Online Low-Rank Subspace Clustering by Basis Dictionary Pursuit

Jie Shen JS2007@RUTGERS.EDU
Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA

Ping Li PINGLI@STAT.RUTGERS.EDU
Dept. of Statistics & Biostatistics, Dept. of Computer Science, Rutgers University, Piscataway, NJ 08854, USA

Huan Xu ISEXUH@NUS.EDU.SG
Department of Industrial and Systems Engineering, National University of Singapore, Singapore

Abstract

Low-Rank Representation (LRR) has been a significant method for segmenting data that are generated from a union of subspaces. It is also known that solving LRR is challenging in terms of time complexity and memory footprint, in that the size of the nuclear norm regularized matrix is n-by-n (where n is the number of samples). In this paper, we thereby develop a novel online implementation of LRR that reduces the memory cost from $\mathcal{O}(n^2)$ to $\mathcal{O}(pd)$, with p being the ambient dimension and d being some estimated rank ($d < p \ll n$). We also establish the theoretical guarantee that the sequence of solutions produced by our algorithm converges to a stationary point of the expected loss function asymptotically. Extensive experiments on synthetic and realistic datasets further substantiate that our algorithm is fast, robust and memory efficient.

1. Introduction

In the past a few years, subspace clustering (Vidal, 2010; Soltanolkotabi & Candès, 2012) has been extensively studied and has established solid applications, for example, in computer vision (Elhamifar & Vidal, 2009) and network topology inference (Eriksson et al., 2011). Among many subspace clustering algorithms which aim to obtain a structured representation to fit the underlying data, two prominent examples are Sparse Subspace Clustering (SSC) (Elhamifar & Vidal, 2009; Soltanolkotabi et al., 2014) and Low-Rank Representation (LRR) (Liu et al., 2013). Both of them utilize the idea of self-expressiveness, i.e., expressing each sample as a linear combination of the remaining. What is of difference is that SSC pursues a sparse solution while LRR prefers a low-rank structure.

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

In this paper, we are interested in the LRR method, which is shown to achieve state-of-the-art performance on a broad range of real-world problems (Liu et al., 2013). Recently, Liu & Li (2014) demonstrated that, when equipped with a proper dictionary, LRR can even handle the coherent data – a challenging issue in the literature (Candès & Recht, 2009; Candès et al., 2011) but commonly emerges in realistic datasets such as the Netflix.

Formally, the LRR problem we investigate here is formulated as follows (Liu et al., 2013):

$$\min_{X,E} \frac{\lambda_1}{2} \left\| Z - YX - E \right\|_F^2 + \left\| X \right\|_* + \lambda_2 \left\| E \right\|_1 . \quad (1.1)$$

Here, $Z = (\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_n) \in \mathbb{R}^{p\times n}$ is the observation matrix with n samples lying in a p-dimensional subspace. The matrix $Y \in \mathbb{R}^{p\times n}$ is a given dictionary, E is some possible sparse corruption and λ_1 and λ_2 are two tunable parameters. Typically, Y is chosen as the dataset Z itself. The program seeks a low-rank representation $X \in \mathbb{R}^{n\times n}$ among all samples, each of which can be approximated by a linear combination of the atoms in the dictionary Y.

While LLR is mathematically elegant, three issues are immediately incurred for LRR in the face of big data:

Issue 1 (Memory cost of X)**.** In the LRR formulation (1.1), there is typically no sparsity assumption on X. Hence, the memory footprint of X is proportional to n^2 which precludes most of the recently developed nuclear norm solvers (Lin et al., 2010; Jaggi & Sulovský, 2010; Avron et al., 2012; Hsieh & Olsen, 2014).

Issue 2 (Computational cost of $\left\| X \right\|_*$)**.** Due to the size of the nuclear norm regularized matrix X is $n \times n$, optimizing such problems can be computationally expensive even when n is not too large (Recht et al., 2010).

Issue 3 (Memory cost of Y)**.** Since the dictionary size is $p \times n$, it is prohibitive to store the entire dictionary Y during optimization when manipulating a huge volume of data.

To remedy these issues, especially the memory bottleneck, one potential way is solving the problem in online manner. That is, we sequentially reveal the samples $\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_n$ and update the components in X and E. Nevertheless, such strategy appears difficult to execute due the the residual term in (1.1). To be more precise, we note that each column of X is the coefficients of a sample with respect to the *entire* dictionary Y, e.g., $\boldsymbol{z}_1 \approx Y\boldsymbol{x}_1 + \boldsymbol{e}_1$. This indicates that without further technique, we have to load the entire dictionary Y so as to update the columns of X. Hence, for our purpose, we need to tackle a more serious challenge:

Issue 4 (Partial realization of Y)**.** We are required to guarantee the optimality of the solution but can only access part of the atoms of Y in each iteration.

Related Works. There are a vast body of works attempting to mitigate the memory and computational bottleneck of the nuclear norm regularizer. However, to the best of our knowledge, none of them can handle Issue 3 and Issue 4.

One of the most popular ways to alleviate the huge memory cost is online implementation. Feng et al. (2013) devised an online algorithm for the Robust Principal Component Analysis (RPCA) problem, which makes the memory cost independent of the sample size. Yet, compared to RPCA where the size of the nuclear norm regularized matrix is $p \times n$, that of LRR is $n \times n$ – a worse and more challenging case. Moreover, their algorithm cannot address the partial dictionary issue that emerges in our case.

To tackle the computational overhead, Jaggi & Sulovský (2010) utilized a sparse semi-definite programming solver to derive a simple yet efficient algorithm. Unfortunately, the memory requirement of their algorithm is proportional to the number of observed entries, making it impractical when the regularized matrix is large and dense (which is the case of LRR). Avron et al. (2012) combined stochastic subgradient and incremental SVD to boost efficiency. But for the LRR problem, the type of the loss function does not meet the requirements and thus, it is still not practical to use that algorithm in our case.

Another line in the literature explores a structured formulation of LRR beyond the low-rankness. For example, Wang et al. (2013) provably showed that combining LRR with SSC can take advantages of both methods. Shen & Li (2016) argued that the vanilla LRR program does not fully characterize the nature of multiple subspaces, and presented several effective alternatives to LRR.

Summary of Contributions. In this paper, henceforth, we propose a new algorithm called Online Low-Rank Subspace Clustering (OLRSC), which admits a low computational complexity. In contrast to existing solvers, OLRSC reduces the memory cost of LRR from $\mathcal{O}(n^2)$ to $\mathcal{O}(pd)$ ($d < p \ll n$). This nice property makes OLRSC an appealing solution for large-scale subspace clustering problems. Furthermore, we prove that the sequence of solutions produced by OLRSC converges to a stationary point of the expected loss function asymptotically even though only one atom of Y is available at each iteration. In a nutshell, OLRSC resolves *all* practical issues of LRR and still promotes global low-rank structure – the merit of LRR.

2. Problem Formulation

Notation. We use bold lowercase letters, e.g. $\boldsymbol{v}$, to denote a column vector. The ℓ_2 norm and ℓ_1 norm of a vector $\boldsymbol{v}$ are denoted by $\left\|\boldsymbol{v}\right\|_2$ and $\left\|\boldsymbol{v}\right\|_1$ respectively. Capital letters such as M are used to denote a matrix, and its transpose is denoted by $M^\top$. For an invertible matrix M, we write its inverse as M^{-1}. The capital letter I_r is reserved for identity matrix where r indicates the size. The jth column of a matrix M is denoted by $\boldsymbol{m}_j$ if not specified. Three matrix norms will be used: $\left\|M\right\|_*$ for the nuclear norm, $\left\|M\right\|_F$ for the Frobenius norm and $\left\|M\right\|_1$ for the ℓ_1 norm of a matrix seen as a long vector. The trace of a square matrix M is denoted by $\mathrm{Tr}(M)$. For an integer n, we use $[n]$ to denote the integer set $\{1, 2, \cdots, n\}$.

Our goal is to efficiently learn the representation matrix X and the corruption matrix E in an online manner so as to mitigate the issues mentioned in Section 1. The first technique for our purpose is a *non-convex reformulation* of the nuclear norm. Assume the rank of X is at most d. Then Fazel et al. (2001) showed that,

$$\left\|X\right\|_* = \min_{U,V,X=UV^\top} \frac{1}{2}\left(\left\|U\right\|_F^2 + \left\|V\right\|_F^2\right), \tag{2.1}$$

where $U \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{n\times d}$. The minimum can be attained at, for example, $U = U_0 S_0^{\frac{1}{2}}$ and $V = V_0 S_0^{\frac{1}{2}}$ where $X = U_0 S_0 V_0^\top$ is the singular value decomposition.

In this way, (1.1) can be written as follows:

$$\min_{U,V,E} \frac{\lambda_1}{2}\left\|Z - YUV^\top - E\right\|_F^2 + \frac{1}{2}\left\|U\right\|_F^2 + \frac{1}{2}\left\|V\right\|_F^2 + \lambda_2\left\|E\right\|_1 .$$

Note that by this reformulation, updating the entries in X amounts to sequentially updating the rows of U and V. Also note that this technique is utilized in Feng et al. (2013) for online RPCA. Unfortunately, the size of U and V in our problem are both proportional to n and the dictionary Y is partially observed in each iteration, making the algorithm in Feng et al. (2013) not applicable to LRR. Related to online implementation, another challenge is that, all the rows of U are coupled together at this moment as U is left multiplied by Y in the first term. This makes it difficult to sequentially update the rows of U.

For the sake of decoupling the rows of U, as part of the crux of our technique, we introduce an auxiliary variable $D = YU$, whose size is $p \times d$ (i.e., independent of the sample size n). Interestingly, in this way, we are approximating the term $Z - E$ with $DV^\top$, which provides an intuition on the role of D: Namely, D can be seen as a *basis dictionary* of the clean data, with V being the coefficients.

These key observations allow us to derive an equivalent reformulation of LRR (1.1):

$$\min_{D,U,V,E} \frac{\lambda_1}{2} \left\| Z - YUV^\top - E \right\|_F^2 + \frac{1}{2}\left(\|U\|_F^2 + \|V\|_F^2 \right) + \lambda_2 \|E\|_1, \quad \text{s.t. } D = YU.$$

By penalizing the constraint in the objective, we obtain a *regularized* version of LRR on which our OLRSC is based:

$$\min_{D,U,V,E} \frac{\lambda_1}{2} \left\| Z - DV^\top - E \right\|_F^2 + \frac{1}{2}\left(\|U\|_F^2 + \|V\|_F^2 \right) + \lambda_2 \|E\|_1 + \frac{\lambda_3}{2} \|D - YU\|_F^2. \tag{2.2}$$

Remark 1 (Superiority to LRR)**.** There are two advantages of (2.2) compared to (1.1). First, it is amenable for online optimization. Second, it is more informative since it explicitly models the basis of the union of subspaces, hence a better subspace recovery and clustering (see Section 5). This actually meets the core idea of (Liu & Li, 2014) but they assumed Y contains true subspaces.

Remark 2 (Connection to RPCA)**.** Due to our explicit modeling of the basis, we unify LRR and RPCA as follows: for LRR, $D \approx YU$ (or $D = YU$ if λ_3 tends to infinity) while for RPCA, $D = U$. That is, ORPCA (Feng et al., 2013) considers a problem of $Y = I_p$ whose size is independent of n, hence can be kept in memory which naturally resolves Issue 3 and 4. This is why RPCA can be easily implemented in an online fashion while LRR cannot.

Remark 3 (Connection to Dictionary Learning)**.** Generally speaking, LRR (1.1) can be seen as a coding algorithm, with the dictionary Y known in advance and X is a desired structured code while other popular algorithms such as dictionary learning (DL) (Mairal et al., 2010) simultaneously optimizes the dictionary and the sparse code. Interestingly, in view of (2.2), the link of LRR and DL becomes more clear in the sense that the difference lies in the way how the dictionary is constrained. That is, for LRR we have $D \approx YU$ and U is further regularized by Frobenius norm whereas for DL, we have $\|\boldsymbol{d}_i\|_2 \leq 1$ for each column of D.

Let $\boldsymbol{z}_i$, $\boldsymbol{y}_i$, $\boldsymbol{e}_i$, $\boldsymbol{u}_i$, and $\boldsymbol{v}_i$ be the ith column of matrices Z, Y, E, $U^\top$ and $V^\top$ respectively and define

$$\tilde{\ell}(\boldsymbol{z}, D, \boldsymbol{v}, \boldsymbol{e}) \overset{\text{def}}{=} \frac{\lambda_1}{2} \|\boldsymbol{z} - D\boldsymbol{v} - \boldsymbol{e}\|_2^2 + \frac{1}{2}\|\boldsymbol{v}\|_2^2 + \lambda_2 \|\boldsymbol{e}\|_1,$$
$$\ell(\boldsymbol{z}, D) = \min_{\boldsymbol{v},\boldsymbol{e}} \tilde{\ell}(\boldsymbol{z}, D, \boldsymbol{v}, \boldsymbol{e}), \tag{2.3}$$
$$\tilde{h}(Y, D, U) \overset{\text{def}}{=} \sum_{i=1}^n \frac{1}{2}\|\boldsymbol{u}_i\|_2^2 + \frac{\lambda_3}{2} \left\| D - \sum_{i=1}^n \boldsymbol{y}_i \boldsymbol{u}_i^\top \right\|_F^2,$$
$$h(Y, D) = \min_U \tilde{h}(Y, D, U). \tag{2.4}$$

Then (2.2) can be rewritten as:

$$\min_D \min_{U,V,E} \sum_{i=1}^n \tilde{\ell}(\boldsymbol{z}_i, D, \boldsymbol{v}_i, \boldsymbol{e}_i) + \tilde{h}(Y, D, U), \tag{2.5}$$

which amounts to minimizing the empirical loss function:

$$f_n(D) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \ell(\boldsymbol{z}_i, D) + \frac{1}{n} h(Y, D). \tag{2.6}$$

In stochastic optimization, we are interested in analyzing the optimality of the obtained solution with respect to the expected loss function. To this end, we first derive the optimal solutions U^*, V^* and E^* that minimize (2.5) which renders a concrete form of the empirical loss function $f_n(D)$, hence we are able to derive the expected loss.

Given D, we need to compute the optimal solutions U^*, V^* and E^* to evaluate the objective value of $f_n(D)$. What is of interest here is that, the optimization procedure of U is totally different from that of V and E. According to (2.3), when D is given, each $\boldsymbol{v}_i^*$ and $\boldsymbol{e}_i^*$ can be solved by only accessing the ith sample $\boldsymbol{z}_i$. However, the optimal $\boldsymbol{u}_i^*$ depends on the whole dictionary Y as the second term in $\tilde{h}(Y, D, U)$ couples all the $\boldsymbol{u}_i$'s. Fortunately, we can obtain a closed form solution for the $\boldsymbol{u}_i^*$'s (see Appendix B):

$$\boldsymbol{u}_i^* = \frac{1}{n} D^\top \left(\frac{1}{\lambda_3 n} I_p + \frac{1}{n} N_n\right)^{-1} \boldsymbol{y}_i, \ \forall\, i \in [n], \tag{2.7}$$

where $N_n = \sum_{i=1}^n \boldsymbol{y}_i \boldsymbol{y}_i^\top \in \mathbb{R}^{p\times p}$. Thus, we have

$$h(Y, D) = \frac{1}{n^2} \sum_{i=1}^n \frac{1}{2} \left\| D^\top \left(\frac{1}{\lambda_3 n} I_p + \frac{1}{n} N_n \right)^{-1} \boldsymbol{y}_i \right\|_2^2 + \frac{\lambda_3}{2n^2} \left\| \left(\frac{1}{n} I_p + \frac{\lambda_3}{n} N_n \right)^{-1} D \right\|_F^2.$$

Now we derive the expected loss function, which is defined as the limit of the empirical loss function when n tends to infinity. If we assume that all the samples are drawn i.i.d. from some unknown distribution, we have

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^n \ell(\boldsymbol{z}_i, D) = \mathbb{E}_{\boldsymbol{z}}[\ell(\boldsymbol{z}, D)].$$

If we further assume that the smallest singular value of $\frac{1}{n}N_n$ is bounded away from zero (which implies the spectrum of N_n^{-1} is bounded from the above), we have

$$0 \leq \lim_{n\to\infty} \frac{1}{n} h(Y, D) \leq \lim_{n\to\infty} \frac{1}{n^3} \sum_{i=1}^{n} \mathrm{C}_0 = 0.$$

Here C_0 is some constant since D is fixed and $\boldsymbol{y}_i$'s are bounded. Hence, it follows that $\lim_{n\to\infty} h(Y,D)/n = 0$. Finally, the expected loss function is given by

$$f(D) \overset{\text{def}}{=} \lim_{n\to\infty} f_n(D) = \mathbb{E}_{\boldsymbol{z}}[\ell(\boldsymbol{z}, D)]. \tag{2.8}$$

3. Algorithm

Our OLRSC algorithm is summarized in Algorithm 1. Recall that OLRSC is an online implementation to solve (2.6), which is derived from the regularized version of LRR (2.2). We optimize the variables in an alternative manner. At the t-th iteration, when the basis dictionary D_{t-1} is given, the optimal solutions $\{\boldsymbol{v}_t, \boldsymbol{e}_t\}$ are produced by minimizing the objective function $\tilde{\ell}(\boldsymbol{z}_t, D_{t-1}, \boldsymbol{v}, \boldsymbol{e})$ over $\boldsymbol{v}$ and $\boldsymbol{e}$. We observe that if $\boldsymbol{e}$ is fixed, we can optimize $\boldsymbol{v}$ in closed form:

$$\boldsymbol{v} = (D_{t-1}^\top D_{t-1} + I_d/\lambda_1)^{-1} D_{t-1}^\top (\boldsymbol{z}_t - \boldsymbol{e}). \tag{3.1}$$

Conversely, given $\boldsymbol{v}$, $\boldsymbol{e}$ is obtained via soft-thresholding:

$$\boldsymbol{e} = \mathcal{S}_{\lambda_2/\lambda_1}[\boldsymbol{z}_t - D_{t-1}\boldsymbol{v}]. \tag{3.2}$$

Thus, we utilize coordinate descent algorithm to optimize $\boldsymbol{v}$ and $\boldsymbol{e}$. See Algorithm 2 in Appendix C for details.

To obtain $\boldsymbol{u}_t$, we need to compute an accumulation matrix $M_{t-1} = \sum_{i=1}^{t-1} \boldsymbol{y}_i \boldsymbol{u}_i^\top$ where we define $M_0 = 0$. In this way, $\boldsymbol{u}_t$ is given by minimizing

$$\begin{aligned}&\tilde{\ell}_2(\boldsymbol{y}_t, D_{t-1}, M_{t-1}, \boldsymbol{u}) \\ \overset{\text{def}}{=} &\frac{1}{2}\left\|\boldsymbol{u}\right\|_2^2 + \frac{\lambda_3}{2}\left\|D_{t-1} - M_{t-1} - \boldsymbol{y}_t \boldsymbol{u}^\top\right\|_F^2,\end{aligned} \tag{3.3}$$

for which we have the closed form solution

$$\boldsymbol{u}_t = (\left\|\boldsymbol{y}_t\right\|_2^2 + 1/\lambda_3)^{-1}(D_{t-1} - M_{t-1})^\top \boldsymbol{y}_t. \tag{3.4}$$

As we discussed in Section 2, the optimal values of variables $\boldsymbol{v}_t$ and $\boldsymbol{e}_t$ can be "accurately" solved by only accessing the t-th sample $\boldsymbol{z}_t$ if D_{t-1} is given. However, this does not hold for the variable $\boldsymbol{u}_t$. In fact, even though D_{t-1} is given, the optimal $\boldsymbol{u}_t$ depends on the entire dictionary Y, see (2.7). In online optimization, we only store the current atom $\boldsymbol{y}_t$. Thus, in order to obtain desirable estimation for $\boldsymbol{u}_t$, we have to "approximately" solve $\boldsymbol{u}_t$[1].

[1]Note that "accurately" and "approximately" here mean that when only D_{t-1}, $\boldsymbol{z}_t$ and $\boldsymbol{y}_t$ are given, whether we can obtain the same solution $\{\boldsymbol{v}_t, \boldsymbol{e}_t, \boldsymbol{u}_t\}$ as for the batch problem (2.6).

Algorithm 1 Online Low-Rank Subspace Clustering

```
Require: $Z \in \mathbb{R}^{p\times n}$ (observed samples), $Y \in \mathbb{R}^{p\times n}$, parameters $\lambda_1$, $\lambda_2$ and $\lambda_3$, random matrix $D_0 \in \mathbb{R}^{p\times d}$ (initial basis), zero matrix $M_0$, $A_0$ and $B_0$.
Ensure: Optimal basis $D_n$.
1: for $t = 1$ to $n$ do
2:     Access the $t$-th sample $\boldsymbol{z}_t$ and the $t$-th atom $\boldsymbol{y}_t$.
3:     Compute the coefficient and noise:
           $\{\boldsymbol{v}_t, \boldsymbol{e}_t\} = \arg\min_{\boldsymbol{v},\boldsymbol{e}} \tilde{\ell}(\boldsymbol{z}_t, D_{t-1}, \boldsymbol{v}, \boldsymbol{e})$,
           $\boldsymbol{u}_t = \arg\min_{\boldsymbol{u}} \tilde{\ell}_2(\boldsymbol{y}_t, D_{t-1}, M_{t-1}, \boldsymbol{u})$.
4:     Update the accumulation matrices:
           $M_t \leftarrow M_{t-1} + \boldsymbol{y}_t \boldsymbol{u}_t^\top$,
           $A_t \leftarrow A_{t-1} + \boldsymbol{v}_t \boldsymbol{v}_t^\top$,
           $B_t \leftarrow B_{t-1} + (\boldsymbol{z}_t - \boldsymbol{e}_t)\boldsymbol{v}_t^\top$.
5:     Update the basis:
           $D_t = \arg\min_D \frac{1}{t}\Big[\frac{1}{2}\operatorname{Tr}\big(D^\top D(\lambda_1 A_t + \lambda_3 I_d)\big) - \operatorname{Tr}\big(D^\top(\lambda_1 B_t + \lambda_3 M_t)\big)\Big]$.
6: end for
```

Intuition for $\tilde{\ell}_2(\boldsymbol{y}_t, D_{t-1}, M_{t-1}, \boldsymbol{u})$. Actually, our strategy can be seen as a one-pass block coordinate descent algorithm for the objective function $\tilde{h}(Y, D_{t-1}, U)$. Assume we have n atoms in total. We initialize all the $\boldsymbol{u}$'s with a zero vector. After accessing the t-th atom $\boldsymbol{y}_t$, we only update $\boldsymbol{u}_t$ while keeping the other $\boldsymbol{u}$'s. In this way, optimizing $\tilde{h}(Y, D_{t-1}, U)$ amounts to minimizing the function $\tilde{\ell}_2(\boldsymbol{y}_t, D_{t-1}, M_{t-1}, \boldsymbol{u})$. So after revealing all the atoms, each $\boldsymbol{u}_t$ is sequentially updated only once.

As soon as $\{\boldsymbol{v}_i, \boldsymbol{e}_i, \boldsymbol{u}_i\}_{i=1}^t$ are available, we can compute a new iterate D_t by optimizing the surrogate function

$$\begin{aligned}g_t(D) \overset{\text{def}}{=} \frac{1}{t}\Big(&\sum_{i=1}^{t} \tilde{\ell}(\boldsymbol{z}_i, D, \boldsymbol{v}_i, \boldsymbol{e}_i) + \sum_{i=1}^{t} \frac{1}{2}\left\|\boldsymbol{u}_i\right\|_2^2 \\ &+ \frac{\lambda_3}{2}\left\|D - M_t\right\|_F^2\Big).\end{aligned} \tag{3.5}$$

Expanding the first term, we find that D_t is given by

$$\begin{aligned}D_t &= \arg\min_D \frac{1}{t}\Big[\frac{1}{2}\operatorname{Tr}\big(D^\top D(\lambda_1 A_t + \lambda_3 I_d)\big) \\ &\qquad - \operatorname{Tr}\big(D^\top(\lambda_1 B_t + \lambda_3 M_t)\big)\Big] \\ &= (\lambda_1 B_t + \lambda_3 M_t)(\lambda_1 A_t + \lambda_3 I_d)^{-1},\end{aligned} \tag{3.6}$$

where $A_t = \sum_{i=1}^t \boldsymbol{v}_i \boldsymbol{v}_i^\top$ and $B_t = \sum_{i=1}^t (\boldsymbol{z}_i - \boldsymbol{e}_i)\boldsymbol{v}_i^\top$. We point out that the size of A_t is $d \times d$ and that of B_t is $p \times d$, i.e., independent of sample size.

Memory Cost. It is remarkable that the memory cost of Algorithm 1 is $\mathcal{O}(pd)$. To see this, note that when solving $\boldsymbol{v}_t$ and $\boldsymbol{e}_t$, we load the auxiliary variable D_t and a sample $\boldsymbol{z}_t$ into the memory, which costs $\mathcal{O}(pd)$. To compute the optimal $\boldsymbol{u}_t$'s, we need to access D_t and $M_t \in \mathbb{R}^{p\times d}$. Although we aim to minimize (3.5), which seems to require all the past information, we actually only need to record A_t, B_t and M_t, whose sizes are at most $\mathcal{O}(pd)$ (since $d < p$).

Computational Efficiency. In addition to memory efficiency, we further elaborate that the the computation in each iteration is cheap. To compute $\{\boldsymbol{v}_t, \boldsymbol{e}_t\}$, one may utilize the block coordinate method in Richtárik & Takác (2014) which enjoys linear convergence due to strong convexity. One may also apply the stochastic variance reduced algorithms which also ensure a geometric rate of convergence (Xiao & Zhang, 2014; Defazio et al., 2014). The $\boldsymbol{u}_t$ is given by simple matrix-vector multiplication, which costs $\mathcal{O}(pd)$. It is easy to see the complexity of Step 4 is $\mathcal{O}(pd)$ and that of Step 5 is $\mathcal{O}(pd^2)$.

A Fully Online Scheme. Now we have provided a way to (approximately) optimize the LRR problem (1.1) in online fashion. Usually, researchers in the literature will take an optional post-processing step to refine the segmentation accuracy, for example, applying spectral clustering on the representation matrix X. In this case, one has to collect all the $\boldsymbol{u}_i$'s and $\boldsymbol{v}_i$'s to compute $X = UV^\top$ which again increases the memory cost to $\mathcal{O}(n^2)$. Here, we suggest an alternative scheme which admits $\mathcal{O}(kd)$ memory usage where k is the number of subspaces. The idea is utilizing the well-known k-means on $\boldsymbol{v}_i$'s. There are two notable advantages compared to the spectral clustering. First, updating the k-means model can be implemented in online manner and the computation is $\mathcal{O}(kd)$. Second, we observe that $\boldsymbol{v}_i$ is actually a robust feature for the ith sample.

An Alternative Online Implementation. Our strategy for solving $\boldsymbol{u}_t$ is based on an approximate routine which resolves Issue 4 as well as has a low complexity. Yet, to tackle Issue 4, another potential way is to avoid the variable $\boldsymbol{u}_t$[2]. Recall that we derive the optimal solution U^* (provided that D is given) to (2.2) as follows (see Appendix B):

$$U^* = Y^\top \left(\lambda_3^{-1} I_p + YY^\top\right)^{-1} D.$$

Plugging it back to (2.2), we obtain

$$\|U^*\|_F^2 = \operatorname{Tr}\left(DD^\top \left(Q_n - {\lambda_3}^{-1} Q_n^2\right)\right),$$
$$\|D - YU^*\|_F^2 = \left\|D - {\lambda_3}^{-1} Q_n D\right\|_F^2,$$

where

$$Q_n = \left({\lambda_3}^{-1} I_p + YY^\top\right)^{-1}.$$

Here, the subscript of Q_n denotes the number of atoms in Y. Note that the size of Q_n is $p \times p$. Hence, if we incrementally compute the accumulation matrix $YY^\top = \sum_{i=1}^t \boldsymbol{y}_i \boldsymbol{y}_i^\top$, we can update the variable D in an online fashion. Namely, at t-th iteration, we re-define the surrogate function as follows:

$$g_t(D) \overset{\text{def}}{=} \frac{1}{t}\left[\sum_{i=1}^t \tilde{\ell}(\boldsymbol{z}_i, D, \boldsymbol{v}_i, \boldsymbol{e}_i) + \frac{\lambda_3}{2}\left\|D - \frac{1}{\lambda_3} Q_t D\right\|_F^2 + \frac{1}{2}\operatorname{Tr}\left(DD^\top\left(Q_t - \frac{1}{\lambda_3} Q_t^2\right)\right)\right].$$

Again, by noting the fact that $\tilde{\ell}(\boldsymbol{z}_i, D, \boldsymbol{v}_i, \boldsymbol{e}_i)$ only involves recording A_t and B_t, we show that the memory cost is independent of sample size.

While promising since the above procedure avoids the approximate computation, the main shortcoming is computing the inverse of a $p \times p$ matrix in each iteration, hence not efficient. Moreover, as we will show in Theorem 1, although the $\boldsymbol{u}_t$'s are approximate solutions, we are still guaranteed the convergence of D_t.

4. Theoretical Analysis

We make three assumptions underlying our analysis.

Assumption 1. The observed data are generated i.i.d. from some distribution and there exist constants α_0 and α_1, such that the conditions $0 < \alpha_0 \leq \|\boldsymbol{z}\|_2 \leq \alpha_1$ and $\alpha_0 \leq \|\boldsymbol{y}\|_2 \leq \alpha_1$ hold almost surely.

Assumption 2. The smallest singular value of the matrix $N_t = \frac{1}{t}\sum_{i=1}^t \boldsymbol{y}_i \boldsymbol{y}_i^\top$ is lower bounded away from zero.

Assumption 3. The surrogate functions $g_t(D)$ are strongly convex for all $t \geq 0$.

Based on these assumptions, we establish the main theoretical result, justifying the validity of Algorithm 1.

Theorem 1. *Let $\{D_t\}_{t=1}^\infty$ be the sequence of optimal bases produced by Algorithm 1. Then, the sequence converges to a stationary point of $f(D)$ (2.8) when t goes to infinity.*

Note that since the reformulation of the nuclear norm (2.1) is non-convex, we can only guarantee that the solution is a stationary point in general (Bertsekas, 1999). We also remark that OLRSC asymptotically fulfills the first order optimality condition of (1.1). To see this, we follow the proof technique of Prop.3 in Mardani et al. (2015) and let $X = UV^\top$, $W_1 = UU^\top$, $W_2 = VV^\top$, $M_1 = M_3 = 0.5I$, $M_2 = M_4 = 0.5\lambda_1 Y^\top(YX + E - Z)$. Due to our uniform bound (Prop. 7), we justify the optimality condition.

[2]We'd like to thank the anonymous NIPS 2015 Reviewer for pointing out this potential solution to the online algorithm. Here we explain why this alternative can be computationally expensive.

More interestingly, as we mentioned in Section 3, the solution (3.4) is not accurate in the sense that it is not equal to that of (2.7) given D. Yet, our theorem asserts that this will not deviate $\{D_t\}_{t\geq 0}$ away from the stationary point. The intuition underlying such amazing phenomenon is that the expected loss function (2.8) is only determined by $\ell(\boldsymbol{z}, D)$ which does not involve $\boldsymbol{u}_t$. What is of matter for $\boldsymbol{u}_t$ and M_t is their uniform boundedness and concentration to establish the convergence. Thanks to the carefully chosen function $\tilde{\ell}(\boldsymbol{z}, D, M, \boldsymbol{u})$ and the surrogate function $g_t(D)$, we are able to prove the desired property by mathematical induction which is a crucial step in our proof.

In particular, we have the following lemma that facilitates our analysis:

Lemma 2. *Let $\{M_t\}_{t\geq 0}$ be the sequence of the matrices produced by Algorithm 1. Then, there exists some universal constant C_0, such that for all $t \geq 0$, $\|M_t\|_F \leq \mathrm{C}_0$.*

Due to the above lemma, the solution D_t is essentially determined by A_t/t and B_t/t when t is a very large quantity since $M_t/t \to 0$. We also have a non-asymptotic rate for the numerical convergence of D_t as $\|D_t - D_{t-1}\|_2 = \mathcal{O}(1/t)$. See Appendix F for more details and a full proof.

5. Experiments

Before presenting the empirical results, we first introduce the universal settings used throughout the section.

Algorithms. For the subspace recovery task, we compare our algorithm with ORPCA (Feng et al., 2013), LRR (Liu et al., 2013) and PCP (Candès et al., 2011). For the subspace clustering task, we choose ORPCA, LRR and SSC (Elhamifar & Vidal, 2009) as the competitive baselines. Recently, Liu & Li (2014) improved the vanilla LRR by utilizing some low-rank matrix for Y. We denote this variant of LRR by LRR2 and accordingly, our algorithm equipped with such Y is denoted as OLRSC2.

Evaluation Metric. We evaluate the fitness of the recovered subspaces D (with each column being normalized) and the ground truth L by the Expressed Variance (EV) (Xu et al., 2010):

$$\mathrm{EV}(D, L) \overset{\text{def}}{=} \mathrm{Tr}(DD^\top LL^\top)/\mathrm{Tr}(LL^\top). \tag{5.1}$$

The value of EV scales between 0 and 1, and a higher value means better recovery.

The performance of subspace clustering is measured by clustering accuracy, which also ranges in the interval $[0, 1]$, and a higher value indicates a more accurate clustering.

Parameters. We set $\lambda_1 = 1$, $\lambda_2 = 1/\sqrt{p}$ and $\lambda_3 = \sqrt{t/p}$, where t is the iteration counter. These settings are actually used in ORPCA. We follow the default parameter setting for the baselines.

5.1. Subspace Recovery

Simulation Data. We use 4 disjoint subspaces $\{\mathcal{S}_k\}_{k=1}^4 \subset \mathbb{R}^p$, whose bases are denoted by $\{L_k\}_{k=1}^4 \in \mathbb{R}^{p\times d_k}$. The clean data matrix $\bar{Z}_k \in \mathcal{S}_k$ is then produced by $\bar{Z}_k = L_k R_k^\top$, where $R_k \in \mathbb{R}^{n_k \times d_k}$. The entries of L_k's and R_k's are sampled i.i.d. from the normal distribution. Finally, the observed data matrix Z is generated by $Z = \bar{Z} + E$, where $\bar{Z}$ is the column-wise concatenation of $\bar{Z}_k$'s followed by a random permutation, E is the sparse corruption whose ρ fraction entries are non-zero and follow an i.i.d. uniform distribution over $[-2, 2]$. We independently conduct each experiment 10 times and report the averaged results.

Robustness. We illustrate by simulation results that OLRSC can effectively recover the underlying subspaces, confirming that D_t converges to the union of subspaces. For the two online algorithms OLRSC and ORPCA, We compute the EV after revealing all the samples. We examine the performance under different intrinsic dimension d_k's and corruption ρ. To be more detailed, the d_k's are varied from $0.01p$ to $0.1p$ with a step size $0.01p$, and the ρ is from 0 to 0.5, with a step size 0.05.

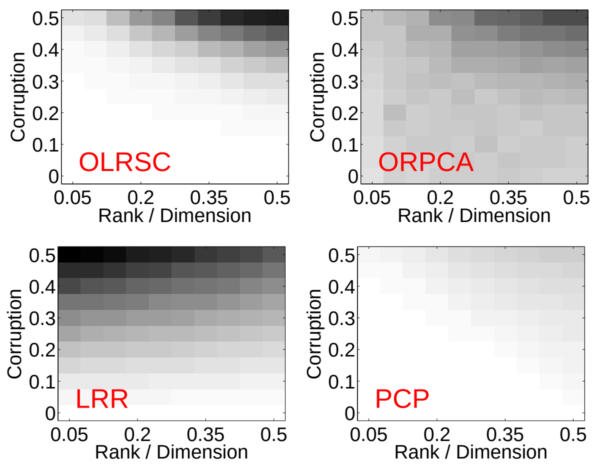

Figure 1. **Subspace recovery under different intrinsic dimensions and corruptions.** Brighter is better. We set $p = 100$, $n_k = 1000$ and $d = 4d_k$. LRR and PCP are batch methods. OLRSC consistently outperforms ORPCA and even improves the performance of LRR. Compared to PCP, OLRSC is competitive in most cases and degrades a little for highly corrupted data, possibly due to the number of samples is not sufficient for its convergence.

The results are presented in Figure 1. The most intriguing observation is that OLRSC as an online algorithm outperforms its batch counterpart LRR! Such improvement may come from the explicit modeling for the basis, which makes OLRSC more informative than LRR. To fully understand the rationale behind this phenomenon is an important direction for future research. Notably, OLRSC consistently beats ORPCA (an online version of PCP), in that OLRSC takes into account that the data are produced by a union of small subspaces. While PCP works well for almost all scenarios, OLRSC degrades a little when addressing difficult cases (high rank and corruption). This is not surprising since Theorem 1 is based on asymptotic analysis and hence, we expect that OLRSC will converge to the true subspace after acquiring more samples.

Convergence Rate. Now we test on a large dataset to show that our algorithm usually converges to the true subspace faster than ORPCA. We plot the EV curve against the number of samples in Figure 2. Firstly, when equipped with a proper matrix Y, OLRSC2 and LRR2 can always produce an exact recovery of the subspace as PCP does. When using the dataset itself for Y, OLRSC still converges to a favorable point after revealing all the samples. Compared to ORPCA, OLRSC is more robust and converges much faster for hard cases (see, e.g., $\rho = 0.5$). Again, we note that in such hard cases, OLRSC outperforms LRR, which agrees with the observation in Figure 1.

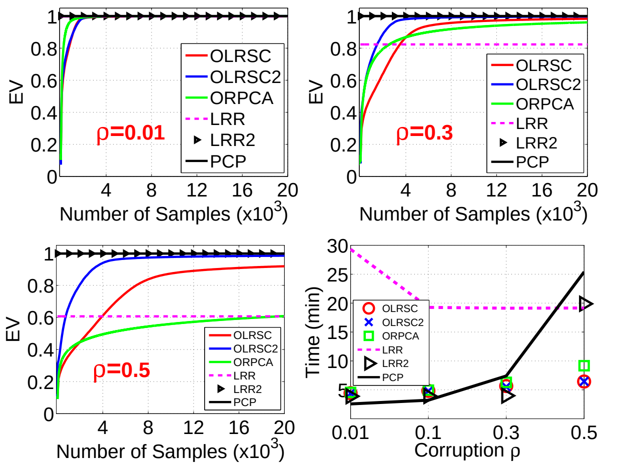

Figure 2. **Convergence rate and time complexity.** A higher EV means better subspace recovery. We set $p = 1000$, $n_k = 5000$, $d_k = 25$ and $d = 100$. OLRSC always converges to or outperforms the batch counterpart LRR. For hard cases, OLRSC converges much faster than ORPCA. Both PCP and LRR2 achieve the best EV value. When equipped with the same dictionary as LRR2, OLRSC2 also well handles the highly corrupted data ($\rho = 0.5$). Our methods are more efficient than the competitors but PCP when ρ is small, possibly because PCP utilizes a highly optimized C++ toolkit while ours are written in Matlab.

Computational Efficiency. We also illustrate the time complexity of the algorithms in the last panel of Figure 2. In short, our algorithms (OLRSC and OLRSC2) admit the lowest computational complexity for all cases. One may argue that PCP spends slightly less time than ours for a small ρ (0.01 and 0.1). However, we remark here that PCP utilizes a highly optimized C++ toolkit to boost computation while our algorithms are fully written in Matlab. We believe that ours will work more efficiently if properly optimized by, e.g., the blas routine. Another important message conveyed by the figure is that, OLRSC is always being orders of magnitude computationally more efficient than the batch method LRR, as well as producing comparable or even better solution.

5.2. Subspace Clustering

Datasets. We examine the performance for subspace clustering on 5 realistic databases shown in Table 1, which can be downloaded from the LibSVM website. For MNIST, We randomly select 20000 samples to form MNIST-20K since we find it time consuming to run the batch methods on the entire database.

Table 1. **Datasets for subspace clustering.**

	#classes	#samples	#features
Mushrooms	2	8124	112
DNA	3	3186	180
Protein	3	24,387	357
USPS	10	9298	256
MNIST-20K	10	20,000	784

Standard Clustering Pipeline. In order to focus on the solution quality of different algorithms, we follow the standard pipeline which feeds X to a spectral clustering algorithm (Ng et al., 2001). To this end, we collect all the $\boldsymbol{u}$'s and $\boldsymbol{v}$'s produced by OLRSC to form the representation matrix $X = UV^\top$. For ORPCA, we use $R_0 R_0^\top$ as the similarity matrix (Liu et al., 2013), where R_0 is the row space of $Z_0 = L_0 \Sigma_0 R_0^\top$ and Z_0 is the clean matrix recovered by ORPCA. We run our algorithm and ORPCA with 2 epochs so as to apply backward correction on the coefficients (U and V in ours and R_0 in ORPCA).

Fully Online Pipeline. As we discussed in Section 3, the (optional) spectral clustering procedure needs the similarity matrix X, making the memory proportional to n^2. To tackle this issue, we proposed a fully online scheme where the key idea is performing k-means on V. Here, we examine the efficacy of this variant, which is called OLRSC-F.

The results are recorded in Table 2, where the time cost of spectral clustering or k-means is not included so we can focus on comparing the efficiency of the algorithms themselves. Also note that we use the dataset itself as the dictionary Y because we find that an alternative choice of Y does not help much on this task. For OLRSC and ORPCA, they require an estimation on the true rank. Here, we use $5k$ as such estimation where k is the number of classes of a dataset. Our algorithm significantly outperforms the two state-of-the-art methods LRR and SSC both for accuracy and efficiency. One may argue that SSC is slightly better than OLRSC on Protein. Yet, it spends 1 hour while OLRSC only costs 25 seconds. Hence, SSC is not practical. Compared to ORPCA, OLRSC always identifies more

Table 2. **Clustering accuracy (%) and computational time (seconds).** For each dataset, the first row indicates the accuracy and the second row the running time. For all the large-scale datasets, OLRSC (or OLRSC-F) has the highest clustering accuracy. Regarding the running time, our method spends comparable time as ORPCA (the fastest solver) does while dramatically improves the accuracy. Although SSC is slightly better than SSC on Protein, it consumes one hour while OLRSC takes 25 seconds.

	OLRSC	OLRSC-F	ORPCA	LRR	SSC
Mush-rooms	85.09	**89.36**	65.26	58.44	54.16
	8.78	8.78	8.30	46.82	32 min
DNA	67.11	**83.08**	53.11	44.01	52.23
	2.58	2.58	2.09	23.67	3 min
Protein	43.30	43.94	40.22	40.31	**44.27**
	24.66	24.66	22.90	921.58	65 min
USPS	65.95	**70.29**	55.70	52.98	47.58
	33.93	33.93	27.01	257.25	50 min
MNIST-20K	**57.74**	55.50	54.10	55.23	43.91
	129	129	121	32 min	7 hours

correct samples as well as consumes comparable running time. For example, on the USPS dataset, OLRSC achieves the accuracy of 65.95% while that of ORPCA is 55.7%. Regarding the running time, OLRSC uses only 7 seconds more than ORPCA – same order of computational complexity, which agrees with the qualitative analysis in Section 3 and the one in Feng et al. (2013).

More interestingly, it shows that the k-means alternative (OLRSC-F) usually outperforms the spectral clustering pipeline. This suggests that perhaps for *robust* subspace clustering formulations, the simple k-means paradigm suffices to guarantee an appealing result. On the other hand, we report the running time of spectral clustering and k-means in Table 3. As expected, since spectral clustering computes SVD for an n-by-n similarity matrix, it is quite slow. In fact, it sometimes dominates the running time of the whole pipeline. In contrast, k-means is extremely fast and scalable, as it can be implemented in online fashion.

Table 3. **Time cost (seconds) of spectral clustering and k-means.** We use the shorthand "Mush" for Mushrooms dataset and "M-20K" for MNIST-20K for a better view.

	Mush	DNA	Protein	USPS	M-20K
Spectral	295	18	7567	482	4402
k-means	2	6	5	19	91

5.3. Influence of d

A key ingredient of our formulation is a factorization on the nuclear norm regularized matrix, which requires an estimation on the rank of the X (see (2.1)). Here we examine the influence of the selection of d (which plays as an upper

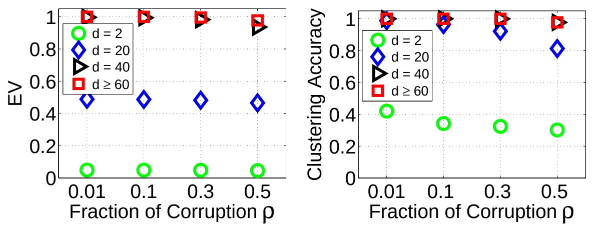

Figure 3. **Examine the influence of d.** We experiment on $d = \{2, 20, 40, 60, 80, 100, 120, 140, 160, 180\}$. The true rank is 40.

bound of the true rank). We report both EV and clustering accuracy for different d under a range of corruptions. The simulation data are generated as in Section 5.1 and we set $p = 200$, $n_k = 1000$ and $d_k = 10$. Since the four subspaces are disjoint, the true rank is 40.

From Figure 3, we observe that our algorithm cannot recover the true subspace if d is smaller than the true rank. On the other hand, when d is sufficiently large (at least larger than the true rank), our algorithm can perfectly estimate the subspace. This agrees with the results in Burer & Monteiro (2005) which says as long as d is large enough, any local minima is global optima. We also illustrate the influence of d on subspace clustering. Generally speaking, OLRSC can consistently identify the cluster of the data points if d is sufficiently large. Interestingly, different from the subspace recovery task, here the requirement for d seems to be slightly relaxed. In particular, we notice that if we pick d as 20 (smaller than the true rank), OLRSC still performs well. Such relaxed requirement of d may benefit from the fact that the spectral clustering step can correct some wrong points as suggested by Soltanolkotabi et al. (2014).

6. Conclusion

In this paper, we have proposed an online algorithm termed OLRSC for subspace clustering, which dramatically reduces the memory cost of LRR from $\mathcal{O}(n^2)$ to $\mathcal{O}(pd)$. One of the key techniques is an explicit basis modeling, which essentially renders the model more informative than LRR. Another important component is a non-convex reformulation of the nuclear norm. Combining these techniques allows OLRSC to simultaneously recover the union of the subspaces, identify the possible corruptions and perform subspace clustering. We have also established the theoretical guarantee that solutions produced by our algorithm converge to a stationary point of the expected loss function. Moreover, we have analyzed the time complexity and empirically demonstrated that our algorithm is computationally very efficient compared to competing baselines. Our extensive experimental study on synthetic and realistic datasets also illustrates the robustness of OLRSC. In a nutshell, OLRSC is an appealing algorithm in all three worlds: memory cost, computation and robustness.

Acknowledgments

J. Shen and P. Li are partially supported by NSF-Bigdata-1419210, NSF-III-1360971 and AFOSR-FA9550-13-1-0137. The research of H. Xu is partially supported by Agency for Science, Technology and Research (A*STAR) of Singapore through SERC PSF Grant R266000101305.

References

Avron, Haim, Kale, Satyen, Kasiviswanathan, Shiva Prasad, and Sindhwani, Vikas. Efficient and practical stochastic subgradient descent for nuclear norm regularization. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

Bertsekas, Dimitri P. *Nonlinear programming*. Athena Scientific, 1999.

Bonnans, J. Frédéric and Shapiro, Alexander. Optimization problems with perturbations: A guided tour. *SIAM Review*, 40(2):228–264, 1998.

Bottou, Léon. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9), 1998.

Burer, Samuel and Monteiro, Renato D. C. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, 2005.

Candès, Emmanuel J. and Recht, Benjamin. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

Candès, Emmanuel J., Li, Xiaodong, Ma, Yi, and Wright, John. Robust principal component analysis? *Journal of the ACM*, 58(3):11, 2011.

Defazio, Aaron, Bach, Francis R., and Lacoste-Julien, Simon. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Neural Information Processing Systems*, pp. 1646–1654, 2014.

Elhamifar, Ehsan and Vidal, René. Sparse subspace clustering. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2790–2797, 2009.

Eriksson, Brian, Balzano, Laura, and Nowak, Robert D. High-rank matrix completion and subspace clustering with missing data. *CoRR*, abs/1112.5629, 2011.

Fazel, Maryam, Hindi, Haitham, and Boyd, Stephen P. A rank minimization heuristic with application to minimum order system approximation. In *Proceedings of the American Control Conference*, volume 6, pp. 4734–4739. IEEE, 2001.

Feng, Jiashi, Xu, Huan, and Yan, Shuicheng. Online robust PCA via stochastic optimization. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*, pp. 404–412, 2013.

Hsieh, Cho-Jui and Olsen, Peder A. Nuclear norm minimization via active subspace selection. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 575–583, 2014.

Jaggi, Martin and Sulovský, Marek. A simple algorithm for nuclear norm regularized problems. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 471–478, 2010.

Lin, Zhouchen, Chen, Minming, and Ma, Yi. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

Liu, Guangcan and Li, Ping. Recovery of coherent data via low-rank dictionary pursuit. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems*, pp. 1206–1214, 2014.

Liu, Guangcan, Lin, Zhouchen, Yan, Shuicheng, Sun, Ju, Yu, Yong, and Ma, Yi. Robust recovery of subspace structures by low-rank representation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(1):171–184, 2013.

Mairal, Julien, Bach, Francis R., Ponce, Jean, and Sapiro, Guillermo. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

Mardani, Morteza, Mateos, Gonzalo, and Giannakis, Georgios B. Subspace learning and imputation for streaming big data matrices and tensors. *IEEE Trans. Signal Processing*, 63(10):2663–2677, 2015.

Ng, Andrew Y., Jordan, Michael I., and Weiss, Yair. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 15th Annual Conference on Neural Information Processing Systems*, pp. 849–856, 2001.

Recht, Benjamin, Fazel, Maryam, and Parrilo, Pablo A. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

Richtárik, Peter and Takác, Martin. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

Shen, Jie and Li, Ping. Learning structured low-rank representation via matrix factorization. In *Proceedings of the*

19th International Conference on Artificial Intelligence and Statistics, pp. 500–509, 2016.

Soltanolkotabi, Mahdi and Candès, Emmanuel J. A geometric analysis of subspace clustering with outliers. *The Annals of Statistics*, 40:2195–2238, 2012.

Soltanolkotabi, Mahdi, Elhamifar, Ehsan, and Candès, Emmanuel J. Robust subspace clustering. *The Annals of Statistics*, 42(2):669–699, 2014.

van der Vaart, A.W. *Asymptotic statistics*. Cambridge University Press, 2000.

Vidal, René. A tutorial on subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2010.

Wang, Yu-Xiang, Xu, Huan, and Leng, Chenlei. Provable subspace clustering: When LRR meets SSC. In *Proceedings of 27th Annual Conference on Neural Information Processing Systems*, pp. 64–72, 2013.

Xiao, Lin and Zhang, Tong. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Xu, Huan, Caramanis, Constantine, and Mannor, Shie. Principal component analysis with contaminated data: The high dimensional case. In *Proceedings of the 23rd Conference on Learning Theory*, pp. 490–502, 2010.