Robust Learning from Noisy Side-information by Semidefinite Programming

En-Liang Hu[1] , **Quanming Yao**[2,3]
[1]Department of Mathematic, Yunnan Normal University
[2]4Paradigm Inc
[3]Department of Computer Science and Engineering, Hong Kong University of Science and Technology
ynel.hu@gmail.com, yaoquanming@4paradigm.com

Abstract

Robustness recently becomes one of the major concerns among machine learning community, since learning algorithms are usually vulnerable to outliers or corruptions. Motivated by such trend and needs, we pursue robustness in semi-definite programming (SDP) in this paper. Specifically, this is done by replacing the commonly used squared loss with the more robust ℓ_1-loss in the low-rank SDP. However, the resulting objective becomes neither convex nor smooth. As no existing algorithms can be applied, we design an efficient algorithm, based on majorization-minimization, to optimize the objective. The proposed algorithm not only has cheap iterations and low space complexity, but also theoretically converges to some critical points. Finally, empirical study shows that the new objective armed with proposed algorithm outperforms state-of-the-arts in terms of both speed and accuracy. [1]

1 Introduction

Semidefinite programming (SDP) studies optimization problems with a convex objective function over semidefinite constraints [Vandenberghe and Boyd, 1996; Boyd and Vandenberghe, 2004]. Many machine learning problems can be reduced as SDPs [Lemon *et al.*, 2016]. Prominent examples include embedding and clustering [Kulis *et al.*, 2007; Royer, 2017], sparse PCA [D'aspremont *et al.*, 2007; Zou and Xue, 2018], maximum variance unfolding (MVU) [Weinberger *et al.*, 2004; Song *et al.*, 2008], non-parametric kernel learning (NPKL) [Li *et al.*, 2008; Zhuang *et al.*, 2011]. Generally, the SDP optimization problem is formulated as

$$\min_{Z \in \mathbb{S}_+} \mathcal{F}(Z) \equiv \sum_{\tau=1}^{m} \frac{1}{2} \left(\text{tr}(ZQ_\tau) - t_\tau\right)^2 + \frac{\gamma}{2}\text{tr}(ZA), \quad (1)$$

where $\{Q_\tau, t_\tau\}_{\tau=1}^m$ comes from the training data (such as side information), A is a symmetric matrix to regularize Z (depending on applications), $\mathbb{S}_+$ is the cone of positive semi-definite (PSD) matrices, and $\gamma > 0$ is a hyper-parameter.

[1]Quanming Yao is the corresponding author.

The PSD constraint, i.e., $Z \in \mathbb{S}_+$, is the most challenging part in solving (1) [Lemon *et al.*, 2016]. For example, it costs $\mathcal{O}(n^3)$ time at each iteration using the interior point algorithm [Helmberg *et al.*, 1996] if Z is of size $n \times n$. Another example is the projection gradient descent algorithm, in which the projection to PSD cone will also cost $\mathcal{O}(n^3)$ time [Jaggi, 2013]. To drop the PSD constraint, there is an (efficient) matrix factorization method to go about fitting a low rank model [Burer and Monteiro, 2003; Lemon *et al.*, 2016]. Namely, factorizing Z to $XX^\top$, then (1) can be converted into as below

$$\min_{X \in \mathbb{R}^{n\times r}} \sum_{\tau=1}^{m} \frac{1}{2}(\text{tr}(X^\top Q_\tau X) - t_\tau)^2 + \frac{\gamma}{2}\text{tr}(X^\top AX). \quad (2)$$

While the problem is nonconvex, instead of optimizing w.r.t. $Z \in \mathbb{S}_+$, we only need to solve an unconstrained optimization problem with variable $X \in \mathbb{R}^{n\times r}$. Moreover, it is theoretically shown that the factorized problem (2) is equivalent to (1) when the rank of solution is deficient [Srinadh *et al.*, 2016; Zheng and Lafferty, 2015].

Many algorithms have been proposed to solve (2) and are all much more efficient than interior method and projection gradient descent for (1). When the objective $\mathcal{F}$ is linear, L-BFGS is introduced in [Burer and Monteiro, 2003; Nocedal and Wright, 2006] for optimization. However, the convergence properties of L-BFGS are unclear for the nonconvex problem here. When $\mathcal{F}$ is convex and smooth, block-cyclic coordinate minimization has been used in [Hu *et al.*, 2011] to solve a special nonconvex program of SDP, but a closed-form solution is preferred in each block coordinate update, which might be overly restrictive. More recently, gradient descent based methods [Srinadh *et al.*, 2016; Zheng and Lafferty, 2015] have been developed as the state-of-the-art for low-rank SDP. These algorithms have a convergence guarantee, and linear/sub-linear convergence rate are also established for some low-rank SDP formulations [Srinadh *et al.*, 2016; Pumir *et al.*, 2018].

In above applications, the squared loss is used in $\mathcal{F}$ to encourage the learned Z to be consistent with give side-information. However, since the squared loss is sensitive to outliers, all existing SDP algorithms are not robust. Moreover, robustness is of a real demand. The side-information utilized in SDP may not be accurate for real applications, e.g., samples can corrupted in MVU [Dekel *et al.*, 2010] and links collected for kernel learning can come from spammer or attacker [Raykar *et al.*, 2010]. Such corruptions and noise can significantly deteriorate performance of learning models [Raykar *et al.*, 2010].

Motivated by the success of making matrix factorization robust by replacing the squared loss with ℓ_1 loss [Lin *et al.*, 2017; Yao and Kwok, 2018], we also proposed to use the ℓ_1 loss in (2) for SDP, and illustrate such need with three applications, i.e., robust NPKL, robust colored MVU and sparse PCA. However, the resulting optimization problem is neither convex nor smooth, and none of existing SDP algorithms can be used for optimization. To solve the new objective, we propose a new optimization algorithm based on Majorization-Minimization (MM), of which the crux is constructing a good surrogate. Besides, while MM generally only guarantees producing limit points, we prove that by iteratively optimizing the constructed surrogate, the proposed algorithm ensures a convergence to some critical points. Finally, we demonstrate the efficiency and robustness of the proposed algorithm using above three applications. Results show that the proposed algorithm is not only faster but also better in recovery over state-of-the-arts. As a summary, we hight-light our contributions as follows:

- We are the first to introduce the robust loss, i.e., ℓ_1 loss, into SDP. To show its necessity, we further illustrate the usage of the new objective with three applications, i.e., robust NPKL, robust colored MVU and sparse PCA;
- As no existing SDP algorithms can be applied, we propose a novel optimization algorithm to solve the new objective, which is not only efficient but also guaranteed converging to some critical points;
- Finally, the robustness of the new objective and the effectiveness of the proposed algorithm over state-of-the-arts SDP algorithms are empirically verified on above three applications.

Notation

We use an uppercase letter to indicate a matrix, and a lowercase letters to a scalar. The transpose of vector or matrix is denoted by the superscript $(\cdot)^\top$. The identity matrix is denoted by I; for a matrix $A = [a_{ij}]$, $\text{tr}(A)$ is the trace of a square matrix, $\|A\|_F = (\sum_{ij} a_{ij}^2)^{1/2}$ is its Frobenius norm. $|a|$ is the absolute of scaler a, and $|\mathcal{S}|$ is cardinal number of set $\mathcal{S}$.

2 Robust Semidefinite Programming

Most SDP learning algorithms assume perfect side information (i.e., perfect triplet constraints in our case). This however is not always the case in practice because in many real-world applications, the constraints are derived from the side information such as users implicit feedbacks and citations among articles. As a result, these constraints are usually noisy and consist of many mistakes. We refer to *the problem of learning SDP matrix from noisy side information as robust SDP learning*. Feeding the noisy constraints directly into a SDP learning algorithm will inevitably degrade its performance, and more seriously.

2.1 Proposed Formulation

Inspired by the recent success of using ℓ_1-loss instead of the squared loss in making matrix factorization robust [Lin *et al.*, 2017; Yao and Kwok, 2018], we also propose to replace the squared loss in (1) by the more robust ℓ_1 loss, which makes low-rank SDP less sensitive to corruptions in the training data. This leads to the objective of robust SDP as

$$\min_X \mathcal{R}(X) \equiv \sum_{\tau=1}^{m} \left|\text{tr}(X^\top Q_\tau X) - t_\tau\right| + \frac{\gamma}{2}\text{tr}(X^\top A X) + \frac{\lambda}{2}\|X\|^2 \quad (3)$$

where $\lambda > 0$ and the last term is to further prevent overfitting. This new objective is neither convex nor smooth. As a result, none of existing algorithms for low-rank SDP, e.g., L-BFGS [Burer and Monteiro, 2003], gradient descent [Srinadh *et al.*, 2016; Zheng and Lafferty, 2015], and coordinate descent [Hu *et al.*, 2011], can be applied. In next Section, we will design an efficient algorithm for (3) based on MM, which also has a convergence guarantee.

2.2 Application Examples

However, before that, we illustrate the usage and importance of the new formulation using three examples.

Example 1: Robust NPKL

Given n patterns, let $\mathcal{M}$ be the must-link set containing pairs that should belong to the same class, and $\mathcal{C}$ be the cannot-link set containing pairs that should not belong to the same class. Denote $\mathcal{T} = \mathcal{M} \cup \mathcal{C}$. Non-parametric kernel learning (NPKL) [Hoi *et al.*, 2007] tries to build a kernel matrix utilizing above side information. We adopt the formulation in [Li *et al.*, 2008; Zhuang *et al.*, 2011], which learns a kernel matrix using the following SDP problem

$$\min_{Z \in \mathbb{S}^+} \sum_{\tau=1}^{|\mathcal{T}|} \left(\text{tr}\left(Z Q_\tau\right) - t_\tau\right)^2 + \frac{\gamma}{2}\text{tr}(ZL), \quad (4)$$

where Z is the target kernel matrix, L is the graph Laplacian matrix of the data, $\mathcal{M}$ and $\mathcal{C}$ are encoded into $\{(Q_\tau, t_\tau)\}_{\tau=1}^{|\mathcal{T}|}$. Let $Q_\tau = I(:,j)\left(I(:,i)\right)^\top$, then $\text{tr}(Q_\tau Z) = Z_{ij}$. Thus $t_\tau = 1$ if $(i,j) \in \mathcal{M}$ and 0 if $(i,j) \in \mathcal{C}$. The first term of objective in (4) measures the difference between Z_{ij} and t_τ, and the second term $\text{tr}(ZL)$ encourages smoothness on the data manifold by aligning Z with L.

The side information in this application is those "can" and "cannot" links. These links are usually provided by humans, e.g., labeled by experts or crowdsourced from the web. As human may not be reliable and there can be spammers and attackers in the crowdsourcing platform, errors and noise can exist in these links [Raykar *et al.*, 2010]. These again inspire a more robust formulation of NPKL as

$$\min_X \sum_{\tau=1}^{|\mathcal{T}|} \left|\text{tr}(X^\top Q_\tau X) - t_\tau\right| + \frac{\gamma}{2}\text{tr}(X^\top L X) + \frac{\lambda}{2}\|X\|^2.$$

Examples 2: Robust CMVU

Maximum variance unfolding (MVU) [Weinberger *et al.*, 2004; Weinberger and Saul, 2006] is an effective method for dimensionality reduction. It produces a low-dimensional representation of the data by simultaneously maximizing the variance of their embeddings and preserving the local distances of the original data. MVU can be viewed as a non-linear generalization of principal component analysis. The colored maximum variance unfolding (CMVU) is a "colored" variants of MVU [Song *et al.*, 2008], subjected to class labels information. In [Song *et al.*, 2008], it is formulated as a low-rank SDP problem as

$$\min_{Z\in\mathbb{S}^+}\sum_{\tau=1}^{|\mathcal{N}|}(\text{tr}(ZQ_\tau)-d_\tau)^2-\frac{\gamma}{2}\text{tr}(ZHTH), \tag{5}$$

where $E_{ij}=\text{I}(:,i)-\text{I}(:,j)$, $Q_\tau=E_{ij}E_{ij}^\top$, $d_\tau=d_{ij}$ denotes the Euclidean distance between the i-th and j-th objects in primal space, $\mathcal{N}$ denotes the set of neighbor pairs, whose distances are to be preserved in the embedding, T is a kernel matrix of the labels, $H_{ij}=\delta_{ij}-\frac{1}{n}$ centers the data and the labels in the feature space, and λ controls the tradeoff between dependence maximization and distance preservation.

In this example, the side information is the local distance d_τ from the original data. However, during the data collection, outliers or corrupted samples can be introduced into feature space [Dekel *et al.*, 2010]. This motivates our robust formulation as:

$$\min_X\sum_{\tau=1}^{|\mathcal{N}|}\left|\text{tr}(X^\top Q_\tau X)-d_\tau\right|-\frac{\gamma}{2}\text{tr}(X^\top HTHX)+\frac{\lambda}{2}\|X\|^2.$$

Example 3: Sparse PCA

Finally, for the last example, we consider sparse PCA [Zou and Xue, 2018]. Here, we are not making sparse PCA more robust, but show sparse PCA also fall into our objective (3), and thus can be solved with proposed algorithm. For a given covariance matrix $\Sigma\in\mathbb{R}^{n\times n}$, sparse PCA tries to find a sparse vector x that maximizes $x^\top\Sigma x$, i.e. a sparse principal component of Σ. Following [D'aspremont *et al.*, 2007], sparse PCA can be relaxed into SDP problem as:

$$\min_{Z\in\mathbb{S}^+,\text{tr}(Z)=1}-\text{tr}(Z\Sigma). \tag{6}$$

We propose to factorize $Z=XX^\top$, and solve the following approximated objective instead

$$\min_X\sum_{\tau=1}^{\frac{n(n+1)}{2}}\left|\text{tr}(X^\top Q_\tau X)-0\right|-\frac{\gamma}{2}\text{tr}(X^\top\Sigma X)+\frac{\lambda}{2}\|X\|^2.$$

where $Q_\tau=I(:,j)\,(I(:,i))^\top$, $i=1,\cdots n, j=i,\cdots,n$.

3 Optimization Algorithm

Majorization minimization (MM) is a general technique to make difficult optimization problems easier [Hunter and Lange, 2004; Lange *et al.*, 2000]. Recently, it has been applied in robust matrix factorization (RMF) [Lin *et al.*, 2017; Yao and Kwok, 2018]. Inspired by such success and the fact that no existing SDP algorithms can be applied here, we give a MM algorithm to solve (3) in the sequel.

3.1 Majorization-Minimization (MM)

Consider a function $g(X)$, which is hard to optimize. Let the iterate at the kth MM iteration be X_k. The next iterate is generated as

$$X_{k+1}=X_k+\arg\min_{\tilde{X}}h^k(\tilde{X};X_k), \tag{7}$$

where h^k is a surrogate that is being optimized instead of g. A good surrogate should have the following properties [Lange *et al.*, 2000]:

(a). $g(\tilde{X}+X_k)\le h^k(\tilde{X};X_k)$ for any $\tilde{X}$;

(b). $0\in\arg\min_{\tilde{X}}(h^k(\tilde{X};X_k)-g(\tilde{X}+X_k))$ and $g(X_k)=h^k(0;X_k)$; and

(c). h^k is convex on $\tilde{X}$.

The first two conditions (a) and (b) ensure $\{g(X_k)\}$ generated from MM is a non-increasing sequence, and (c) encourages sub-problems $h^k(\tilde{X};X_k)$ can be easily solved.

However, MM only guarantees that the objectives obtained in successive iterations are non-increasing, but does not guarantee convergence of the sequence $\{X_k\}$ [Hunter and Lange, 2004; Lange *et al.*, 2000; Lin *et al.*, 2017].

3.2 Constructing the Convex Surrogate

Here, we show how a surrogate can be constructed from $\mathcal{R}$, which can meet the above three conditions of MM. First, we upper bound $\mathcal{R}$ (see (3)) in following Lemma 1 based on (7).

Lemma 1. *Let $\dot{A}=A+\frac{\lambda}{\gamma}I$, for any $\tilde{X}\in\mathbb{R}^{n\times r}$ we have* $\mathcal{R}(\tilde{X}+X_k)\le\sum_{\tau=1}^m\left|\text{tr}(2\tilde{X}^\top Q_\tau X_k+X_k^\top Q_\tau X_k)-t_\tau\right|+\sum_{\tau=1}^m\left|\text{tr}(\tilde{X}^\top Q_\tau\tilde{X})\right|+\frac{\gamma}{2}\text{tr}\left(\tilde{X}^\top\dot{A}\tilde{X}+(X_k+2\tilde{X})^\top\dot{A}X_k\right)$.

However, the upper bound in Lemma 1 is not convex, as the term $|\text{tr}(\tilde{X}^\top Q_\tau\tilde{X})|$ is convex only when $Q_\tau\in\mathbb{S}_+$ [Boyd and Vandenberghe, 2004], which is not guaranteed. Let (γ_i,v_i)'s be the eigen-pairs of a symmetric square matrix M, we use $(\cdot)_+$ and $(\cdot)_-$ to denote positive and negative eigen values of M, i.e., $M_+=\sum_i\max(\gamma_i,0)v_iv_i^\top$ and $M_-=\sum_i\min(\gamma_i,0)v_iv_i^\top$, thus $M=M_++M_-$. To address this issue, we make use of following Lemma 2.

Lemma 2. $|\text{tr}(\tilde{X}^\top Q_\tau\tilde{X})|\le\text{tr}(\tilde{X}^\top\bar{Q}_\tau\tilde{X})$ *where* $\bar{Q}_\tau=\frac{1}{2}(Q_\tau+Q_\tau^\top)_+-\frac{1}{2}(Q_\tau+Q_\tau^\top)_-$.

Combining Lemma 1 and 2, a convex surrogate is constructed as follow:

Proposition 1. *Let $B=Q+\frac{1}{2}(\lambda I+\gamma A_+)$, $C=A+\frac{\lambda}{\gamma}I$, $Q=\sum_{\tau=1}^m\bar{Q}_\tau$, $b_\tau^k=\frac{1}{2}(\text{tr}(X_k{}^\top Q_\tau X_k)-t_\tau)$, and $c_k=\frac{\gamma}{2}\text{tr}(X_k{}^\top(A+\frac{\lambda}{\gamma}I)X_k)$, then $\mathcal{R}(\tilde{X}+X_k)\le\mathcal{H}^k(\tilde{X},X_k)$ where*

$$\begin{aligned}\mathcal{H}^k(\tilde{X},X_k)=&\text{tr}(\tilde{X}^\top(B\tilde{X}+\gamma CX_k))\\&+2\sum_{\tau=1}^m\left|\text{tr}(\tilde{X}^\top Q_\tau X_k)+b_\tau^k\right|+c_k,\end{aligned}$$

and the equality holds iff $\tilde{X}=\mathbf{0}$.

		model		complexity	
		factorized	loss	space	iteration
FW [Laue, 2012]		×	squared loss	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
L-BFGS [Nocedal and Wright, 2006]		$\surd$	squared loss	$\mathcal{O}(nr)$	$\mathcal{O}(nr^2)$
nmAPG [Li and Lin, 2015]		$\surd$	squared loss	$\mathcal{O}(nr)$	$\mathcal{O}(nr^2)$
SADMM [Boyd *et al.*, 2011]		×	ℓ_1 loss	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2r)$
SDPLR [Burer and Monteiro, 2003]		$\surd$	ℓ_1 loss	$\mathcal{O}(n^2)$	$\mathcal{O}(nr^2)$
SDPNAL [Toh *et al.*, 2015]		×	ℓ_1 loss	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
RSDP	ADMM	$\surd$	ℓ_1 loss	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
	APG	$\surd$	ℓ_1 loss	$\mathcal{O}(\text{nnz}(E){+}nr)$	$\mathcal{O}(\text{nnz}(E){+}nr^2)$

Table 1: Comparison of exemplar existing algorithms with the proposed one. The space and iteration complexity are derived based on the example of robust NPKL; $E = \sum_{\tau=1}^m Q_\tau + C$ and "nnz" denotes the number of nonzero elements in a matrix.

Obviously, $\mathcal{H}^k(\tilde{X}, X)$ is convex w.r.t. $\tilde{X}$, so it is a convex surrogate of $\mathcal{R}$. Besides, from Proposition 1, it can also been seen that $\mathcal{R}(\tilde{X} + X_k) \le \mathcal{H}^k(\tilde{X}; X_k)$ for any $\tilde{X}$, $0 = \arg\min_{\tilde{X}}(\mathcal{H}^k(\tilde{X}; X_k) - \mathcal{R}(\tilde{X} + X_k))$ and $\mathcal{R}(X_k) = \mathcal{H}^k(0; X_k)$. Thus, all three desired properties on surrogate of MM (Section 3.1) are satisfied.

Remark 1. *MM algorithm, recently, has also been considered in RMF-MM [Lin* et al.*, 2017] and RMFNL [Yao and Kwok, 2018] for RMF. While our algorithm is also based on MM and adopt matrix factorization, since our objective comes from low-rank SDP, the way to construct the surrogate is significantly different. Specifically, in [Lin* et al.*, 2017; Yao and Kwok, 2018], Z is factorized as $XY^\top$, X and Y are bounded separately; we need to find other ways to bound X here, which is enabled by Lemma 2.*

3.3 Solving the Surrogate

According to the framework of MM algorithm, i.e., (7), at each iteration we need to update X_k by $X_{k+1} = X_k + \arg\min_{\tilde{X}} \mathcal{H}^k(\tilde{X}, X_k)$. Previously, ADMM (Alternating Direction Method of Multipliers) [Boyd *et al.*, 2011] is used in [Lin *et al.*, 2017] for RMF, and APG (Accelerated Proximal Gradient) [Beck and Teboulle, 2009] is later proposed in [Yao and Kwok, 2018] to further explore data sparsity in RMF. Here, we show both ADMM and APG can still be applied to solve the optimization problem $\arg\min_{\tilde{X}} \mathcal{H}^k(\tilde{X}, X_k)$. In Table 1, we can see APG can need less space than ADMM when E is sparse.

Using ADMM

Since terms in the ℓ_1 loss is complex, we reformulate $\mathcal{H}^k$ as

$$\min_{\tilde{X}} \; \text{tr}\left(\tilde{X}^\top(B\tilde{X} + \gamma C X)\right) + 2\sum_{\tau=1}^m |e_\tau|, \qquad (8)$$
$$\text{s.t.} \;\; e_\tau = \text{tr}(\tilde{X}^\top Q_\tau X) + b_\tau^k.$$

Then, we can introduce dual parameter p_τ for each linear constraint, and use ADMM algorithm [Boyd *et al.*, 2011; Lin *et al.*, 2017] to solve the augmented Lagrangian of (8). It can be easily checked that updates of $\tilde{X}$, e_τ and p_τ have closed-form solutions.

Using APG

In the other way, using $\|x\|_1 = \inf_z x^\top z : \|z\|_\infty \le 1$ [Boyd and Vandenberghe, 2004]. We can derive the dual form $\mathcal{D}^k$ of $\mathcal{H}^k$ from

$$\max_{|z_\tau|\le 1} \min_{\tilde{X}} \tilde{X}^\top(B\tilde{X} + \gamma C X_k) + 2\sum_{\tau=1}^m z_\tau(\text{tr}(\tilde{X}^\top Q_\tau X_k) + b_\tau^k).$$

which is given by

$$\min_{|z_\tau|\le 1} : \mathcal{D}^k(z_\tau) = \text{tr}\left(P(z_\tau)^\top B^{-1} P(z_\tau)\right) - 2\sum_{\tau=1}^m b_\tau^k z_\tau.$$

where $P(z_\tau) = (\sum_{\tau=1}^m z_\tau Q_\tau + \frac{\gamma}{2}C)X_k$. Since the dual problem is a smooth and convex optimization problem with simple box constraints. Thus, as [Yao and Kwok, 2018], we also solve this problem by APG, and recover $\tilde{X} = -B^{-1}(\sum_{\tau=1}^m z_\tau Q_\tau + \frac{\gamma}{2}C)X_k$. Let $Q = \sum_{\tau=1}^m Q_\tau$, it only requires $nnz(Q)$ entries to store dual variables z_τ's when using APG.

3.4 Complete Algorithm

Based on the above analysis, we list the complete steps for solving (3) in Algorithm 1.

Algorithm 1 RSDP: Robust semi-definite programming by majorization-minimization.

```
1: Initialization: $X_1 = 0$.
2: for $k = 1, \dots, K$ do
3:     $\tilde{X}^k = \arg\min_{\tilde{X}} \mathcal{H}(\tilde{X}, X_k)$ via ADMM or APG;
4:     update $X_{k+1} = \tilde{X}^k + X_k$;
5: end for
6: return $X_{K+1}$.
```

The convergence guarantee is in Theorem 1. Note that, as in Section 3.1, MM generally only guarantees the convergence of $\{\mathcal{R}(X_k)\}$ not $\{X_k\}$. Besides, the proofs in RMF-MM and RMFNL cannot be directly applied neither, due to the difference in Remark 1.

Theorem 1. *If $\lim_{\|X\|_F\to\infty} \mathcal{R}(X) = \infty$ and $\inf_X \mathcal{R}(X) > -\infty$, then for Algorithm 1, we have*

(a). there exists a constant $\alpha > 0$ such that $\mathcal{R}(X_k) - \mathcal{R}(X_{k+1}) \ge \frac{\alpha}{2}\|X_{k+1} - X_k\|^2$;

(b). the sequence $\{X_k\}$ is bounded;

(c). any limit points of $\{X_k\}$ are also critical points of $\mathcal{R}$.

An overall comparison of the proposed Algorithm 1 and other algorithms used in Section 4 is summarized in Table 1.

		testing RMSE				CPU time(sec)			
loss	algorithm	Gaussian noise		flipping labels		Gaussian noise		flipping labels	
		5%	10%	5%	10%	5%	10%	5%	10%
squared	FW	0.50±0.01	0.70±0.01	0.31±0.01	0.35±0.01	68.2±3.0	69.2±1.0	47.5±9.9	55.7±6.8
	nmAPG	0.47±0.01	0.63±0.04	0.31±0.01	0.35±0.01	12.1±4.6	15.8±14.5	6.8±1.7	7.6±1.1
	L-BFGS	0.50±0.01	0.69±0.01	0.31±0.01	0.35±0.01	**4.7±0.2**	**4.8±0.3**	**3.6±0.1**	**4.3±0.2**
ℓ_1	SADMM	0.27±0.07	**0.34±0.01**	0.23±0.01	0.29±0.01	886.5±26.6	1002.6±755.9	774.8±247.3	783.6±190.1
	SDPNAL	0.24±0.06	**0.35±0.02**	**0.22±0.02**	**0.28±0.01**	3291.5±73.6	3281.8±258.5	3141.7±412.1	3241.5±322.4
	SDPLR	**0.23±0.01**	**0.34±0.01**	**0.21±0.01**	**0.27±0.01**	3617.9±1.3	3620.6±0.7	3617.4±0.8	3601.8±33.2
	RSDP(ADMM)	**0.23±0.02**	**0.34±0.01**	**0.21±0.01**	**0.28±0.01**	45.9±23.4	117.9±49.6	55.1±33.4	71.9±36.1
	RSDP(APG)	**0.23±0.01**	**0.35±0.01**	**0.21±0.02**	**0.28±0.01**	34.5±4.0	44.6±5.6	48.5±12.4	43.9±3.6

Table 2: Testing RMSEs and CPU time (sec) of various algorithms in the application of robust NPKL.

		testing RMSE			CPU time(sec)		
loss	algorithm	small	large outliers		small	large outliers	
		deviations	5%	10%	deviations	5%	10%
squared	FW	9.16±0.81	12.99±1.39	13.52±2.49	882.4±46.9	786.8±129.9	729.2±37.5
	nmAPG	9.07±0.89	12.14±1.07	12.59±1.84	86.9±17.1	82.3±13.7	82.1±4.5
	L-BFGS	9.08±0.91	12.64±0.99	13.50±2.37	**55.5±7.3**	**65.6±4.7**	**64.6±4.1**
ℓ_1	SADMM	**0.06±0.02**	0.71±0.02	2.43±0.81	1750.4±101.3	333.8±109.6	304.8±18.4
	SDPNAL	**0.06±0.01**	0.86±0.02	**2.06±0.25**	2935.6±347.5	7015.3±455.9	5454.7±398.1
	SDPLR	**0.06±0.01**	0.85±0.02	**2.04±0.21**	1249.3±109.9	3628.3±1.1	3624.9±2.2
	RSDP(ADMM)	**0.06±0.01**	**0.67±0.01**	**2.06±0.26**	238.9±37.8	305.5±65.6	308.7±68.1
	RSDP(APG)	**0.06±0.01**	**0.66±0.01**	**2.06±0.23**	269.6±64.8	240.5±2.7	229.1±13.1

Table 3: Testing RMSEs and CPU time (sec) of various algorithms in the application of robust CMVU.

4 Empirical Study

In this section, we perform experiments on three applications of SDP, namely, robust NPKL (Section 4.1), robust MVU (Section 4.2), and sparse PCA (Section 4.3). These are also three applications we discussed in Section 2.2. The following algorithms based on squared loss will be compared:

1. FW [Laue, 2012]: an application of Frank-Wolf algorithm [Jaggi, 2013] in SDP problem (1);
2. L-BFGS [Nocedal and Wright, 2006]: solve (2) with the most commonly used quasi-Newton solver for smooth minimization problem;
3. nmAPG [Li and Lin, 2015]: an application of state-of-the-art accelerated gradient descent algorithm for problem (2);

The following algorithms based on ℓ_1-loss are compared:

1. SADMM [Boyd *et al.*, 2011]: replace the squared loss in (1) by ℓ_1 loss, and solve the resulting nonsmooth but convex problem with ADMM;
2. SDPLR [Burer and Monteiro, 2003]: solve the same problem as SADMM, but the SDPLR package is used;
3. SDPNAL [Toh *et al.*, 2015]: solve the same problem as SADMM, but SDPNAL package (a newton-CG augmented Lagrangian method) is used;
4. RSDP(ADMM): the proposed Algorithm 1 for the robust objective (3); and ADMM is used as solver for the convex surrogate;
5. RSDP(APG): same as RSDP(ADMM) but APG is used as the solver instead of ADMM.

All algorithm is stopped when the relative change of objective values in successive iterations is smaller than 10^{-5} or when the number of iterations reaches 2000. As for the rank r of initial solution X, in Sections 4.1 and 4.2 we follow [Burer and Monteiro, 2003] and set its value to be the largest r satisfying $r(r+1) \leq m$, where m is the total number of observed data (i.e, m is the number of must-link and cannot-link pairs in Section 4.1, the number of given neighbor pairs in Section 4.2 respectively). In Section 4.3 we set $r = 10$. Finally, all algorithms are implemented in Matlab run on a PC with a 3.07GHz CPU and 24GB RAM. To reduce statistical variability, all results are averaged over five repetitions. Availability of codes and data sets are in Appendix.B.

4.1 Robust NPKL

Experiments are performed on the adult data sets that has been commonly used as benchmark data about NPKL learning [Zhuang *et al.*, 2011]. Let the number of training samples be $\bar{n}$, we randomly sample $6\bar{n}$ pairs and construct set $\mathcal{T} = \{(Q_\tau, t_\tau)\}$, i.e., $|\mathcal{T}| = 6\bar{n}$. We randomly sample 20% pairs from $\mathcal{T}$ for training, 20% for validation, and the rest for testing. For performance evaluation, we follow [Lin *et al.*, 2017; Yao and Kwok, 2018] and use the (i) testing root mean square error, RMSE $= (\sum_{\tau=1}^{\bar{n}_t}(\text{tr}(\bar{X}^\top Q_\tau \bar{X}) - t_\tau)^2/\bar{n}_t)^{1/2}$, where $\bar{X}$ is the output of the algorithm, $\bar{n}_t$ is a the number of the testing pairs; and (ii) CPU time (sec).

Robustness Against Gaussian Noise

Gaussian noise is the most natural noise type, here, to test the robustness of RSDP against such noise. Specifically, we randomly sample respectively 5% or 10% pairs from training pairs; and for selected pairs, all their labels t_τ's are added with Gaussian noise $\mathcal{N}(0, 5)$. Table 2 [2] shows the perfor-

[2]For all tables in the sequel, the best and comparable results according to the pair-wise 95% significance test are high-lighted.

n	algorithm	f value	time	sparsity.	explained var.
50	SADMM	**-18.08 ± 3.39**	2.83 ± 1.15	**0.76 ± 0.09**	**8.43 ± 0.89**
	SDPLR	**-18.08 ± 2.48**	28.20 ± 18.99	**0.76 ± 0.07**	8.33 ± 0.86
	RSDP(ADMM)	**-18.08 ± 5.30**	2.89 ± 1.26	**0.76 ± 0.08**	**8.43 ± 0.91**
	RSDP(APG)	-18.07 ± 5.29	**2.29 ± 1.16**	**0.76 ± 0.08**	**8.43 ± 0.91**
100	SADMM	**-31.57 ± 4.67**	51.02 ± 18.62	**0.79 ± 0.08**	**15.76 ± 1.82**
	SDPLR	-31.66 ± 2.93	442.15 ± 19.64	**0.79 ± 0.03**	15.38 ± 1.19
	RSDP(ADMM)	**-31.57 ± 4.43**	49.83 ± 17.20	**0.79 ± 0.05**	**15.75 ± 1.13**
	RSDP(APG)	-31.55 ± 4.42	**9.32 ± 1.24**	**0.79 ± 0.05**	**15.75 ± 1.13**
200	SADMM	**-58.77 ± 6.48**	321.27 ± 26.83	**0.82 ± 0.04**	**30.24 ± 1.54**
	SDPLR	-58.50 ± 6.43	3600.65 ± 195.62	**0.82 ± 0.03**	29.99 ± 2.69
	RSDP(ADMM)	**-58.77 ± 6.35**	316.42 ± 23.69	**0.82 ± 0.02**	**30.24 ± 1.97**
	RSDP(APG)	-58.76 ± 6.35	**74.54 ± 11.35**	**0.82 ± 0.02**	**30.24 ± 1.97**

Table 4: Performance of various sparse PCA algorithms on the colon cancer data set.

mance of the all compared algorithms. As can be seen, while squared loss based algorithms, i.e., *FW*, nmAPG and L-BFGS are very fast, they produce much higher testing RMSEs than those based on the ℓ_1-loss, i.e, *ADMM*, *SDPNAL*, *SDPLR* and *RSDP*. Among algorithms for the ℓ_1-loss, *RSDP(APG)* is the fastest and *RSDP(ADMM)* is the second fastest due to the usage of factorization. These demonstrate the robustness of the proposed formulation and the efficiency of the proposed robust SDP algorithms.

Robustness Against Flipping Labels

In real applications, some attackers want to deteriorate the system's learning performance by directly flipping the labels [Raykar *et al.*, 2010]. This is also the worst case of label noise. To further test the robustness our method, we consider such scenario here. Specifically, we take 5% or 10% pairs from training pairs; for selected pairs, all their labels t_τ's are reverse (i.e., making $t_\tau = 1 - t_\tau$). Table 2 shows performance of the all compared algorithms. As can be seen, all algorithms based on ℓ_1-loss again produces lower testing RMSE than the squared loss based algorithms; and RSDP is faster than other ℓ_1-loss based algorithms, i.e., SADMM, SDPNAL and SDPLR.

4.2 Robust CMVU

Newsgroups 20, is used here. As in [Song *et al.*, 2008], we construct the set $\mathcal{N}$ by considering the 1% nearest neighbor pairs of each point. We first construct $1\%n$ square Euclid distances $\{d_\tau\}$'s as reference 'labels' using noiseless data, then add random or outlier to data in MVU modle to output embdding $\bar{X}$ by all compared algorithms. The tradeoff parameter γ is set to 0.01 as a default. For performance evaluation, same as in robust NPLK, we also use the testing RMSE and CPU time (sec).

Robustness Again Small Deviations

We add Gaussian noise $\mathcal{N}(0, 0.01\bar{x})$, where $\bar{x}$ is a vector contains means of each features, to *Newsgroups 20* data in each dimension. Tabel 3 shows the performance. The observations are the same as that of Section 4.1. We can see that RSDP produces much lower testing RMSE and is the most efficient among algorithms working with the ℓ_1 loss.

Robustness Against Large Outliers

Except small deviations, there also can be outliers in the data. Here, we randomly smaple 5% and 10% data points respectively, for the selected each data points x_i, we convert it into outliers by adding large random noise, $\mathcal{N}(0, 5\tilde{x})$ where $\tilde{x}$ is a vector made from largest elements in the absolute value among each feature. Tabel 3 shows the performance against outliers. Again, by adopting factorization and the ℓ_1-loss, RSDP is not only efficient but also achieves the lowest testing RMSE.

4.3 Sparse PCA

We use the colon cancer data set which contains 2000 microarray readings from 62 subjects. In order to vary the problem dimension n, we randomly sampled readings. We set $\gamma = 10$ to obtain sparse solution. Results are reported in Table 4 (including the running CPU time (sec), the objective value at convergence, the sparsity of the solution and the captured variance, and these measurements have been used in [Laue, 2012; D'aspremont *et al.*, 2007]). As can been seen, all compared algorithms produce close solutions. The RSDP(ADMM) is comparable with the state-of-the-arts, and RSDP(APG) is much faster than the others when n is large.

5 Conclusion

In this paper, we propose a robust formulation of semidefinite programing (SDP) by replacing commonly used squared loss with ℓ_1 loss in standard SDP applications. As the resulting optimization problem is neither convex nor smooth, where existing SDP algorithms cannot be applied, we propose a new algorithm based on majorization-minimization. The algorithm is not efficient, but also guaranteed converging to some critical points. Finally, we demonstrate the efficiency and robustness over state-of-the-arts SDP solvers using three applications as kernel learning, matrix variance unfolding and sparse PCA. As a future work, we plan to automatically select the choice of loss function by automated machine learning [Yao *et al.*, 2018].

Acknowledgments

This research was supported in part by the National Natural Science Foundation of China (No. 61663049, 61165012 and 2019010108203008).

References

[Beck and Teboulle, 2009] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIJIS*, 2(1):183–202, 2009.

[Boyd and Vandenberghe, 2004] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Boyd *et al.*, 2011] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[Burer and Monteiro, 2003] S. Burer and R.D.C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *MathProg*, 95:329–357, 2003.

[D'aspremont *et al.*, 2007] A. D'aspremont, El L. Ghaoui, M. I. Jordan, and G. R. G. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. *SIAM Review*, 49(3):434–48, 2007.

[Dekel *et al.*, 2010] Ofer Dekel, Ohad Shamir, and Lin Xiao. Learning to classify with missing and corrupted features. *ML*, 2010.

[Helmberg *et al.*, 1996] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIOPT*, 1996.

[Hoi *et al.*, 2007] S. Hoi, R. Jin, and M.R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *ICML*, pages 361–368, 2007.

[Hu *et al.*, 2011] E.-L. Hu, B. Wang, and S.-C. Chen. BCD-NPKL: Scalable non-parametric kernel learning using block coordinate descent. In *ICML*, pages 209–216, 2011.

[Hunter and Lange, 2004] D. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37, 2004.

[Jaggi, 2013] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *ICML*, 2013.

[Kulis *et al.*, 2007] B. Kulis, A. C. Surendran, and J. C. Platt. Fast low-rank semidefinite programming for embedding and clustering. In *AISTAT*, pages 235–242, 2007.

[Lange *et al.*, 2000] K. Lange, R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *JCGS*, 9(1):1–20, 2000.

[Laue, 2012] S. Laue. A hybrid algorithm for convex semidefinite optimization. In *ICML*, pages 177–184. Omnipress, 2012.

[Lemon *et al.*, 2016] A. Lemon, A. So, and Y. Ye. Low-rank semidefinite programming: Theory and applications. *Foundations and Trends in Optimization*, 2(1-2):1–156, 2016.

[Li and Lin, 2015] H. Li and Z. Lin. Accelerated proximal gradient methods for nonconvex programming. In *NeurIPS*, pages 379–387. 2015.

[Li *et al.*, 2008] Z. Li, J. Liu, and X. Tang. Pairwise constraint propagation by semidefinite programming for semi-supervised classification. In *ICML*, pages 576–583, 2008.

[Lin *et al.*, 2017] Z. Lin, C. Xu, and H. Zha. Robust matrix factorization by majorization minimization. *TPAMI*, (99), 2017.

[Nocedal and Wright, 2006] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.

[Pumir *et al.*, 2018] T. Pumir, S. Jelassi, and N. Boumal. Smoothed analysis of the low-rank approach for smooth semidefinite programs. In *NeurIPS*, 2018.

[Raykar *et al.*, 2010] V. Raykar, S. Yu, L. Zhao, G. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *JMLR*, 2010.

[Royer, 2017] M. Royer. Adaptive clustering through semidefinite programming. In *NeurIPS*, pages 1793–1801, 2017.

[Song *et al.*, 2008] L. Song, A. Smola, K. Borgwardt, and A. Gretton. Colored maximum variance unfolding. In *NeurIPS*, 2008.

[Srinadh *et al.*, 2016] B. Srinadh, K. Anastasios, and Sujay S. Dropping convexity for faster semidefinite optimization. Preprint arXiv:1509.03917, 2016.

[Toh *et al.*, 2015] K.-C. Toh, L. Yang, and D. Sun. SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints. *MPC*, 2015.

[Vandenberghe and Boyd, 1996] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[Weinberger and Saul, 2006] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *IJCV*, 2006.

[Weinberger *et al.*, 2004] K. Q. Weinberger, F. Sha, and L.K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *ICML*, pages 839–846, 2004.

[Yao and Kwok, 2018] Q. Yao and J. Kwok. Scalable robust matrix factorization with nonconvex loss. In *NeurIPS 31*, pages 5066–5075. 2018.

[Yao *et al.*, 2018] Q. Yao, M. Wang, Y. Chen, W. Dai, Y. Hu, Y. Li, W. Tu, Q. Yang, and Y. Yu. Taking human out of learning applications: A survey on automated machine learning. Technical report, arXiv preprint, 2018.

[Zheng and Lafferty, 2015] Q. Zheng and J. Lafferty. A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements. In *NeurIPS*, 2015.

[Zhuang *et al.*, 2011] J. Zhuang, I. Tsang, and S. Hoi. A family of simple non-parametric kernel learning algorithms. *JMLR*, 12:1313–1347, 2011.

[Zou and Xue, 2018] H. Zou and L.Z. Xue. A selective overview of sparse principal component analysis. *Proceedings of the IEEE*, 2018.