Low-Rank Quadratic Semidefinite Programming

Ganzhao Yuan[a], Zhenjie Zhang[b], Bernard Ghanem[c], Zhifeng Hao[a,d]

[a]*School of Computer Science & Engineering, South China University of Technology*
[b]*Advanced Digital Sciences Center, Illinois at Singapore Pte.*
[c]*King Abdullah University of Science & Technology, KSA*
[d]*Faculty of Computer, Guangdong University of Technology*

Abstract

Low rank matrix approximation is an attractive model in large scale machine learning problems, because it can not only reduce the memory and runtime complexity, but also provide a natural way to regularize parameters while preserving learning accuracy. In this paper, we address a special class of nonconvex quadratic matrix optimization problems, which require a low rank positive semidefinite solution. Despite their non-convexity, we exploit the structure of these problems to derive an efficient solver that converges to their local optima. Furthermore, we show that the proposed solution is capable of dramatically enhancing the efficiency and scalability of a variety of concrete problems, which are of significant interest to the machine learning community. These problems include the *Top-k Eigenvalue Problem*, *Distance Learning* and *Kernel Learning*. Extensive experiments on UCI benchmarks have shown the effectiveness and efficiency of our proposed method.

Keywords: Semidefinite Programming, Metric Learning, Kernel Learning, Eigenvalue Decomposition, Low-Rank and Sparse Matrix Approximation

1. Introduction

Low rank matrix approximation is a hot topic in the machine learning community and has been well studied in the last decade. It is used in a wide spectrum of applications in different problem settings, e.g. dimensionality reduction [31], distance learning [31], kernel learning [22] and principal component analysis [31]. Low rank matrices are attractive because they can not only reduce memory and runtime complexity, but also provide a natural way to regularize parameters while preserving learning accuracy.

While traditional low rank matrix approximation purely focuses on the identification of a low rank positive semidefinite matrix (PSD) "close" to a target matrix, many learning problems involve the minimization of an objective cost function *and* low rank approximation simultaneously. In this paper, we show that a general class of popular learning problems (e.g. top-k eigenvalue approximation, Mahananobis distance learning, and inductive kernel learning [20], etc) can be cast as quadratic semidefinite programming problems, whose objective functions have the same overall form with varying regularization terms. The solution to any of these problems is a low rank symmetric matrix. The

Email addresses: y.ganzhao@scut.edu.cn (Ganzhao Yuan), zhenjie@adsc.com.sg (Zhenjie Zhang), b.ghanem@kaust.edu.sa (Bernard Ghanem), mazfhao@scut.edu.cn (Zhifeng Hao)

overall matrix form of the objective function is usually non-convex, so greedy descent algorithms that guarantee convergence to local extrema can be used. Despite their popularity, many of these algorithms do not scale well with matrix dimensions. In this paper, we show that the Conjugate Gradient Method coupled with an effective linear search s-trategy solves the non-convex problem efficiently, even in high matrix dimensions. To the best of our knowledge, the use of this decent method in quadratic matrix optimization is novel.

Apart from high dimension of the learning task, handling constraints is also an issue. Most metric learning approaches need to cycle through metric constraints many times before converging [1, 20, 31, 11]. Since the number of constraints scale quadratically with the number of training samples, this becomes very expensive for large scale data. In this paper, we cast the metric learning problem into a quadratic matrix optimization framework, where constraints are handled implicitly. Consequently, our proposed solution can easily scale to large scale learning problems.

Not surprisingly, our approach has close connections with recent work on low-rank and sparse matrix approximation. Low-rank and sparse approximation is now becoming a fundamental tool in fields as diverse as image analysis [19], collaborative prediction [28] and background modeling [35]. To some extent, its popularity in the machine learning community is mainly due to the advent of low-rank non-negative matrix factorization [23], and sparse recovery / compressed sensing [6]. (i) Low-rank approximation to a target matrix is very attractive. Since large datasets often take the form of matrices which may contain millions of entries, it is a daunting task to store and operate on these large matrices. A natural way to promote the efficiency (and also regularize the learning model) is to limit the rank of the corresponding matrix. This is often achieved by nuclear norm regularizer [35] and low rank factorization [28]. (ii) On the other hand, sparse learning is another adaptive model in machine learning. Generally speaking, sparse learning refers to a set of methods to learning that seek a trade-off between some goodness-of-fit measure and sparsity of the result, the latter property allowing better interpretability. To promote sparsity, some sparsity-inducing constraints/regularizers are often used (e.g. l_0/l_1 regularized [35, 6], non-negative [23] and hinge-loss based constraints [28]). (iii) It may not be appropriate to distinguish the low-rank learning and sparse learning, since the low-rank can be viewed as a special case of sparsity (sparsity on the singular value). The combination of the two is also desirable [5, 35]. By considering different aspects of low-rank and sparse matrix approximation, different matrix approximation methods have been developed in recent years, e.g., to improve robustness of principle component analysis [5, 9], to promote the efficiency of matrix decomposition [18, 35], to deal with online stochastic approximation [19], and to make extensions by the idea of manifold learning [17].

While working with low rank and sparse *rectangular matrix* has attracted extensive research efforts, low rank optimization on the cone of positive semidefinite *square matrix* also has become an active research direction in recent years. Low rank metric learning with Bergman matrix divergences is studied in [22], where the learning process is viewed as a matrix nearness problem with linear inequality constraints. The *Kullback-Leibler (KL) divergence* and *von Neumann (VN) divergence* are employed to measure the "closeness" between two gaussian distributions. These divergences are natural for learning low rank kernels. Since they are defined only over the cone of PSD matrices, they not only preserve rank but also positive semidefiniteness. It was shown that this algorithm scales linearly in the number of data points and quadratically in the rank of the input matrix. Due to restrictions on the rank of the kernel matrix, this method cannot be used for dimensionality reduction. More recently, the squared Frobenius divergence regularizer has received much attention in the metric learning community [20, 25]. Unlike *KL* divergence and *VN* divergence, positive semidefiniteness has to be explicitly enforced when using the Frobenius divergence. Given an $n \times n$ matrix $\mathbf{M}$, if the matrix is low rank PSD with rank $d<n$, one intuitive method to enforce the low rank positive semidefiniteness is to represent the matrix in terms of a factorization $\mathbf{M} = \mathbf{L}\mathbf{L}^T$, with $\mathbf{L}$ an $n \times d$ matrix. In [25], the metric learning process is viewed as a linear regression problem on fixed rank PSD matrices. It was shown that this Frobenius divergence technique yields comparable performance to *KL* divergence in metric learning. Other works on PSD optimization can be found in [4, 21, 1, 2] and therein, it is generally believed that the low-rank optimization is more computationally challenging than its full rank counterpart, but in practice, when the final optimal solution presents low-rank property, it often leads to a more efficient algorithm.

Contributions: Our proposed optimization framework makes the following contributions. **(1)** We show that our framework is general, since it is applicable to a wide variety of problems that are of significant interest to the machine learning community. **(2)** We study the theoretical properties of low rank factorization $\mathbf{M} = \mathbf{L}\mathbf{L}^\mathbf{T}$ for positive semidefinite problems and show that the change of variables does not introduce any local minima. **(3)** We show that our framework is significantly more computationally efficient than existing solutions, while still achieving (and in some cases improving upon) state-of-the-art performance. Efficiency is achieved by applying a traditional conjugate gradient method that makes use of the inherent structured nature of the problem.

The paper is organized as follows. Section 2 proposes our low rank matrix optimization framework and emphasizes three popular problems that fall in this framework. We shed theoretical light on the low rank optimization problem and the proposed solution in Section 3. Section 4 presents the general and efficient solver algorithm. In Section 5, we report empirical results on benchmark datasets and provide quantitative comparison with existing methods. Section 6 concludes the paper and discusses future research directions.

Mathematical Notation: The inner product between two matrices $\mathbf{A}$ and $\mathbf{B}$ is defined as $\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{ij} \mathbf{A}_{ij}\mathbf{B}_{ij}$, and Frobenius norm of a matrix $\mathbf{A}$ as $\|\mathbf{A}\|_F$. We use $\mathbf{I}_n$ to denote the identity matrix of size $n \times n$ and $\mathbf{A} \succcurlyeq \mathbf{0}$ to denote $\mathbf{A}$ is a Positive Semi-Definite (PSD) matrix.

2. Low Rank Optimization Framework

In this paper, we focus on a general class of quadratic matrix optimization problems. We are given PSD matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ (i.e. $\mathbf{A} \succcurlyeq \mathbf{0}, \mathbf{B} \succcurlyeq \mathbf{0}$) and an arbitrary symmetric matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. We seek a symmetric PSD matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ that solves Eq (1).

$$\min_{\mathbf{M} \in \mathbb{R}^{n \times n}} \quad G(\mathbf{M}) = \operatorname{tr}\left((\mathbf{AM})^2 + \mathbf{B}\mathbf{M}^2 + 2\mathbf{CM} \right) \qquad (1)$$
$$s.t. \quad \mathbf{M} \succcurlyeq \mathbf{0}$$

In general, the solution of Eq(1) has to be searched in a space of dimension $\mathcal{O}(n^2)$. Solving such a problem becomes rapidly untractable for large n. In order to solve Eq(1) at a reduced computational cost, we assume Eq(1) presents a low rank solution, i.e.

$$rank(\mathbf{M}^*) \ll n$$

In order to handle the PSD constraint, we let $\mathbf{M} = \mathbf{L}\mathbf{L}^T$ with $\mathbf{L} \in \mathbb{R}^{n\times d}$, $d < n$ and rewrite the problem in Eq (2).

$$\min_{\mathbf{L}\in\mathbb{R}^{n\times d}} F(\mathbf{L}) = \operatorname{tr}\left((\mathbf{A}\mathbf{L}\mathbf{L}^T)^2 + \mathbf{B}(\mathbf{L}\mathbf{L}^T)^2 + 2\mathbf{C}\mathbf{L}\mathbf{L}^T\right) \tag{2}$$

The existence of a solution depends on the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. However, since we assume that $\mathbf{A}$ and $\mathbf{B}$ are PSD, $G(\mathbf{M})$ is convex with respect to $\mathbf{M}$ (refer to Section 3). The problem is bounded and the solution always exists. Next, we show how three popular learning problems are special cases of Eq (2).

2.1. Top-k Eigenvalue Approximation

In many applications, it is important to find the "closest" (in Frobenius norm) rank k approximation to a symmetric matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$. This k rank approximation is constructed using the k largest positive eigenvalues of $\mathbf{K}$ and their corresponding eigenvectors. We reformulate the problem without explicitly computing the eigenvalues and eigenvectors. We minimize the Frobenius norm between $\mathbf{K}$ and $\mathbf{M}$, i.e. $\|\mathbf{M} - \mathbf{K}\|_F^2 = \operatorname{tr}\left((\mathbf{M}-\mathbf{K})(\mathbf{M}-\mathbf{K})^T\right) = \operatorname{tr}\left(\mathbf{M}^2 - 2\mathbf{M}\mathbf{K} + \mathbf{K}^2\right)$, leading to the problem in Eq (3). Clearly, it is consistent with Eq (2), where $\mathbf{A} = \mathbf{0}$, $\mathbf{B} = \mathbf{I}_n$ and $\mathbf{C} = -\mathbf{K}$.

$$\min_{\mathbf{L}\in\mathbb{R}^{n\times k}} \operatorname{tr}\left(\left(\mathbf{L}\mathbf{L}^T\right)^2 - 2\mathbf{K}\mathbf{L}\mathbf{L}^T\right) \tag{3}$$

We now discuss some applications of the top-k eigenvalue approximation. Low rank approximation of a symmetric matrix has been studied extensively in the kernel machine literature [12, 33, 34]. Generally, one need to solve the linear systems of the form $(\mathbf{K} + \lambda\mathbf{I}_n)\vec{x} = \vec{b}$, where $\mathbf{K}$ is the kernel matrix, $\lambda > 0$ is a regularization parameter and $\mathbf{I}_n$ is the $n \times n$ identity matrix. Given the low-rank approximation $\mathbf{K} \approx \mathbf{L}\mathbf{L}^\mathbf{T}$, the linear system can be solved via the Woodbury formula:

$$(\mathbf{K} + \lambda\mathbf{I}_n)^{-1} \approx \frac{1}{\lambda}\left(\mathbf{I}_n - \mathbf{L}\left(\lambda\mathbf{I}_k + \mathbf{L}^\mathbf{T}\mathbf{L}\right)^{-1}\mathbf{L}^\mathbf{T}\right)$$

which only needs $\mathcal{O}(k^2 n)$ time and $\mathcal{O}(kn)$ memory. Therefore, it can be used to accelerate the gaussian processes [33] and least-squares SVMs.

Another application of top-k eigenvalue approximation is to reconstruct the top-k eigenvector of a Gram matrix from its low-rank decomposition. Note that our low rank optimization problem Eq.(3) does not provide orthogonal approximations to the eigenfunctions. Thanks to Fowlkes's matrix completion view [14], our low rank method can also be utilized for obtaining orthogonal eigenvectors. Given the low-rank approximation $\mathbf{K} \approx \mathbf{L}\mathbf{L}^\mathbf{T}$, the top k eigenvectors $\mathbf{U}$ of $\mathbf{K}$ can be obtained as

$$\mathbf{U} = \mathbf{L}\mathbf{V}\mathbf{D}^{-\frac{1}{2}} \tag{4}$$

where $\mathbf{V}$ and $\mathbf{D}$ are from the eigenvalue decomposition decomposition of the $k \times k$ matrix $\mathbf{L}^\mathbf{T}\mathbf{L} = \mathbf{V}\mathbf{D}\mathbf{V}^\mathbf{T}$. This is equivalent to solving the following top-k eigenvalue system:

$$\max_{\mathbf{U}\in\mathbb{R}^{n\times k}} \quad tr(\mathbf{U}^T\mathbf{K}\mathbf{U}), \;\; s.t. \; \mathbf{U}^T\mathbf{U} = \mathbf{I}_k \tag{5}$$

The reader is referred to [14, 34] for more details. Therefore low-rank approximation is useful for algorithms that rely on eigenvectors of the Gram matrix, such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Laplacian eigenmap, etc.
Furthermore, the eigenvalue approximation is also applicable to indefinite kernel learning[7] and semidefinite programming[32]. For example, [32] suggests using this type of low-rank approximation (See Page 11 of [32]) to perform PSD cone projection. One merit of such iterative method is that it can take advantage of a good initial guess in every iteration.

2.2. Distance Learning

Given a d-dimensional vector space $\mathbb{R}^d$ and two points $\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j \in \mathbb{R}^d$, a distance function $D(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)$ returns a non-negative real value, indicating the dissimilarity of $\vec{\mathbf{x}}_i$ and $\vec{\mathbf{x}}_j$. Distance learning is the problem of constructing $D(.,.)$ that conforms to two groups of point pairs, $\mathcal{S}$ and $\mathcal{N}$. A point pair $(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)$ from $\mathcal{S}$ (or $\mathcal{N}$) indicates that $\vec{\mathbf{x}}_i$ and $\vec{\mathbf{x}}_j$ are similar (or dissimilar) to each other. The optimal $D(\cdot,\cdot)$ is the function that minimizes the generalized objective function in Eq (6).

$$\min_{D(\cdot,\cdot)} \; \text{Reg}(D) + \lambda \text{Loss}(D, \mathcal{S}, \mathcal{N})$$

Here, the loss function $\text{Loss}(D, \mathcal{S}, \mathcal{N})$ measures the consistency between the values of $D(\cdot,\cdot)$ and pairwise relations in $\mathcal{S}$ and $\mathcal{N}$. The regularizer $\text{Reg}(D)$ alleviates the problem of overfitting. The tradeoff factor λ balances the loss function and regularization. Next, we discuss different regularizer and loss functions, as well as, their relationship with our optimization framework. We focus on a special class of distance functions, called the *Mahalanobis* distance, defined as $D(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = (\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j)\mathbf{M}(\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j)^T$, where $\mathbf{M}$ is PSD.

Since the choice of regularizer is application dependent, we discuss three popular distance regularizers. **(i)** Trace regularizer: $\text{Reg}_t(D) = \text{tr}(\mathbf{M})$. It is analogous to the ℓ_1 norm regularizer for vector spaces and is thus related to the sparseness on the singular values of $\mathbf{M}$. Thus the trace regularizer encourages low rank solutions. **(ii)** Frobenius norm regularizer: $\text{Reg}_f(D) = \|\mathbf{M}\|_F^2$. It is analogous to the ℓ_2 norm regularizer for vector spaces. Generally, it will result in $\mathbf{M}$ being full rank [20]. **(iii)** Kernel Target Alignment regularizer: $\text{Reg}_i(\mathbf{M}) = \|\mathbf{M} - \mathbf{I}_d\|_F^2$. It tends to capture additional information contained in the spectral domain [11]. Since these regularizer functions are compatible with our framework in Eq (1), it is straightforward to employ any of them in distance learning. Since $\text{Reg}_i(\mathbf{M})$ has a kernel target alignment interpretation [8], we use it as our default regularizer. In what follows, we examine two popular loss models that are also compatible with Eq (1).

Linearity Loss Model: It measures the distance separation between pairs of samples in $\mathcal{S}$ and $\mathcal{N}$. Of course, a small separation is desired. This loss function is concisely formulated as $\text{Loss}_l(D, \mathcal{S}, \mathcal{N}) = \sum_{(i,j)\in\mathcal{S}} D(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) - \sum_{(i,j)\in\mathcal{N}} D(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \frac{1}{2}\sum_{i,j}\left(\vec{\mathbf{x}}_i^T\mathbf{M}\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j^T\mathbf{M}\vec{\mathbf{x}}_j\right)\mathbf{S}_{ij}$, where $\mathbf{S}_{ij} = +1$ if $\vec{\mathbf{x}}_j \in \mathcal{S}$ and $\mathbf{S}_{ij} = -1$ if $\vec{\mathbf{x}}_j \in \mathcal{N}$. Let $\mathbf{R}$ be a diagonal matrix with each diagonal entry filled with the sum of the corresponding row in $\mathbf{S}$. Defining the Laplace matrix $\mathbf{T} = \mathbf{R} - \mathbf{S}$, we have $\text{Loss}_l(D, \mathcal{S}, \mathcal{N}) =$

$\mathrm{tr}(\mathbf{X}^T\mathbf{MXR}) - \mathrm{tr}(\mathbf{X}^T\mathbf{MXS}) = \mathrm{tr}(\mathbf{XTX}^T\mathbf{M})$.

Combining the above loss function with the default regularizer $\mathrm{Reg}_i(\mathbf{M})$, we obtain the objective in Eq (6). This objective function is consistent with our framework, whereby $\mathbf{A} = \mathbf{0}$, $\mathbf{B} = \mathbf{I}_d$ and $\mathbf{C} = \frac{1}{2}\lambda\mathbf{XTX}^T - \mathbf{I}_d$.

$$\min_{\mathbf{M}\succcurlyeq\mathbf{0}} \mathrm{tr}(\mathbf{M}^2 - 2\mathbf{M} + \lambda\mathbf{XTX}^T\mathbf{M}) \tag{6}$$

Kernel Target Alignment Loss Model: It assumes that points in the training set are attributed labels that indicate which classes they belong to. The model compares two kernel matrices: one is based on the distance metric and the other on class labels. Let c be the number of classes and let $\mathbf{Y} = [\vec{\mathbf{y}}_i|\dots|\vec{\mathbf{y}}_m]$ denote the class labels assigned to all training data $\{\vec{\mathbf{x}}_1,\dots,\vec{\mathbf{x}}_m\}$. Each $\vec{\mathbf{y}}_i = (y_i^1\dots y_i^c)^T \in \{0,1\}^c$ is a binary vector of c elements. Following [8], this loss model introduces a kernel $\mathbf{K}_D = \mathbf{YY}^T$. To construct the Mahalanobis distance that preserves the class memberships of the training samples, another kernel is constructed using matrix $\mathbf{M}$. Since $\mathbf{M}$ is PSD, the distance function with respect to $\mathbf{M}$ is equivalent to the Euclidean distance after a linear transformation $\mathbf{L}$, such that $\mathbf{M} = \mathbf{LL}^T$. It is easy to show that the dot product between training points in the new vector space is $\mathbf{K}_X = (\mathbf{XL})(\mathbf{XL})^T = \mathbf{XMX}^T$. This loss model computes the difference between $\mathbf{K}_D$ and $\mathbf{K}_X$ as the dissimilarity between two zero-mean Gaussian distributions with covariance matrices $\mathbf{K}_D$ and $\mathbf{K}_X$ respectively [30]. Here we use the Frobenius norm to measure the distance between $\mathbf{K}_D$ and $\mathbf{K}_X$. Again, adding the regularizer $\mathrm{Reg}_i(\mathbf{M})$, we obtain Eq (7) that is consistent with our framework, where $\mathbf{A} = \sqrt{\lambda}\mathbf{X}^T\mathbf{X}$, $\mathbf{B} = \mathbf{I}_d$ and $\mathbf{C} = -\lambda\mathbf{X}^T\mathbf{YY}^T\mathbf{X} - \mathbf{I}_d$.

$$\min_{\mathbf{M}\succcurlyeq\mathbf{0}} \mathrm{tr}\left(\lambda\left(\mathbf{X}^T\mathbf{XM}\right)^2 + \mathbf{M}^2 - 2\mathbf{M} - 2\lambda\mathbf{X}^T\mathbf{YY}^T\mathbf{XM}\right) \tag{7}$$

2.3. Kernel Learning

Kernel learning is another popular problem that deals with the optimization of low rank PSD matrices. In [20], Jain et al. presented a kernel learning problem stated as follows. Given an existing kernel matrix $\mathbf{N}$, it is required to construct a new kernel matrix $\mathbf{K}$ which is close to $\mathbf{N}$ while satisfying some constraint conditions, as in Eq (8). Here, g_{ij} and b_{ij} are pairs of constraints that impact the shape of $\mathbf{K}$, here $i, j \in 1,\dots,m$, where m stands for the number of training examples. In [20], the authors proved that the above problem is equivalent to the problem of learning linear transformation kernel functions, as indicated in Eq (8).

$$\min_{\mathbf{K}\succcurlyeq\mathbf{0}} \ \mathrm{Reg}(\mathbf{N}^{-1}\mathbf{K}) \quad \text{s.t. } g_{ij}(\mathbf{K}) \le b_{ij} \Leftrightarrow \min_{\mathbf{W}\succcurlyeq\mathbf{0}} \ \mathrm{Reg}(\mathbf{W}) \quad \text{s.t. } \ g_{ij}(\phi^T\mathbf{W}\phi) \le b_{ij} \tag{8}$$

Here, $\mathbf{W}^* = \alpha\mathbf{I}_m + \phi^T\mathbf{Q}^*\phi$ and $\mathbf{Q}^* = \mathbf{N}^{-1}(\mathbf{K}^* - \alpha\mathbf{N})\mathbf{N}^{-1}$. Furthermore, $\mathbf{K}^* = \phi^T\mathbf{W}^*\phi$, α is a parameter. $\mathbf{K}^*$ and $\mathbf{W}^*$ are the optimal solutions for the equivalent problems in Eq. (8). The trace and Fronbenius regularizers were introduced to extend the learning framework to supervised and semi-supervised learning [20], as shown in Eq. (9).

$$\min_{\mathbf{K}\succeq\mathbf{0}} \left(\tau \text{tr}\left(\mathbf{N}^{-1}\mathbf{K}\right) + \|\mathbf{N}^{-1}\mathbf{K}\|_F^2\right) \;\; s.t. \; g_{ij}(\mathbf{K}) \leq b_{ij} \tag{9}$$

where $\tau \geq 0$ is a parameter. When $\tau > 0$, the above formulation is the semi-supervised learning model; when $\tau = 0$, it reduces to the supervised learning model. In terms of constraints, Euclidean distance constraints between a pair of points (ϕ_i, ϕ_j) can be formulated as $\mathbf{K}(i,i)+\mathbf{K}(j,j)-2\mathbf{K}(i,j) \geq b_{ij}$ or $\mathbf{K}(i,i)+\mathbf{K}(j,j)-2\mathbf{K}(i,j) \leq b_{ij}$. Furthermore, we can simplify $\sum_{i,j\in\mathcal{S}} \mathbf{G}_{ij} - \sum_{i,j\in\mathcal{N}} \mathbf{G}_{ij} = \sum_{i,j=1} \mathbf{G}_{ij}\mathbf{S}_{i,j} = \sum_{i,j=1} \left(\vec{e}_i\vec{e}_i^T - 2\vec{e}_i\vec{e}_j^T + \vec{e}_j\vec{e}_j^T\right)$ $\mathbf{S}_{i,j} = \sum_{i,j=1} \left(2\vec{e}_i\vec{e}_i^T - 2\vec{e}_i\vec{e}_j^T\right)\mathbf{S}_{i,j} = 2\sum_{i,j=1} \vec{e}_i\vec{e}_i^T\mathbf{S}_{i,j} - 2\sum_{i,j=1} \vec{e}_i\vec{e}_j^T\mathbf{S}_{i,j} = 2\mathbf{R} - 2\mathbf{S} = 2\mathbf{T}$, where $\mathbf{S}$ and $\mathbf{T}$ are the score matrix and Laplacian matrix respectively as defined earlier. Therefore, it is consistent with the framework with $\mathbf{A} = \mathbf{N}^{-1}$, $\mathbf{B} = \mathbf{0}$ and $\mathbf{C} = \tau\mathbf{N}^{-1} + \lambda\mathbf{T}$.

$$\min_{\mathbf{K}\succeq\mathbf{0}} tr(\mathbf{N}^{-1}\mathbf{K}\mathbf{N}^{-1}\mathbf{K}) + tr(\tau\mathbf{N}^{-1} + \lambda\mathbf{T})\mathbf{K} \tag{10}$$

3. Optimality Analysis

In this section, we will discuss the optimality of the results using our low-rank optimization framework. In particular, we aim to provide a general guideline on the local minima our framework stops at. We will show when the result matrix $\mathbf{L}$ spans in the full column rank space, $\mathbf{M} = \mathbf{L}\mathbf{L}^T$ is the global minima of the original optimization problem. To prove this property, we start with the following lemmas proved in previous studies.

Lemma 1. *If $\mathbf{X}$ and $\mathbf{Y}$ are PSD, then $\mathbf{X} \otimes \mathbf{Y}$ is PSD, where $\otimes$ is the tensor product between two matrices. See P620 in [10].*

It is not difficult to show that the gradient and hessian matrix of Eq (1) are

$$\frac{\partial G}{\partial \mathbf{M}} = 2\mathbf{AMA} + \mathbf{BM} + \mathbf{MB} + 2\mathbf{C} \tag{11}$$

$$\frac{\partial^2 G}{\partial^2 \mathbf{M}} = 2\mathbf{A} \otimes \mathbf{A} + \mathbf{I}_n \otimes \mathbf{B} + \mathbf{B} \otimes \mathbf{I}_n \tag{12}$$

Similarly, the gradient of Eq (2) can be computed as

$$\frac{\partial F}{\partial \mathbf{L}} = 4\mathbf{A}\mathbf{L}\mathbf{L}^T\mathbf{A}\mathbf{L} + 2\mathbf{B}\mathbf{L}\mathbf{L}^T\mathbf{L} + 2\mathbf{L}\mathbf{L}^T\mathbf{B}\mathbf{L} + 4\mathbf{C}\mathbf{L} \tag{13}$$

According to Lemma 1, the Hessian matrix is PSD, so $G(\mathbf{M})$ is convex with respect to $\mathbf{M}$. For more details of matrix differential calculus, please refer to [24].

To analyze the optimality of the solutions, we first present the first order KKT condition. By introducing the dual variables $\mathbf{S} \in \mathbb{R}^{d\times d}$ for the PSD constraints, we obtain the Karush-Kuhn-Tucker (KKT) condition for Eq. (1) as follows.

$$\mathbf{M} \succeq 0 \;\; \text{(Feasibility)} \tag{14}$$

$$\mathbf{S} \succeq 0 \;\; \text{(Non-Negativity)} \tag{15}$$

$$\frac{\partial G}{\partial \mathbf{M}} - \mathbf{S} = 0 \;\; \text{(Optimality)} \tag{16}$$

$$\mathbf{SM} = 0 \;\; \text{(Complementary Slackness)} \tag{17}$$

Since $G(\mathbf{M})$ is a convex function with respect to $\mathbf{M}$, the KKT optimality conditions are *necessary and sufficient* for convex optimization problems[3].

Lemma 2. *Suppose* $\mathbf{P}, \mathbf{Q} \in \mathbf{R}^{n\times d}$ *satisfy* $\mathbf{PP^T} = \mathbf{QQ^T}$. *Then for some orthogonal matrix* $\mathbf{U} \in \mathbf{R}^{d\times d}$, $\mathbf{Q} = \mathbf{PU}$. *See Lemma 2.1 in [4].*

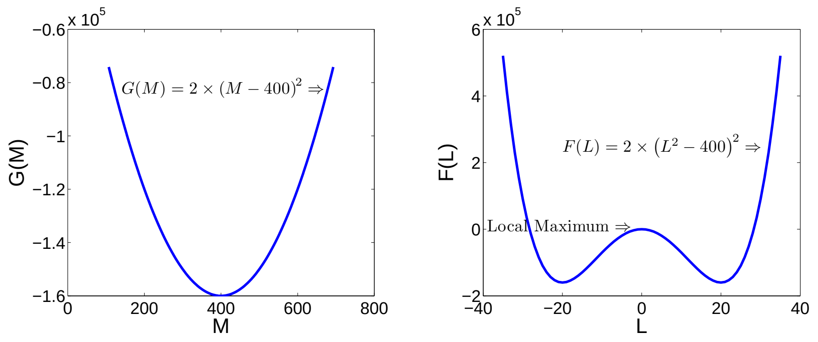
Lemma 3. *For all orthogonal* $\mathbf{U}$, $\mathbf{R}$ *is a local minimum of Eq (2)* $\Leftrightarrow$ $\mathbf{RU}$ *is a local minimum of Eq (2).*
Proof. *(⇒)* $\mathbf{R}$ *is a local minimum of Eq (2)* $\Rightarrow \frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{R}} = 4\mathbf{ARR}^T\mathbf{AR} + 2\mathbf{BRR}^T\mathbf{R} + 2\mathbf{RR}^T\mathbf{BR} + 4\mathbf{CR} = \mathbf{0}$. *But,* $\frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{RU}} = \left(\frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{R}}\right)\mathbf{U} \Rightarrow \mathbf{RU}$ *is also a local minima. (⇐)* $\mathbf{RU}$ *is a local minimum of Eq (2)* $\Rightarrow \frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{RU}} = \left(\frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{R}}\right)\mathbf{U} = \mathbf{0} \Rightarrow \frac{\partial F}{\partial \mathbf{L}}|_{\mathbf{R}} = \mathbf{0} \Rightarrow \mathbf{R}$ *is also a local minimum.*

Lemma 4. *Suppose* $\mathbf{M}^* = \mathbf{R}^*\mathbf{R}^{*T}$, *where* $\mathbf{M}^*$ *is feasible for Eq (1) and* $\mathbf{R}^*$ *is feasible for Eq (2). Then* $\mathbf{M}^*$ *is a local minimum of Eq (1)* $\Leftrightarrow$ $\mathbf{R}^*$ *is a local minimum of Eq (2).*
Proof. *(⇒) The proof is by continuity. Since* $G(\mathbf{M})$ *is continually differential, if* $\mathbf{M}$ *is a local minimum of Eq(1), then each* $\mathbf{R}$ *satisfying* $\mathbf{M} = \mathbf{RR}^T$ *is a local minimum of Eq(2). (⇐) The proof is by contradiction. We now suppose that* $\mathbf{M}^*$ *is not a local minimum of Eq (1). Therefore, there always exists a sequence of feasible solutions* $\mathbf{M}^k$ *of Eq (1) converging to* $\mathbf{M}^*$ *such that* $G(\mathbf{M}^k) < G(\mathbf{M}^*)$ *for all* k. *For each* k, *we can choose* $\mathbf{R}^k$ *such that* $\mathbf{M}^k = \mathbf{R}^k(\mathbf{R}^k)^T$. *Since* $\mathbf{M}^k$ *is bounded, it follows that* $\mathbf{R}^k$ *is bounded and hence has a subsequence* $\mathbf{R}^k$ *converging to some* $\mathbf{R}$ *such that* $\mathbf{M}^* = \mathbf{RR}^T$. *Since* $F(\mathbf{R}^k) = G(\mathbf{M}^k) < G(\mathbf{M}^*) = F(\mathbf{R}^*)$, *we see that* $\mathbf{R}$ *is not a local minimum of Eq (2).* ***(i)****Since* $\mathbf{M}^* = \mathbf{R}^*(\mathbf{R}^*)^T = \mathbf{RR}^T$, $\mathbf{R}^* = \mathbf{RU}$ *for some orthogonal* $\mathbf{U}$ *(See Lemma (2)).* ***(ii)*** *Since* $\mathbf{R}$ *is not a local minima of Eq (2),* $\mathbf{RU}$ *is not a local minima for all orthogonal* $\mathbf{U}$ *(See Lemma (3)). Combining (i) and (ii), we conclude that* $\mathbf{R}^* = \mathbf{RU}$ *is not a local minimum of Eq (2).*

Remark: Since any local minima is a global minima for the convex optimization problem (1), Lemma 4 reveals a nice property that any local minima of Eq (2) is a global minima. This is based on the assumption that the matrix $\mathbf{R}^*$ spans in the full column rank space (i.e. there is no null space in $\mathbf{R}^*$), so that we can always choose $\mathbf{R}$ satisfying $\mathbf{M}^* = \mathbf{R}^*(\mathbf{R}^*)^\mathbf{T}$. Therefore the low rank matrix M^* can be uniquely determined by a full column rank matrix $\mathbf{R}^*$, (i.e. rank($\mathbf{R}^*$)=rank($\mathbf{M}^*$)). However, in real world applications, we find that this assumption is mild and always holds. Empirically, we always obtain a resulting full column rank matrix $\mathbf{R}^*$ in our experiments with random initializations. A simple one-dimensional case is given in Figure 1. It is clear that the change of variable of the form $\mathbf{M} = \mathbf{LL^T}$ only introduce one local maximum which is defined in the null space, and does not introduce any extraneous local minima. To sum up, we obtain two conclusions. **(a)** Eq (2) is *equivalent* to Eq (1), Eq (2) is bounded, and the solution of Eq (2) always exists. **(b)** When the full column rank assumption holds, the change of variable does not introduce any extraneous local minima (i.e. any local minima of Eq (2) is a global minima).

4. Efficient Solver

In this section, we describe an efficient solver for the problem in Eq 2. We call our proposed algorithm the Low Rank Optimization (LRO) method. The gradient of $F(\mathbf{L})$ with respect to $\mathbf{L}$ can be computed using Eq (13). Generally, since $F(\mathbf{L})$ is nonconvex, it



(a) original function (b) transformed function

Figure 1: Geometric interpretation of the $\mathbf{M} = \mathbf{L}\mathbf{L}^\mathbf{T}$ factorization in one-dimensional case when $\mathbf{A} = [1]; \mathbf{B} = [1]; \mathbf{C} = [-800]$.

is hard to utilize the second order information in this nonconvex optimization problem. Newton method in this scheme may diverge or converge to a saddle point or even to a point of local maximum [26]. Since the limited-memory quasi-Newton method (L-bfgs) requires matrix manipulation over its previous directions in each iteration, it leads to expensive computation when the number of columns in $\mathbf{L}$ is large (refer to Section 5.4 for a comparison with LRO). On the other hand, the first-order *Conjugate Gradient (CG) Algorithm* [27] is quite a popular iterative approach for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite. In many cases (e.g. especially when the matrices are sparse), the CG method significantly outperforms L-bfgs. Therefore, we iteratively apply the CG algorithm to Eq (13) until it converges to a local minimum. The complete algorithm is summarized in Algorithm 1.

In each CG iteration, the conjugate direction picks up the moving direction based on the gradient of the objective function at $\mathbf{L}_t$. A good survey on classical rules for updating the conjugate direction are found in [29]. The Polak-Ribiere (PR) update automatically restarts an iteration when the update term ρ becomes negative. In our experiments, we observed that this update rule is generally the best choice, both for convergence speed and numerical stability.

Computing a proper step size μ is an important feature of the CG method (refer to Step 13), since it significantly impacts the convergence of Algorithm 1. The optimal step size μ^* is computed by performing a local line search around the current solution $\mathbf{L}_{t-1}$. As such, $\mu^* = \min_{0<\mu<1} F(\mathbf{L}_{t-1} + \mu \boldsymbol{\Delta}_{t-1})$. Since it is quite expensive to evaluate $F(\mathbf{L})$ for a specific $\mathbf{L}$, a simple inexact line search using the Armijo sufficient decreasing condition may incur high computational overhead. To overcome such difficulty, we analyze the objective function and derive a more efficient solution.

In the following, we discuss how to find μ using first order condition. To simplify the notation, we use $\mathbf{L}$ and $\boldsymbol{\Delta}$ to denote $\mathbf{L}_{t-1}$ and $\boldsymbol{\Delta}_{\mathbf{t-1}}$ respectively. The objective along

Algorithm 1 Conjugate Gradient Descent for the Quadratic Semidefinite Programming

```
1: Input: stopping parameter $\epsilon$
2: Initialize the original solution $\mathbf{L}_0$ to a random full column rank matrix {We simply
   initialize $L_0$ =orth(randn($n, d$)) in MATLAB.}, set $t = 0$
3: while not converge do
4:   if $t > 0$ then
5:     if $\frac{F(\mathbf{L}_t)-F(\mathbf{L}_{t-1})}{|F(\mathbf{L}_t)|+1} < \epsilon$ or $\frac{\|\mathbf{L}_t-\mathbf{L}_{t-1}\|_F}{\sqrt{n}} < \epsilon$ then
6:       Return $\mathbf{L}_t$
7:     end if
8:     $\rho = \max\left\{0, \frac{\langle \frac{\partial F}{\partial \mathbf{L_t}}, \frac{\partial F}{\partial \mathbf{L_t}} - \frac{\partial F}{\partial \mathbf{L_{t-1}}} \rangle}{\langle \frac{\partial F}{\partial \mathbf{L_t}}, \frac{\partial F}{\partial \mathbf{L_t}} \rangle}\right\}$
9:     $\mathbf{\Delta}_t = -\frac{\partial F}{\partial \mathbf{L_t}} + \rho \mathbf{\Delta}_{t-1}$
10:  else
11:    $\mathbf{\Delta}_t = -\frac{\partial F}{\partial \mathbf{L_t}}$
12:  end if
13:  Solve the cubic equation $\frac{\mathrm{d}F(\mathbf{L}_t+\mu\mathbf{\Delta}_t)}{\mathrm{d}\mu} = 0$ $(0 < \mu < 1)$ to get the candidate steps:
     *cstep*
14:  if (# of *cstep* =0) Return $\mathbf{L}_t$,
15:  elseif (# of *cstep* =1) $\mu = cstep(1)$
16:  elseif (# of *cstep* =3) select one that makes the greatest descent as $\mu$,
17:  endif
18:  Increment $t$ by 1
19:  Update $\mathbf{L}_t = \mathbf{L}_{t-1} + \mu \mathbf{\Delta}_{t-1}$
20: end while
```

the search direction $\mathbf{\Delta}$ can be computed as:

$$\begin{aligned} F(\mathbf{L}+\mu\mathbf{\Delta}) &= \operatorname{tr}\big(\mathbf{A}(\mathbf{L}+\mu\mathbf{\Delta})^T(\mathbf{L}+\mu\mathbf{\Delta})\mathbf{A}(\mathbf{L}+\mu\mathbf{\Delta})^T(\mathbf{L}+\mu\mathbf{\Delta}) \\ &\quad +\mathbf{B}(\mathbf{L}+\mu\mathbf{\Delta})^T(\mathbf{L}+\mu\mathbf{\Delta})(\mathbf{L}+\mu\mathbf{\Delta})^T(\mathbf{L}+\mu\mathbf{\Delta})\mathbf{B} \\ &\quad +2\mathbf{C}(\mathbf{L}+\mu\mathbf{\Delta})^T(\mathbf{L}+\mu\mathbf{\Delta})\big) \end{aligned}$$

It is easy to see that $F(\mathbf{L}+\mu\mathbf{\Delta})$ can be written as a 4th order polynomial function in μ, as $F(\mathbf{L}+\mu\mathbf{\Delta}) = \sum_{i=1}^{5} a_i\mu^{i-1}$, where $\{a_i\}_{i=1}^{5}$ are constants that depend on $\mathbf{\Delta}, \mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{L}$ and can be efficiently computed using properties of the matrix trace. Therefore, solving $\frac{\mathrm{d}F(\mathbf{L}+\mu\mathbf{\Delta})}{\mathrm{d}\mu} = 0$ is equivalent to solving the cubic equation $\sum_{i=1}^{4}(ia_{i+1})\mu^{i-1} = 0$. Based on the current descent direction $\mathbf{\Delta}$, the cubic equation may produce 0, 1 or 3 real-valued solutions. If the equation has no solution (Step 14), the algorithm has converged. If the equation produces a single solution, it is used as the optimal step size (Step 15). If the equation produces multiple solutions, we select the μ that leads to the greatest descent with respect to the objective. One may worry that this may impair the performance of the line search, however, in practice, even for a problem which contains tens of thousands of variables in our experiments, the number of times the line search produces 3 solutions is usually negligible (less than 5). One cause may be the general local convexity of the optimization problem (refer to the right subfigure in Figure 1).

The convergence of Algorithm 1 is evident. Since F has continuous first-order gradients over the open set domain, then for any $\mathbf{L}_t$ and $\mathbf{\Delta}_t$, the Taylor series on some open interval of $\mu(\mu > 0)$ can be expressed as:

$$F(\mathbf{L}_{t+1}) = F(\mathbf{L}_t + \mu\mathbf{\Delta}_t) = F(\mathbf{L}_t) + \mu\langle\mathbf{\Delta}_t, \frac{\partial F}{\partial \mathbf{L}_t}\rangle + o(\mu^2)$$

where $o(\mu^2)$ the Taylor polynomials of higher degree which is larger than 2. Since the conjugate gradient direction is a descent direction [27, 29] with $\langle\mathbf{\Delta}_t, \frac{\partial F}{\partial \mathbf{L}_t}\rangle < 0$, there exist a sufficient small μ such that $F(\mathbf{L}_{t+1}) < F(\mathbf{L}_t)$. Combining our exact line search program, Algorithm 1 is guaranteed to decrease the objection function in every iteration. A global convergence proof of the Polak-Ribiere conjugate gradient method based on Armijo line search for general smooth (not necessarily convex) functions can be found in [16]. Due to the greedy descent property of Algorithm 1, convergence to a local minimum is guaranteed.

Time Complexity: The total number of variables is $n \times d$. According to the quadratic termination property of the CG algorithm, Algorithm 1 converges to a local minimum in $n \times d$ or fewer iterations ($\#it$). Since computing the gradient requires $\mathcal{O}(n^2 d)$ operations, the complexity of Algorithm 1 is $\mathcal{O}(\#it \times n^2 d)$.

5. Experiments

In this section, we provide experimental validation of our LRO method on both synthetic and real data. We evaluate the performance of LRO in three applications: Top-k eigenvalue problem, distance learning and kernel learning. All experiments are performed using MATLAB 7.8 on a single core Intel Pentium E3300 1.86GHZ processor with 1GB RAM. The CG algorithm is stopped when the relative change in F or $\mathbf{L}$ (refer to Line 5 in Algorithm 1) is below a tolerance $\epsilon = 10^{-6}$ in all our experiments.

5.1. Top-k Eigenvalue

The Top-k Eigenvalue approximation problem can be solved by MATLAB solver 'eig', whose time complexity is $\mathcal{O}(n^3)$. As such, it is not scalable to large scale matrix problems. However, efficient Top-k eigenvalue solvers exists: 'JDQR' is a publicly available MATLAB package[1] based on the Jacobi-Davidson method [13], 'eigs' is a Matlab function based on Lanczos Method. We use its default stopping criterion for all methods. In our experiments, we compare our LRO method against 'JDQR' and 'eigs' by generating an arbitrary indefinite matrix $\mathbf{K}$ and applying the three Matlab solvers to estimate the closest rank d approximation to $\mathbf{K}$. Note that LRO approximates $\mathbf{K}$ with $\mathbf{L}\mathbf{L}^T$ where $\mathbf{L} \in \mathbb{R}^{n\times d}$ and reconstruct the orthogonal eigenvectors by Eq (4). Experimental results have been obtained by averaging 10 runs over different sized matrices $\mathbf{H}$ and different ranks ($d = \{1, 10, 40, 80, \sqrt{n}\}$) . We report the runtime and normalized objective value $Objective = \frac{1}{n}tr(\mathbf{U}^T\mathbf{K}\mathbf{U})$ for the three solvers in Table 1. The orthogonality measure $Orthogonality = \|\mathbf{U}^T\mathbf{U} - \mathbf{I}_n\|_F$ for LRO is also included in the table. Two conclusions can be drawn. **(1)** LRO outperforms 'JDQR' in terms of computational efficiency while achieving a larger objective value, moreover, LRO gives comparable efficiency to 'eigs'. **(2)** Although the problem is nonconvex, our experiments demonstrate that LRO tends

[1]http://www.staff.science.uu.nl/∽vorst102/JDQR.html

		JDQR		eigs		LRO		
n	d	*Time*	*Objective*	*Time*	*Objective*	*Time*	*Objective*	*Orthogonality*
1000	1	0.40	0.044442	0.44	0.044442	0.35	0.044442	1.11e-16
1000	10	1.77	0.432857	0.89	0.432857	0.65	0.432856	3.01e-15
1000	40	3.45	0.722600	1.13	1.617016	1.27	1.617010	1.18e-14
1000	80	7.22	0.963321	1.71	3.018015	2.60	3.018006	2.20e-14
1000	31	3.10	0.722600	1.02	1.275306	1.22	1.275302	9.72e-15
2000	1	1.37	0.031521	1.76	0.031521	0.89	0.031521	2.22e-16
2000	10	7.66	0.308372	3.91	0.308372	2.35	0.308371	2.56e-15
2000	40	10.08	0.459197	4.94	1.187446	4.92	1.187442	1.17e-14
2000	80	14.00	0.519013	7.18	2.281043	8.25	2.281015	2.30e-14
2000	44	10.35	0.459197	5.19	1.300596	5.97	1.300588	1.29e-14
3000	1	4.13	0.025740	4.77	0.025740	3.75	0.025739	0.00e-00
3000	10	17.23	0.253965	8.82	0.253965	7.75	0.253964	3.07e-15
3000	40	20.50	0.328882	11.93	0.984637	15.24	0.984630	1.25e-14
3000	80	24.88	0.378555	17.66	1.909765	17.97	1.909729	2.18e-14
3000	54	21.53	0.353763	14.03	1.313890	18.40	1.313872	1.48e-14
4000	1	7.42	0.022359	8.71	0.022359	5.06	0.022359	2.22e-16
4000	10	35.98	0.198505	18.41	0.220359	14.05	0.220358	2.68e-15
4000	40	37.74	0.263871	25.08	0.859493	20.59	0.859483	1.30e-14
4000	80	41.53	0.307176	37.39	1.676963	34.53	1.676943	2.21e-14
4000	63	38.03	0.285531	29.73	1.333807	28.34	1.333783	1.83e-14
5000	1	13.71	0.019956	16.62	0.019956	8.77	0.019956	4.44e-16
5000	10	55.08	0.177945	41.48	0.197501	23.15	0.197500	3.43e-15
5000	40	56.95	0.217035	41.28	0.773370	40.40	0.773364	1.30e-14
5000	80	62.47	0.256050	62.48	1.514662	66.05	1.514646	2.26e-14
5000	70	61.17	0.236555	56.68	1.331808	53.77	1.331795	1.90e-14

Table 1: Time comparison (in seconds) with 'JDQR' and 'eigs'.

to converge to the global solution, so the risk of falling into undesirable local minima is generally avoided.

Remark: Our experimental results demonstrate the potential of our low-rank optimization as a general quadratic semidefinite programming algorithms. Current modern technique of computing leading eigenpairs such as 'eigs' is based on Lanczos algorithm. However, the Lanczos algorithm is known to suffer from numerical stability and the orthogonality of the solution $\mathbf{U}$ is not always guaranteed. In practice, the orthogonality of $\mathbf{U}$ is quickly lost and in some cases could even be linearly dependent. One needs to employ Gram-Schmidt process to reorthogonalize $\mathbf{U}$ into a basis spanning the Krylov subspace corresponding to the target matrix $\mathbf{K}$. For more discussions on the Lanczos algorithm, please refer to Section 5 of [9]. On the other hand, our method is based on simple gradient descent and the orthogonal eigenvectors can be directly reconstructed by Eq (4). It is expected to be more robust.

dataset	LRO-KTA	LRO-L	ITML	NCA	LMNN	IGML
soybean	**00.0 ± 0.0**	05.4 ± 5.2	00.7 ± 1.0	00.1 ± 0.8	02.2 ± 2.1	01.8 ± 2.1
iris	**02.7 ± 1.5**	04.3 ± 2.1	04.3 ± 2.7	04.1 ± 1.6	04.5 ± 2.1	**02.7 ± 1.7**
wine	**01.5 ± 1.0**	02.8 ± 1.6	07.7 ± 3.0	04.6 ± 1.5	04.1 ± 1.8	05.0 ± 1.6
sonar	22.4 ± 2.5	23.9 ± 3.8	28.3 ± 6.3	26.5 ± 4.6	**20.3 ± 4.4**	28.1 ± 4.5
glass	**30.6 ± 2.4**	31.0 ± 3.3	36.2 ± 3.4	36.9 ± 2.7	34.9 ± 3.2	35.8 ± 2.3
ionosphere	14.1 ± 3.1	15.5 ± 1.5	**11.1 ± 2.6**	17.2 ± 1.6	15.0 ± 1.9	16.6 ± 1.8
pima	**26.5 ± 1.7**	27.6 ± 1.6	27.8 ± 1.7	27.8 ± 1.6	27.1 ± 1.7	27.6 ± 1.9
segment	03.7 ± 0.6	11.4 ± 0.7	11.6 ± 1.0	09.0 ± 1.7	**03.6 ± 0.9**	05.9 ± 3.4
optdigits	02.9 ± 0.3	02.5 ± 0.3	02.1 ± 0.3	02.1 ± 0.3	**01.6 ± 0.3**	03.2 ± 0.3
waveform	16.6 ± 0.6	**16.0 ± 0.6**	19.7 ± 0.7	20.1 ± 0.7	19.1 ± 0.7	30.6 ± 0.7

Table 2: Error rates (in %) on 10 UCI benchmark datasets for distance learning

5.2. Distance Learning

In this section, we demonstrate the efficacy and efficiency of LRO and compare it to two popular metric learning algorithms on several real-world benchmark datasets. After learning the optimal Mahalanobis distance function from the training set, we use a kNN classifier ($k = 3$) to classify each test sample. The hyper-parameter λ is set by a typical fivefold cross-validation procedure. We search over the values $\lambda = \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. To reduce the number of parameters, we set $d = \min(n, \min(60, \max(40, \sqrt{n})))$ in all experiments.

Low dimensional Data: We compare our LRO method using the kernel target alignment model (LRO-KTA) and the linearity model (LRO-L) against popular methods, including ITML [11], NCA [15], LMNN [31] and IGML [30]. The experiments are run on 10 datasets from the UCI data repository. We use the default stopping criterion for all other methods.

Table 2 shows the classification error of six methods on 10 datasets averaged over 20 runs with the corresponding standard deviation. A detailed description of these datasets are available in Table 3. The best result is highlighted in bold. From Table 2, several conclusions can be drawn. Firstly, for 5 out of 10 datasets, LRO-KTA performs better than the other classifiers. LMNN performs better than the other approaches for 3 out of 10 datasets. Secondly, LRO-L only gives modest performance. This demonstrates that the linear loss model enforces the similarity or dissimilarity between training samples too strictly and is sensitive to noise. This impairs classification performance.

High dimensional Data: We also apply LRO to high-dimensional large scale datasets, where sparsity is an important factor that can impact the efficiency of the algorithm. Because many metric learning methods cycle through the distance constraints many times, they fail to make good use of the sparsity inherent to the data. Our LRO algorithm can handle the constraints implicitly by simply employing sparse matrix multiplication. We use both sparse and non-sparse datasets in this experiment. First, we compare the computational efficiency of LRO to that of existing full-rank methods ITML and LMNN, and the low-rank method SURF [25]. Figure 2(a) shows the run-time taken to learn metrics of dimensionality $n = 100$ to $n = 5000$ on the “gisette” dataset with $m = 3000$ instances and $c = 2$ classes. All methods are run to convergence. We observe that the full-rank methods scale quadratically with the dimension n, while the

dataset	# class	# tr	# te	# feat	sparsity
w1a	2	2,477	47,272	300	0.9618
w3a	2	4,912	44,837	300	0.9612
isolet	26	6238	1559	617	0
gisette	2	1,000	1,000	5,000	0.01
letters	26	15,000	5,000	16	0
20news	20	15,935	3,993	62,061	0.9987
real-sim	2	36,155	36,154	20,958	0.9976

Table 3: Description of 7 large scale benchmark datasets on distance learning

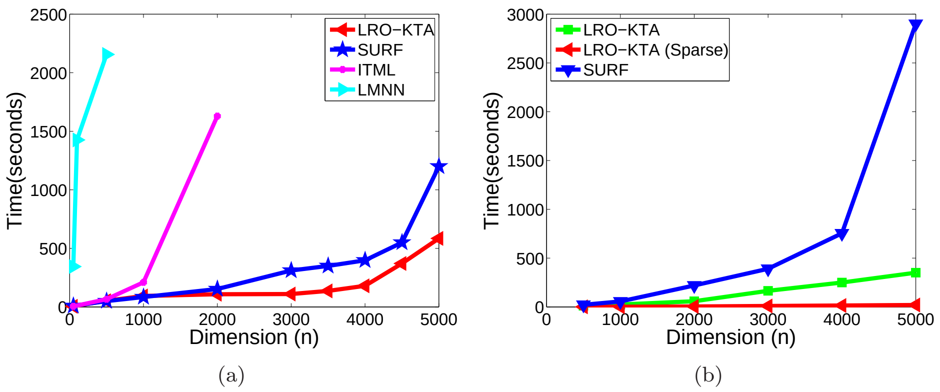
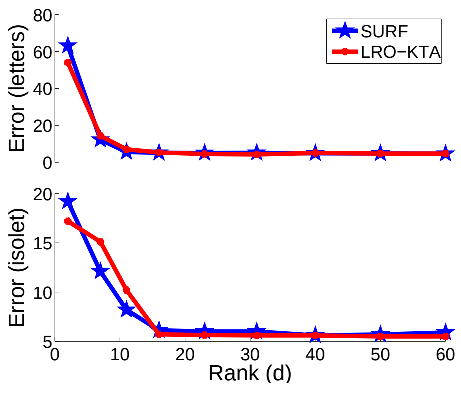


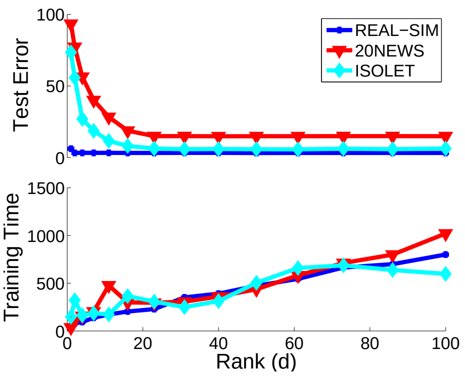
Figure 2: (a): Run-time comparison on the "gisette" dataset between full-rank methods: ITML, LMNN, and low-rank methods: SURF and LRO-KTA. Full-rank methods do not scale well, whereas low-rank methods scale roughly linearly with d. (b): Run-time comparison with SURF on the "realsim" dataset. LRO-KTA is substantially faster than SURF.

low-rank ones scale roughly linearly with LRO-KTA outperforming SURF. Here, we note that "gisette" is a non-sparse dataset. Next, we focus on comparing LRO with SURF on the large scale sparse "realsim" dataset. We select $m = 6000$ instances from the dataset. Figure 4(b) shows the run-time taken to learn the metrics of dimensionality $n = 500$ to $n = 5000$. LRO-KTA is more than 10 times faster than SURF on this sparse dataset.

We make further comparisons with SURF. Table 4 shows the mean classification error of the two methods over seven datasets averaged over three runs. For both of the algorithms, we set the rank of the learned matrix equal to $d \in \{20, 20, 20, 40, 20, 16, 20\}$. Based on the results in Table 4, we draw the following observations. Firstly, LRO-KTA is significantly faster on the sparse datasets and generally faster on the non-sparse datasets. For example, LRO-KTA is about 10 times faster than SURF on the sparse "realsim" and "20news" datasets. For the non-sparse dataset "gisette", LRO-KTA is four times faster. Secondly, LRO-KTA only gives modest convergence speed on the "letters" and "isolet" datasets. Stopping criterion may be responsible for this result. Thirdly, both low rank algorithms yield similar test error, with LRO-KTA performing slightly better. We also study the relationship between the reduced rank d, training time, and test error rate. Figure 3(a) shows the test error rate as a function of d for SURF and LRO-KTA on the



(a)



(b)

Figure 3: (a): Test error rate vs. rank on "isolet" and "letter" datasets. (b): *Training time* and *Test error rate* of LRO-KTA with varying rank.

"isolet" and "letter" datasets, respectively. Figure 3(b) shows training time and error rate as a function of matrix rank for LRO-KTA. Clearly, we observe that approximating the matrix with a higher rank will be less efficient *without* necessarily resulting in better accuracy.

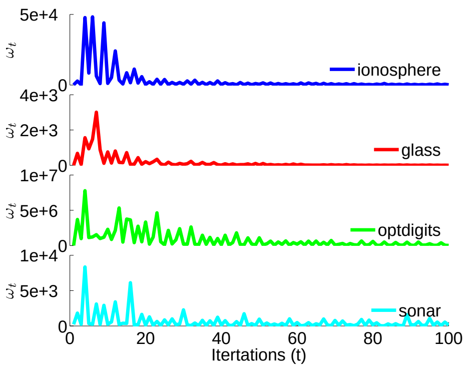
Run Time Comparison							
Alg.	w1a	w3a	isolet	gisette	letters	20news	realsim
SURF	40 s	39 s	91 s	1020 s	5 s	7140 s	4200 s
LRO-KTA	6 s	13 s	289 s	300 s	34 s	587 s	360 s
Error Rate Comparison							
Alg.	w1a	w3a	isolet	gisette	letters	20news	realsim
SURF	2.7%	2.2%	3.26%	5.1%	5.3%	18.0%	5.8%
LRO-KTA	2.4%	2.4%	5.8%	5.3%	4.0%	14.8%	3.24%

Table 4: LRO-KTA vs. SURF when applied to distance learning

Global Convergence: According to the KKT optimality condition, our local search algorithm terminates at global optimum when $\frac{\partial G}{\partial(\mathbf{L_t L_t^T})}$ becomes PSD, i.e. $\omega_t = 0$, where ω_t is defined as:

$$\omega_t = \lambda_{\max}\left(-\frac{\partial G}{\partial\left(\mathbf{L_t L_t^T}\right)}\right)$$

Here $\lambda_{\max}(\cdot)$ is the largest eigenvalue of a matrix. In order to learn some asymptotic behavior of the algorithms, we report the values of ω_t after t^{th} iteration in Figure 4. It is demonstrated that the algorithm converges to the local minima with ω decreasing to 0 ($\omega \searrow 0$). To further evaluate global convergence of the algorithm, we learn the best linear transformation $\mathbf{L}^*$ starting from different initializations with different ϵ ($10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}$). We report the average and standard deviation of


(a) on low dimensional dataset

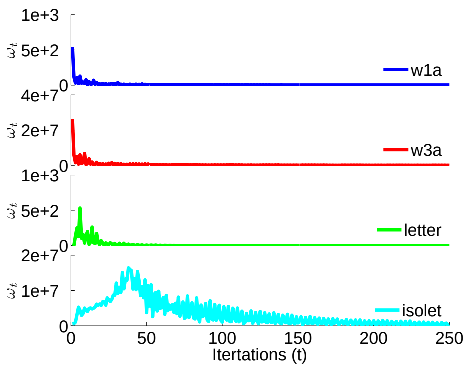

(b) on high dimensional dataset

Figure 4: (a): The value of ω_t after t^{th} iteration on low dimensional dataset: "ionosphere", "glass", "optdigits", "sonar". (b): The value of ω_t after t^{th} iteration on low dimensional dataset: "w1a", "w3a", "letter", "isolet".

$\{F(\mathbf{L}^*), \omega_{\#it}$ and $\#it\}$ over 10 runs in Table 6. Here $\#it$ is defined as the number of iterations which the algorithm takes to converge. Therefore, two conclusions can be obtained. (i) The small standard deviations of the objective function and the small value of the eigenfunction $\omega_{\#it}$ indicate that LRO tends not to get stuck at undesirable local minima. (ii)Although CG is guaranteed to converge in $\mathcal{O}(n \times d)$ iterations, in our experiments, we find reasonable solutions are found often in much fewer iterations.

5.3. Kernel Learning

We apply LRO to the kernel learning problem and study its computational efficiency. In our experiments, we select a subset of training patterns from the "letters" dataset, varying from $m = 100$ to $m = 1500$. A gaussian kernel $K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \exp\left(\frac{-\|\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j\|^2}{\sigma^2}\right)$ with varying bandwidth $\sigma = \{1, 5, 8\}$ is used for learning. Here, we set the rank $d = 20$. As can be seen from Figure 5, our algorithm scales roughly linearly with the number of training examples.

A recent kernel learning algorithm can be found in [20]. They represent the PSD matrix in terms of a factorization $\mathbf{M} = \mathbf{JOJ}^T$, where $\mathbf{J} \in \mathbb{R}^{n \times d}$ is a pre-specified matrix, $\mathbf{O} \in \mathbb{R}^{d \times d}$ is unknown PSD matrix. The number of parameters in the model can thus be reduced from n^2 to d^2. The optimization algorithm is based on Uzawa's inexact algorithm which is original designed for matrix completion. The learning procedure in [20] is repeated by cycling through the constraints, whereby each iteration involves an SVD operation that projects the current solution into the feasible set (i.e. the convex cone of positive semidefinite). Since there are too many parameters that need to be set beforehand and due to the inexact line search technique used in this method, slow convergence might ensue. Setting the parameters of this method is non-trivial, so it is difficult to conduct a fair comparison between its performance and that of our LRO method.

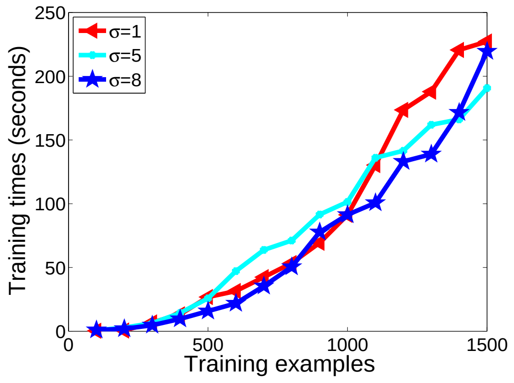


Figure 5: Scalability of LRO-KTA with training set size for kernel learning.

dataset	CG (sec)	L-bfgs (sec)
optdigits	5.14(0.76)	5.88(1.00)
w1a	4.55(1.09)	5.57(0.43)
w3a	6.13(0.21)	6.82(0.90)
isolet	184.41(6.29)	176.76(7.09)
letters	17.15(0.59)	13.23(0.66)
gisette	364.06(83.25)	678.53(39.31)
real-sim	159.72(1.88)	386.16(18.04)

Table 5: Training Time Comparison: CG v.s. L-bfgs

5.4. Conjugate Gradient vs. L-bfgs

Here, we validate the use of the CG method in LRO. To do this, we implement two versions of LRO-KTA: one with the CG method and the other with the L-bfgs method. For the L-bfgs version, we use 20 past updates of the position and gradient to approximate the Hessian matrix. We report the average training time over 7 datasets averaged over 20 runs with the corresponding standard deviation in Table 5. The CG method often results in faster convergence especially for large scale datasets (e.g. "gisette" and "realsim").

5.5. Discussions

It is hard to compare against other solvers for PSD matrix optimization in all experiments, since they usually optimize different objectives. For example, the Top-k eigenvalue problem can be also approximately solved by Nyström method [33, 14, 34], however, such sampling method is inefficient when high accuracy is required. Therefore, we only compare against exact sparse eigenvalue methods such as 'eigs' and 'JQDR'. Burer and Kulis's optimization method [4] mainly focuses on linear low rank Semidefinite Programming (SDP). Journee's optimization method [21] is more general, but incapable of solving all the different problems LRO can. Generally, these methods can not utilize the structure of the optimization problem. Moreover, at every iteration, these methods need to perform eigenvalue or singular value decomposition, which is not scalable to large scale applications. Furthermore, NCA [15] is a also metric learning algorithm which uses the decomposition $\mathbf{M} = \mathbf{L}\mathbf{L}^{\mathbf{T}}$. Since the objective with respective to $\mathbf{M}$ is non-convex, they uses Carl Edward Rasmussen's Matlab solver 'minimize' to ensure a robust solution. Note that 'minimize' is designed for general purpose, it uses CG and cubic polynomial interpolation method, which is inefficient. In our experiments, we found their inexact algorithm, even for median scale dataset, often converges much slower than our method. We therefore skip the results with NCA and compare our LRO method with the new metric learning solver SURF. We also report the scalability of our method for kernel learning.

	$\epsilon = 1e^{-6}$			$\epsilon = 1e^{-8}$		
dataset	$F(\mathbf{L})$	$\omega_{\#it}$	$\#it$	$F(\mathbf{L})$	$\omega_{\#it}$	$\#it$
soybean	1.3e1(5.2e-7)	4.1e1(7.4e-3)	1.0e2(0.0e0)	1.3e1(8.7e-8)	4.2e0(6.0e-3)	1.8e2(0.0e0)
iris	1.1e0(5.2e-16)	1.1e-9(2.0e-16)	1.1e1(0.0e0)	1.1e0(5.2e-16)	1.1e-9(2.0e-16)	1.1e1(0.0e0)
wine	1.1e2(0.0e0)	2.7e0(6.4e-13)	4.7e1(0.0e0)	1.1e2(2.8e-14)	8.6e-1(2.6e-12)	7.8e1(0.0e0)
sonar	2.2e3(1.3e0)	4.4e1(2.8e1)	1.2e2(1.8e1)	2.2e3(5.8e-1)	8.7e0(4.2e0)	4.3e2(6.7e1)
glass	1.8e3(4.0e-9)	3.2e1(3.1e-6)	6.9e1(0.0e0)	1.8e3(1.5e-10)	9.2e0(6.0e-7)	9.7e1(0.0e0)
ionosphere	5.2e3(1.0e-11)	3.6e2(5.3e-8)	1.3e2(0.0e0)	5.2e3(3.5e-12)	1.2e2(1.1e-9)	2.2e2(0.0e0)
pima	3.6e4(3.0e-11)	5.7e2(2.2e-9)	4.0e1(0.0e0)	3.6e4(2.4e-11)	2.1e2(4.7e-9)	7.8e1(0.0e0)
optdigits	1.8e5(3.3e1)	4.9e4(3.8e4)	2.3e2(6.7e0)	1.7e5(8.6e0)	2.3e4(1.7e3)	4.8e2(1.6e1)
waveform	2.2e3(2.7e-6)	5.8e1(8.1e-4)	6.3e1(0.0e0)	2.2e3(1.1e-5)	1.2e2(3.3e-1)	1.0e2(0.0e0)
w1a	5.8e2(1.1e-2)	2.4e0(7.1e-2)	1.4e2(7.2e0)	5.8e2(9.5e-3)	2.0e0(5.7e-3)	3.0e2(3.2e1)
w3a	1.2e7(1.7e3)	7.0e4(3.9e4)	1.9e2(9.8e0)	1.2e7(5.2e2)	2.3e4(1.7e4)	3.7e2(4.1e1)
letters	6.7e2(1.2e-12)	1.0e0(2.6e-9)	4.8e1(0.0e0)	6.7e2(1.5e-13)	1.7e0(6.9e-9)	6.5e1(0.0e0)

	$\epsilon = 1e^{-10}$			$\epsilon = 1e^{-12}$		
dataset	$F(\mathbf{L})$	$\omega_{\#it}$	$\#it$	$F(\mathbf{L})$	$\omega_{\#it}$	$\#it$
soybean	1.3e1(3.0e-6)	2.0e-1(2.1e-1)	4.1e2(7.1e0)	1.3e1(1.0e-10)	4.3e-3(9.1e-4)	6.1e2(0.0e0)
iris	1.1e0(3.3e-16)	1.1e-9(1.9e-16)	1.1e1(0.0e0)	1.1e0(3.3e-16)	1.1e-9(1.9e-16)	1.1e1(0.0e0)
wine	1.1e2(1.7e-14)	2.1e-2(5.1e-13)	1.5e2(0.0e0)	1.1e2(1.4e-14)	2.0e-3(1.2e-12)	2.0e2(0.0e0)
sonar	2.2e3(1.7e-3)	2.0e0(5.6e-3)	2.8e3(6.2e2)	2.2e3(4.0e-4)	2.0e0(1.5e-3)	4.1e3(2.2e2)
glass	1.8e3(1.4e-12)	2.0e-1(1.8e-7)	2.4e2(0.0e0)	1.8e3(8.2e-13)	1.4e-2(1.5e-8)	3.4e2(0.0e0)
ionosphere	5.2e3(4.6e-12)	2.4e0(1.6e-9)	7.0e2(0.0e0)	5.2e3(3.8e-12)	5.4e-1(4.6e-9)	1.9e3(0.0e0)
pima	3.6e4(1.4e-11)	6.3e0(5.3e-9)	3.0e2(0.0e0)	3.6e4(2.5e-11)	1.4e-1(2.5e-9)	5.9e2(0.0e0)
optdigits	1.7e5(3.8e-3)	4.3e2(3.2e2)	2.7e3(3.9e1)	1.7e5(1.7e-4)	9.5e0(3.5e0)	5.7e3(2.3e2)
waveform	2.3e3(7.2e-13)	1.2e1(9.8e-8)	6.2e2(0.0e0)	2.3e3(4.5e-13)	1.0e0(3.0e-8)	1.4e3(0.0e0)
w1a	5.8e2(7.9e-5)	2.0e0(4.9e-6)	2.4e3(2.2e2)	5.8e2(2.3e-6)	2.0e0(1.0e-7)	1.1e4(8.9e2)
w3a	1.2e7(1.2e0)	3.8e2(1.5e2)	2.3e3(8.8e1)	1.2e7(0.2e0)	1.5e1(0.9e2)	2.2e4(1.2e2)
letters	6.7e2(1.1e-13)	8.0e-2(1.7e-10)	1.1e2(0.0e0)	6.7e2(5.7e-14)	1.1e-3(1.1e-11)	1.4e2(0.0e0)

Table 6: Global Convergence of the LRO-KTA Algorithm.

6. Conclusion and Future Work

In this paper, we propose an efficient, simple and general optimization framework for the low rank PSD matrix problem. We apply this framework to a variety of important problems that occur in machine learning. Extensive empirical results on benchmark datasets show that our proposed method is accurate, stable, efficient, and scalable to large and sparse datasets. We also show that our method can outperform popular and state-of-the-art methods. Our experiments hint that our method enjoys stable convergence and does not get stuck at undesirable low minima. Therefore, future work will focus on providing theoretical validation for the convergence process.

References

[1] W. Bian and D. Tao. Learning a distance metric by empirical loss minimization. In *IJCAI*, pages 1186–1191, 2011.

[2] W. Bian and D. Tao. Max-min distance analysis by using sequential sdp relaxation for dimension reduction. *TPAMI*, 33(5):1037–1050, 2011.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[4] S. Burer and R. D. C. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103:2005, 2003.

[5] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of ACM*, 58(3):11, 2011.

[6] E. J. Candès, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[7] J. Chen and J. Ye. Training svm with indefinite kernels. In *ICML*, pages 136–143, 2008.

[8] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. S. Kandola. On kernel-target alignment. In *NIPS*, pages 367–373, 2001.

[9] A. d' Aspremont. Subsampling algorithms for semidefinite programming. *Stochastic Systems*, 2(1):274–305, 2011.

[10] J. Dattorro. *Convex Optimization & Euclidean Distance Geometry*. Meboo Publishing USA, 2011.

[11] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *ICML*, pages 209–216, 2007.

[12] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[13] D. R. Fokkema, G. L. G. Sleijpen, and H. A. V. der Vorst. Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils. *Journal of Scientific Computing*, 20:94–125, 1998.

[14] C. Fowlkes, S. Belongie, F. R. K. Chung, and J. Malik. Spectral grouping using the nyström method. *TPAMI*, 26(2):214–225, 2004.

[15] J. Goldberger, S. T. Roweis, G. E. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[16] L. Grippo and S. Lucidi. A globally convergent version of the polak-ribière conjugate gradient method. *Mathematical Programming*, 77:375–391, 1997.

[17] N. Guan, D. Tao, Z. Luo, and B. Yuan. Manifold regularized discriminative nonnegative matrix factorization with fast gradient descent. *IEEE Transactions on Image Processing*, 20(7):2030–2048, 2011.

[18] N. Guan, D. Tao, Z. Luo, and B. Yuan. Nenmf: An optimal gradient method for nonnegative matrix factorization. *IEEE Transactions on Image Processing*, 60(6):2882–2898, 2012.

[19] N. Guan, D. Tao, Z. Luo, and B. Yuan. Online nonnegative matrix factorization with robust stochastic approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1087–1099, 2012.

[20] P. Jain, B. Kulis, and I. S. Dhillon. Inductive regularized learning of kernel functions. In *NIPS*, pages 946–954, 2010.

[21] M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization (SIOPT)*, 20:2327–2351, May 2010.

[22] B. Kulis, M. A. Sustik, and I. S. Dhillon. Low-rank kernel learning with bregman matrix divergences. *JMLR*, 10:341–376, 2009.

[23] D. Lee, H. Seung, et al. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[24] J. R. Magnus and H. Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 2nd edition, 1999.

[25] G. Meyer, S. Bonnabel, and R. Sepulchre. Regression on fixed-rank positive semidefinite matrices: A riemannian approach. *JMLR*, 12:593–625, 2011.

[26] Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108:177–205, August 2006.

[27] R. Pytlak. Conjugate gradient algorithms in nonconvex optimization. *Springer Press*, 89, 2009.

[28] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719, 2005.

[29] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *Carnegie Mellon University Technical Report*, 1994.

[30] S. Wang and R. Jin. An information geometry approach for distance metric learning. *AISTATS*, 5:591–598, 2009.

[31] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009.

[32] Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3):203–230, 2010.

[33] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, pages 682–688. MIT Press, 2001.

[34] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved nyström low-rank approximation and error analysis. In *ICML*, pages 1232–1239, 2008.

[35] T. Zhou and D. Tao. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *ICML*, pages 33–40, 2011.