The first 4 sparse PCs for a data set consisting of 65k Tweets and 64k unique words.

hoc way. We also discard all words that are less than three characters, or words that appear once in the corpus. We represent each tweet as a long vector consisting of n words, with a 1 whenever a word appears.[6] In the following tests, we compare against TPower and FullPath. TPower is run for $10k$ iterations, and is initialized with a vector having 1s on the k words of highest variance. For FullPath we restrict the covariance to its first 5k words of highest variance.[7] In our experiments, we use a simpler deflation method: once k words appear in the first k-sparse PC, we strip them from the data set, recompute the new convariance, and then run all algorithms. The performance metric here is again the explained variance over its maximum possible value.

In Table 2, we show our results for all tweets that contain the word `Japan`, for a 5-day and then a month-length time window. In all these tests, our rank-3 approximation consistently captured more variance than all other compared methods. In Table 1, we show a day-length experiment and report the first 4 sparse PCs of all methods. The average computation times for this time-window where less than 1 second for the rank-1 approximation, less than 5 seconds for rank-2, and less than 2 minutes for the rank-3 approximation on a Macbook Pro 5.1 running MATLAB 7. The main reason for these tractable running times is the use of our elimination scheme which left only around $40 - 80$ rows of the initial matrix of 64k rows. In terms of running speed, we empirically observed that our algorithm is slower than Tpower but faster than FullPath for the values of d tested. In Table 1, words with strike-through are what we consider non-matching to the "main topic" of that PC. Words marked with G are translated from Greek. From the PCs we see that the main topics are about Skype's acquisition by Microsoft, the European Music Contest "Eurovision", a crime that occurred in the downtown of Athens.

We conclude that our algorithm can efficiently provide interpretable sparse PCs and matches or outperforms the accuracy of previous methods. In terms of running speed, our algorithm is slower compared to the Tpower method and faster than FullPath for $d \leq 3$. A parallel implementation in the MapReduce framework and larger data studies are exciting future directions.

5. Acknowledgments

This work was supported by NSF Awards 1055099, 1218235 and research gifts by Google, Intel, and Microsoft.

	*japan	1-5 May 2011	May 2011
$m \times n$	12k × 15k	267k × 148k	1.9mil × 222k
k	$k = 10$	$k = 4$	$k = 5$
#PCs	5	7	3
Rank-1	0.600	0.815	0.885
TPower	0.595	0.869	0.915
Rank-2	**0.940**	0.934	0.885
Rank-3	**0.940**	**0.936**	**0.954**
FullPath	0.935	0.886	0.953

Table 2. Performance comparison on the Twitter data set.

[6] Further details about our data set, and the observed power-law decays of the spectrum, can be found in (Papailiopoulos et al., 2013).

[7] For $n = 30$k, FullPAth's running time on a dual-core machine was 3 hours (for 5 PCs) and followed a cubic growth in n, as expected. For a month length data set with $n = 220$k and no truncation, FullPath did not terminate after 20 hours.

References

Amini, A.A. and Wainwright, M.J. High-dimensional analysis of semidefinite relaxations for sparse principal components. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pp. 2454–2458. IEEE, 2008.

Asteris, M., Papailiopoulos, D.S., and Karystinos, G.N. Sparse principal component of a rank-deficient matrix. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 673–677. IEEE, 2011.

Cadima, J. and Jolliffe, I.T. Loading and correlations in the interpretation of principle compenents. *Journal of Applied Statistics*, 22(2):203–214, 1995.

d'Aspremont, A., El Ghaoui, L., Jordan, M.I., and Lanckriet, G.R.G. A direct formulation for sparse pca using semidefinite programming. *SIAM review*, 49(3):434–448, 2007a.

d'Aspremont, A., Bach, F., and Ghaoui, L.E. Optimal solutions for sparse principal component analysis. *The Journal of Machine Learning Research*, 9:1269–1294, 2008.

d'Aspremont, A., Bach, F., and Ghaoui, L.E. Approximation bounds for sparse principal component analysis. *arXiv preprint arXiv:1205.0121*, 2012.

d'Aspremont, Alexandre, Bach, Francis R., and Ghaoui, Laurent El. Full regularization path for sparse principal component analysis. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pp. 177–184, 2007b.

Gawalt, B., Zhang, Y., and El Ghaoui, L. Sparse pca for text corpus summarization and exploration. *NIPS 2010 Workshop on Low-Rank Matrix Approximation*, 2010.

Jolliffe, I.T. Rotation of principal components: choice of normalization constraints. *Journal of Applied Statistics*, 22(1):29–35, 1995.

Jolliffe, I.T., Trendafilov, N.T., and Uddin, M. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics*, 12(3):531–547, 2003.

Journée, M., Nesterov, Y., Richtárik, P., and Sepulchre, R. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

Kaiser, H.F. The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, 23(3):187–200, 1958.

Ma, Zongming. Sparse principal component analysis and iterative thresholding. *arXiv preprint arXiv:1112.2432*, 2011.

Mackey, L. Deflation methods for sparse pca. *Advances in neural information processing systems*, 21:1017–1024, 2009.

Moghaddam, B., Weiss, Y., and Avidan, S. Generalized spectral bounds for sparse lda. In *Proceedings of the 23rd international conference on Machine learning*, pp. 641–648. ACM, 2006a.

Moghaddam, B., Weiss, Y., and Avidan, S. Spectral bounds for sparse pca: Exact and greedy algorithms. *Advances in neural information processing systems*, 18:915, 2006b.

Moghaddam, B., Weiss, Y., and Avidan, S. Fast pixel/part selection with sparse eigenvectors. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8. IEEE, 2007.

Papailiopoulos, D. S., Dimakis, A. G., and Korokythakis, S. Sparse pca through low-rank approximations. *arXiv preprint arXiv:1303.0551*, 2013.

Shen, H. and Huang, J.Z. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034, 2008.

Sriperumbudur, B.K., Torres, D.A., and Lanckriet, G.R.G. Sparse eigen methods by dc programming. In *Proceedings of the 24th international conference on Machine learning*, pp. 831–838. ACM, 2007.

Yuan, X.T. and Zhang, T. Truncated power method for sparse eigenvalue problems. *arXiv preprint arXiv:1112.2679*, 2011.

Zhang, Y. and El Ghaoui, L. Large-scale sparse principal component analysis with application to text data. *Advances in Neural Information Processing Systems*, 2011.

Zhang, Y., d'Aspremont, A., and Ghaoui, L.E. Sparse pca: Convex relaxations, algorithms and applications. *Handbook on Semidefinite, Conic and Polynomial Optimization*, pp. 915–940, 2012.

Zou, H., Hastie, T., and Tibshirani, R. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.