Journal of Machine Learning Research 10 (2009) 341-376 Submitted 8/07; Revised 10/08; Published 2/09

Low-Rank Kernel Learning with Bregman Matrix Divergences

Brian Kulis KULIS@CS.UTEXAS.EDU
Mátyás A. Sustik SUSTIK@CS.UTEXAS.EDU
Inderjit S. Dhillon INDERJIT@CS.UTEXAS.EDU
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA

Editor: Michael I. Jordan

Abstract

In this paper, we study low-rank matrix nearness problems, with a focus on learning low-rank positive semidefinite (kernel) matrices for machine learning applications. We propose efficient algorithms that scale linearly in the number of data points and quadratically in the rank of the input matrix. Existing algorithms for learning kernel matrices often scale poorly, with running times that are cubic in the number of data points. We employ Bregman matrix divergences as the measures of nearness—these divergences are natural for learning low-rank kernels since they preserve rank as well as positive semidefiniteness. Special cases of our framework yield faster algorithms for various existing learning problems, and experimental results demonstrate that our algorithms can effectively learn both low-rank and full-rank kernel matrices.

Keywords: kernel methods, Bregman divergences, convex optimization, kernel learning, matrix nearness

1. Introduction

Underlying many machine learning algorithms is a measure of distance, or divergence, between data objects. A number of factors affect the choice of the particular divergence measure used: the data may be intrinsically suited for a specific divergence measure, an algorithm may be faster or easier to implement for some measure, or analysis may be simpler for a particular measure. For example, the KL-divergence is popular when the data is represented as discrete probability vectors (i.e., non-negative vectors whose sum is 1), and the ℓ_1-norm distance is often used when sparsity is desired.

When measuring the divergence between matrices, similar issues must be considered. Typically, matrix norms such as the Frobenius norm are used, but these measures are not appropriate for all problems. Analogous to the KL-divergence for vectors, when the matrices under consideration are positive semidefinite (i.e., they have non-negative eigenvalues), then one may want to choose a divergence measure that is well-suited to such matrices; positive semidefinite (PSD) matrices arise frequently in machine learning, in particular with kernel methods. Existing learning techniques involving PSD matrices often employ matrix norms, but their use requires an additional constraint that the matrix stay positive semidefinite, which ultimately leads to algorithms involving expensive eigenvector computation. Kernel alignment, a measure of similarity between PSD matrices used in some kernel learning algorithms (Cristianini et al., 2002), also requires explicit enforcement of positive definiteness. On the other hand, the two main divergences used in this paper are defined only

©2009 Brian Kulis, Mátyás A. Sustik and Inderjit S. Dhillon.

over the cone of positive semidefinite matrices, and our algorithms lead to automatic enforcement of positive semidefiniteness.

This paper focuses on kernel learning using two divergence measures between PSD matrices—the LogDet divergence and the von Neumann divergence. Our kernel learning goal is to find a PSD matrix which is as close as possible (under the LogDet or von Neumann divergence) to some input PSD matrix, but which additionally satisfies linear equality or inequality constraints. We argue that these two divergences are natural for problems involving positive definite matrices. First, they have several properties which make optimization computationally efficient and useful for a variety of learning tasks. For example, the LogDet divergence has a scale-invariance property which makes it particularly well-suited to machine learning applications. Second, these divergences arise naturally in several problems; for example, the LogDet divergence arises in problems involving the differential relative entropy between Gaussian distributions while the von Neumann divergence arises in quantum computing and problems such as online PCA.

One of the key properties that we demonstrate and exploit is that, for low-rank matrices, the divergences enjoy a *range-space preserving* property. That is, the LogDet divergence between two matrices is finite if and only if their range spaces are identical, and a similar property holds for the von Neumann divergence. This property leads to simple projection-based algorithms for learning PSD (or kernel) matrices that preserve the rank of the input matrix—if the input matrix is low-rank then the resulting learned matrix will also be low-rank. These algorithms are efficient; they scale linearly with the number of data points n and quadratically with the rank of the input matrix. The efficiency of the algorithms arises from new results developed in this paper which demonstrate that Bregman projections for these matrix divergences can be computed in time that scales quadratically with the rank of the input matrix. Our algorithms stand in contrast to previous work on learning kernel matrices, which scale as $O(n^3)$ or worse, relying on semidefinite programming or repeated eigenvector computation. We emphasize that our methods *preserve* rank, so they can learn low-rank kernel matrices when the input kernel matrix has low rank; however our methods do not decrease rank so they are not applicable to the non-convex problem of finding a low-rank solution given a full (or higher) rank input kernel matrix.

One special case of our method is the DefiniteBoost optimization problem from Tsuda et al. (2005); our analysis shows how to improve the running time of their algorithm by a factor of n, from $O(n^3)$ time per projection to $O(n^2)$. This projection is also used in online-PCA (Warmuth and Kuzmin, 2006), and we obtain a factor of n speedup for that problem as well. In terms of experimental results, a direct application of our techniques is in learning low-rank kernel matrices in the setting where we have background information for a subset of the data; we discuss and experiment with learning low-rank kernel matrices for classification and clustering in this setting, demonstrating that we can scale to large data sets. We also discuss the use of our divergences for learning Mahalanobis distance functions, which allows us to move beyond the transductive setting and generalize to new points. In this vein, we empirically compare our methods to existing metric learning algorithms; we are particularly interested in large-scale applications, and discuss some recent applications of the algorithms developed in this paper to computer vision tasks.

2. Background and Related Work

In this section, we briefly review relevant background material and related work.

2.1 Kernel Methods

Given a set of training points $\boldsymbol{a}_1, ..., \boldsymbol{a}_n$, a common step in kernel algorithms is to transform the data using a nonlinear function ψ. This mapping, typically, represents a transformation of the data to a higher-dimensional feature space. A kernel function κ gives the inner product between two vectors in the feature space:

$$\kappa(\boldsymbol{a}_i, \boldsymbol{a}_j) = \psi(\boldsymbol{a}_i) \cdot \psi(\boldsymbol{a}_j).$$

It is often possible to compute this inner product without explicitly computing the expensive mapping of the input points to the higher-dimensional feature space. Generally, given n points $\boldsymbol{a}_i$, we form an $n \times n$ matrix K, called the kernel matrix, whose (i, j) entry corresponds to $\kappa(\boldsymbol{a}_i, \boldsymbol{a}_j)$. In kernel-based algorithms, the only information needed about the input data points is the inner products; hence, the kernel matrix provides all relevant information for learning in the feature space. A kernel matrix formed from any set of input data points is always positive semidefinite. See Shawe-Taylor and Cristianini (2004) for more details.

2.2 Low-Rank Kernel Representations and Kernel Learning

Despite the popularity of kernel methods in machine learning, many kernel-based algorithms scale poorly; low-rank kernel representations address this issue. Given an $n \times n$ kernel matrix K, if the matrix is of low rank, say $r < n$, we can represent the kernel matrix in terms of a factorization $K = GG^T$, with G an $n \times r$ matrix.

In addition to easing the burden of memory overhead from $O(n^2)$ storage to $O(nr)$, this low-rank decomposition can lead to improved efficiency. For example, Fine and Scheinberg (2001) show that SVM training reduces from $O(n^3)$ to $O(nr^2)$ when using a low-rank decomposition. Empirically, the algorithm in Fine and Scheinberg (2001) outperforms other SVM training algorithms in terms of training time by several factors. In clustering, the kernel k-means algorithm (Dhillon et al., 2004) has a running time of $O(n^2)$ per iteration, which can be improved to $O(nrc)$ time per iteration with a low-rank kernel representation, where c is the number of desired clusters. Low-rank kernel representations are often obtained using incomplete Cholesky decompositions (Fine and Scheinberg, 2001). Recently, work has been done on using labeled data to improve the quality of low-rank decompositions (Bach and Jordan, 2005).

Low-rank decompositions have also been employed for solving a number of other machine learning problems. For example, in Kulis et al. (2007b), low-rank decompositions were employed for clustering and embedding problems. In contrast to work in this paper, their focus was on using low-rank decompositions to develop algorithms for such problems as k-means and maximum variance unfolding. Other examples of using low-rank decompositions to speed up machine learning algorithms include Weinberger et al. (2006) and Torresani and Lee (2006).

In this paper, our focus is on using distance and similarity constraints to learn a low-rank kernel matrix. In related work, Lanckriet et al. (2004) have studied transductive learning of the kernel matrix and multiple kernel learning using semidefinite programming. In Kwok and Tsang (2003), a formulation based on idealized kernels is presented to learn a kernel matrix when some labels are given. Another recent paper (Weinberger et al., 2004) considers learning a kernel matrix for nonlinear dimensionality reduction; like much of the research on learning a kernel matrix, they use semidefinite programming and the running time is at least cubic in the number of data points. Our work is closest to that of Tsuda et al. (2005), who learn a (full-rank) kernel matrix using von Neumann divergence under linear constraints. However, our framework is more general and our

emphasis is on low-rank kernel learning. Our algorithms are more efficient than those of Tsuda et al.; we use exact instead of approximate projections to speed up convergence, and we consider algorithms for the LogDet divergence in addition to the von Neumann divergence. For the case of the von Neumann divergence, our algorithm also corrects a mistake in Tsuda et al. (2005); see Appendix B for details.

An earlier version of our work appeared in Kulis et al. (2006). In this paper, we substantially expand on the analysis of Bregman matrix divergences, giving a formal treatment of low-rank Bregman matrix divergences and proving several new properties. We also present further algorithmic analysis for the LogDet and von Neumann algorithms, provide connections to semidefinite programming and metric learning, and present additional experiments, including ones on much larger data sets.

3. Optimization Framework

We begin with an overview of the optimization framework applied to the problem studied in this paper. We introduce Bregman matrix divergences—the matrix divergence measures considered in this paper—and discuss their properties. Then we overview Bregman's method, the optimization algorithm that is used to learn low-rank kernel matrices.

3.1 Bregman Matrix Divergences

To measure the nearness between two matrices, we will use Bregman matrix divergences, which are generalizations of Bregman vector divergences. Let φ be a real-valued strictly convex function defined over a convex set $S = \mathrm{dom}(\varphi) \subseteq \mathbb{R}^m$ such that φ is differentiable on the relative interior of S. The *Bregman vector divergence* (Bregman, 1967) with respect to φ is defined as

$$D_\varphi(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x}) - \varphi(\boldsymbol{y}) - (\boldsymbol{x} - \boldsymbol{y})^T \nabla\varphi(\boldsymbol{y}).$$

For example, if $\varphi(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$, then the resulting Bregman divergence is $D_\varphi(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2^2$. Another example is $\varphi(\boldsymbol{x}) = \sum_i (x_i \log x_i - x_i)$, where the resulting Bregman divergence is the (unnormalized) relative entropy $D_\varphi(\boldsymbol{x}, \boldsymbol{y}) = KL(\boldsymbol{x}, \boldsymbol{y}) = \sum_i (x_i \log \frac{x_i}{y_i} - x_i + y_i)$. Bregman divergences generalize many properties of squared loss and relative entropy. See Censor and Zenios (1997) for more details.

We can naturally extend this definition to real, symmetric $n \times n$ matrices, denoted by S^n. Given a strictly convex, differentiable function $\phi : S^n \to \mathbb{R}$, the Bregman matrix divergence is defined to be

$$D_\phi(X, Y) = \phi(X) - \phi(Y) - \mathrm{tr}((\nabla\phi(Y))^T (X - Y)),$$

where $\mathrm{tr}(A)$ denotes the trace of matrix A. Examples include $\phi(X) = \|X\|_F^2$, which leads to the well-known squared Frobenius norm $\|X - Y\|_F^2$. In this paper, we will extensively study two less well-known divergences. Let ϕ be the entropy of the eigenvalues of a positive definite matrix. Specifically, if X has eigenvalues $\lambda_1, \ldots, \lambda_n$, let $\phi(X) = \sum_i (\lambda_i \log \lambda_i - \lambda_i)$, which may be expressed as $\phi(X) = \mathrm{tr}(X \log X - X)$, where $\log X$ is the matrix logarithm.[1] The resulting Bregman divergence is

$$D_{vN}(X, Y) = \mathrm{tr}(X \log X - X \log Y - X + Y), \tag{1}$$

1. If $X = V\Lambda V^T$ is the eigendecomposition of the positive definite matrix X, the matrix logarithm can be written as $V \log\Lambda V^T$, where $\log\Lambda$ is the diagonal matrix whose entries contain the logarithm of the eigenvalues. The matrix exponential can be defined analogously.

and we call it the von Neumann divergence. This divergence is also called quantum relative entropy, and is used in quantum information theory (Nielsen and Chuang, 2000). Another important matrix divergence arises by taking the Burg entropy of the eigenvalues, that is, $\phi(X) = -\sum_i \log \lambda_i$, or equivalently as $\phi(X) = -\log\det X$. The resulting Bregman divergence over positive definite matrices is

$$D_{\ell d}(X,Y) = \operatorname{tr}(XY^{-1}) - \log\det(XY^{-1}) - n, \tag{2}$$

and is commonly called the LogDet divergence (we called it the Burg matrix divergence in Kulis et al. (2006)). For now, we assume that X is positive definite for both divergences; later we will discuss extensions when X is positive semidefinite, that is, low-rank.

3.2 Properties

It is important to justify the use of the divergences introduced above for kernel learning, so we now discuss some important properties of the divergences.

The most obvious computational benefit of using the divergences for kernel learning arises from the fact that they are defined only over positive definite matrices. Because of this, our algorithms will not need to explicitly constrain our learned matrices to be positive definite, which in turn leads to efficient algorithms. Beyond automatic enforcement of positive definiteness, we will see later that the divergences have a range-space preserving property, which leads to further computational benefits.

Appendix A covers some important properties of the divergences. Examples include the scale-invariance of the LogDet divergence ($D_{\ell d}(X,Y) = D_{\ell d}(\alpha X, \alpha Y)$) and, more generally, transformation-invariance ($D_{\ell d}(X,Y) = D_{\ell d}(M^T XM, M^T YM)$ for any square, non-singular M), which are useful properties for learning algorithms. For instance, if we have applied a linear kernel over a set of data vectors, scale-invariance implies that we can scale all the features of our data vectors, and our learned kernel will simply be scaled by the same factor. Similarly, if we scale each of the features in our data vectors differently (for example, if some features are measured in feet and others in meters as opposed to all features measured in feet), then the learned kernel will be scaled by an appropriate diagonal transformation. Note that this natural property does not hold for loss functions such as the Frobenius norm distance.

Another critical property concerns generalization to new points. Typically kernel learning algorithms work in the transductive setting, meaning that all of the data is given up-front, with some of it labeled and some unlabeled, and one learns a kernel matrix over all the data points. If some new data point is given, there is no way to compare it to existing data points, so a new kernel matrix must be learned. However, as has recently been shown in Davis et al. (2007), we can move beyond the transductive setting when using the LogDet divergence, and can evaluate the kernel function over new data points. A similar result can be shown for the von Neumann divergence. We overview this recent work in Section 6.1, though the main focus in this paper remains in the transductive setting.

Furthermore, both the LogDet divergence and the von Neumann divergence have precedence in a number of areas. The von Neumann divergence is used in quantum information theory (Nielsen and Chuang, 2000), and has been employed in machine learning for online principal component analysis (Warmuth and Kuzmin, 2006). The LogDet divergence is called Stein's loss in the statistics literature, where it has been used as a measure of distance between covariance matrices (James and Stein, 1961). It has also been employed in the optimization community; the updates for the BFGS and DFP algorithms (Fletcher, 1991), both quasi-Newton algorithms, can be viewed as LogDet

optimization programs. In particular, the update to the approximation of the Hessian given in these algorithms is the result of a LogDet minimization problem with linear constraints (given by the secant equation).

For all three divergences introduced above, the generating convex function of the Bregman matrix divergence can be viewed as a composition $\phi(X) = (\varphi \circ \lambda)(X)$, where $\lambda(X)$ is the function that lists the eigenvalues in algebraically decreasing order, and φ is a strictly convex function defined over vectors (Dhillon and Tropp, 2007). In general, every such φ defines a Bregman matrix divergence over real, symmetric matrices via the eigenvalue mapping. For example, if $\varphi(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$, then the resulting composition $(\varphi \circ \lambda)(X)$ is the squared Frobenius norm. We call such divergences *spectral Bregman matrix divergences*.

Consider a spectral Bregman matrix divergence $D_\phi(X,Y)$, where $\phi = \varphi \circ \lambda$. We now show an alternate expression for $D_\phi(X,Y)$ based on the eigenvalues and eigenvectors of X and Y, which will prove to be useful when motivating extensions to the low-rank case.

Lemma 1 *Let the eigendecompositions of X and Y be $V\Lambda V^T$ and $U\Theta U^T$ respectively, and assume that φ is* separable, *that is, $\phi(X) = (\varphi \circ \lambda)(X) = \sum_i \varphi_i(\lambda_i)$. Then*

$$D_\phi(X,Y) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 (\varphi_i(\lambda_i) - \varphi_j(\theta_j) - (\lambda_i - \theta_j)\nabla\varphi_j(\theta_j)).$$

Proof We have

$$\begin{aligned} D_\phi(X,Y) &= \phi(X) - \phi(Y) - \mathrm{tr}((X-Y)^T(\nabla\phi(Y))) \\ &= \sum_i \varphi_i(\lambda_i) - \sum_j \varphi_j(\theta_j) - \mathrm{tr}((V\Lambda V^T - U\Theta U^T)^T(\nabla\phi(Y))) \\ &= \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \varphi_i(\lambda_i) - \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \varphi_j(\theta_j) - \mathrm{tr}((V\Lambda V^T - U\Theta U^T)^T(\nabla\phi(Y))). \end{aligned}$$

The second line above uses the separability of φ, while the third line uses the fact that $\sum_i (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 = \sum_j (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 = 1$. We can express $\nabla\phi(Y)$ as:

$$\nabla\phi(Y) = U \begin{pmatrix} \nabla\varphi_1(\theta_1) & 0 & \dots \\ 0 & \nabla\varphi_2(\theta_2) & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} U^T,$$

and so we have $\mathrm{tr}(U\Theta U^T \nabla\phi(Y)) = \sum_j \theta_j \nabla\varphi_j(\theta_j) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \theta_j \nabla\varphi_j(\theta_j)$. Finally, the term $\mathrm{tr}(V\Lambda V^T \nabla\phi(Y))$ can be expanded as $\sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \lambda_i \nabla\varphi_j(\theta_j)$. Putting this all together, we have:

$$D_\phi(X,Y) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 (\varphi_i(\lambda_i) - \varphi_j(\theta_j) - \nabla\varphi_j(\theta_j) \cdot (\lambda_i - \theta_j)).$$

■

Note that each of the divergences discussed earlier—the squared Frobenius divergence, the LogDet divergence, and the von Neumann divergence—arise from separable convex functions. Furthermore, in these three cases, the functions φ_i do not depend on i (so we denote $\varphi = \varphi_1 = \ldots = \varphi_n$) and the corollary below follows:

Corollary 2 *Given $X = V\Lambda V^T$ and $Y = U\Theta U^T$, the squared Frobenius, von Neumann and LogDet divergences satisfy:*

$$D_\phi(X,Y) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 D_\varphi(\lambda_i, \theta_j). \tag{3}$$

This formula highlights the connection between a separable spectral matrix divergence and the corresponding vector divergence. It also illustrates how the divergence relates to the geometrical properties of the argument matrices as represented by the eigenvectors.

3.3 Problem Description

We now give a formal statement of the problem that we study in this paper. Given an input kernel matrix X_0, we attempt to solve the following for X:

$$\begin{array}{ll} \text{minimize} & D_\phi(X, X_0) \\ \text{subject to} & \text{tr}(XA_i) \leq b_i,\ 1 \leq i \leq c, \\ & \text{rank}(X) \leq r, \\ & X \succeq 0. \end{array} \tag{4}$$

Any of the linear inequality constraints $\text{tr}(XA_i) \leq b_i$ may be replaced with equalities. The above problem is clearly non-convex in general, due to the rank constraint. However, when the rank of X_0 does not exceed r, then this problem turns out to be convex when we use the von Neumann and LogDet matrix divergences. This is because these divergences restrict the search for the optimal X to the linear subspace of matrices that have the same range space as X_0. The details will emerge in Section 4. Another advantage of using the von Neumann and LogDet divergences is that the algorithms used to solve the minimization problem implicitly maintain the positive semidefiniteness constraint, so no extra work needs to be done to enforce positive semidefiniteness. This is in contrast to the squared Frobenius divergence, where the positive semidefiniteness constraint has to be explicitly enforced.

Though it is possible to handle general linear constraints of the form $\text{tr}(XA_i) \leq b_i$, we will focus on two specific types of constraints, which will be useful for our kernel learning applications. The first is a distance constraint. The squared Euclidean distance in feature space between the jth and the kth data points is given by $X_{jj} + X_{kk} - 2X_{jk}$. The constraint $X_{jj} + X_{kk} - 2X_{jk} \leq b_i$ can be represented as $\text{tr}(XA) \leq b_i$, where $A_i = \boldsymbol{z}_i \boldsymbol{z}_i^T$, and $\boldsymbol{z}_i(j) = 1$, $\boldsymbol{z}_i(k) = -1$, and all other entries 0 (equivalently, $\boldsymbol{z}_i = \boldsymbol{e}_j - \boldsymbol{e}_k$). The second type of constraint has the form $X_{jk} \leq b_i$, which can be written as $\text{tr}(XA_i) \leq b_i$ using $A_i = \frac{1}{2}(\boldsymbol{e}_j \boldsymbol{e}_k^T + \boldsymbol{e}_k \boldsymbol{e}_j^T)$ (we maintain symmetry of A_i).

For the algorithms developed in Section 5, we will assume that the matrices A_i in (4) are rank one (so $A_i = \boldsymbol{z}_i \boldsymbol{z}_i^T$) and that $r = \text{rank}(X_0)$ (we briefly discuss extensions to higher-rank constraints in Section 6.2.2). In this case, we can write the optimization problem as:

$$\begin{array}{ll} \text{minimize} & D_\phi(X, X_0) \\ \text{subject to} & \boldsymbol{z}_i^T X \boldsymbol{z}_i \leq b_i,\ 1 \leq i \leq c, \\ & \text{rank}(X) \leq r, \\ & X \succeq 0. \end{array} \tag{5}$$

Furthermore, we assume that there exists a feasible solution to the above problem; we discuss an extension to the infeasible case involving slack variables in Section 6.2.1. Note that in this case, we assume $b_i \geq 0$, as otherwise there cannot be a feasible solution.

The key application of the above optimization problem is in learning a kernel matrix in the setting where we have side information about some of the data points (e.g., labels or constraints), and we want to learn a kernel matrix over all the data points in order to perform classification, regression, or semi-supervised clustering. We will also discuss other applications throughout the paper.

3.4 Bregman Projections

Consider the convex optimization problem (4) presented above, but without the rank constraint (we will see how to handle the rank constraint later). To solve this problem, we use the method of Bregman projections (Bregman, 1967; Censor and Zenios, 1997). Suppose we wish to minimize $D_\phi(X, X_0)$ subject to linear equality and inequality constraints. The idea behind Bregman projections is to choose one constraint per iteration, and perform a projection so that the current solution satisfies the chosen constraint. Note that the projection is not an orthogonal projection, but rather a *Bregman projection*, which is tailored to the particular function that is being minimized. In the case of inequality constraints, an appropriate correction is also enforced. This process is then repeated by cycling through the constraints (or employing a more sophisticated control mechanism). This method may also be viewed as a dual coordinate ascent procedure that optimizes the dual with respect to a single dual variable per iteration (with all other dual variables remaining fixed). Under mild conditions, it can be shown that the method of cyclic Bregman projections (or a control mechanism that visits each constraint infinitely often) converges to the globally optimal solution; see Censor and Zenios (1997) and Dhillon and Tropp (2007) for more details.

Now for the details of each iteration. For an equality constraint of the form $\mathrm{tr}(XA_i) = b_i$, the Bregman projection of the current iterate X_t may be computed by solving:

$$\begin{aligned} \text{minimize}_X \quad & D_\phi(X, X_t) \\ \text{subject to} \quad & \mathrm{tr}(XA_i) = b_i. \end{aligned} \tag{6}$$

Introduce the dual variable α_i, and form the Lagrangian $\mathcal{L}(X, \alpha_i) = D_\phi(X, X_t) + \alpha_i(b_i - \mathrm{tr}(XA_i))$. By setting the gradient of the Lagrangian (with respect to X and α_i) to zero, we can obtain the Bregman projection by solving the resulting system of equations simultaneously for α_i and X_{t+1}:

$$\begin{aligned} \nabla\phi(X_{t+1}) &= \nabla\phi(X_t) + \alpha_i A_i \\ \mathrm{tr}(X_{t+1}A_i) &= b_i. \end{aligned} \tag{7}$$

For an inequality constraint of the form $\mathrm{tr}(XA_i) \leq b_i$, let $\nu_i \geq 0$ be the corresponding dual variable. To maintain non-negativity of the dual variable (which is necessary for satisfying the KKT conditions), we can solve (7) for α_i and perform the following update:

$$\alpha_i' = \min(\nu_i, \alpha_i), \;\; \nu_i \leftarrow \nu_i - \alpha_i'. \tag{8}$$

See Appendix B for a discussion on why the dual variable corrections are needed. Clearly the update guarantees that $\nu_i \geq 0$. Finally, update X_{t+1} via

$$\nabla\phi(X_{t+1}) = \nabla\phi(X_t) + \alpha_i' A_i. \tag{9}$$

Note that (8) may be viewed as a correction step that follows the projection given by (7). Both of our algorithms in Section 5 are based on this method, which iteratively chooses a constraint and performs a Bregman projection until convergence. The main difficulty lies in efficiently solving the nonlinear system of equations given in (7).

In the case where $A_i = z_i z_i^T$, by calculating the gradient for the LogDet and von Neumann matrix divergences, respectively, (7) simplifies to:

$$\begin{aligned} X_{t+1} &= \left(X_t^{-1} - \alpha_i z_i z_i^T\right)^{-1} \\ X_{t+1} &= \exp(\log(X_t) + \alpha_i z_i z_i^T), \end{aligned} \tag{10}$$

subject to $z_i^T X_{t+1} z_i = b_i$. In (10), exp and log denote the matrix exponential and matrix logarithm, respectively. As we will see, for the von Neumann and LogDet divergences, these projections can be computed very efficiently (and are thus more desirable than other methods that involve multiple constraints at a time).

4. Bregman Divergences for Rank-Deficient Matrices

As given in (1) and (2), the von Neumann and LogDet divergences are undefined for low-rank matrices. For the LogDet divergence, the convex generating function $\phi(X) = -\log\det X$ is infinite when X is singular, that is, its effective domain is the set of positive definite matrices. For the von Neumann divergence the situation is somewhat better, since one can define $\phi(X) = \mathrm{tr}(X\log X - X)$ via continuity for rank-deficient matrices.

The key to using these divergences in the low-rank setting comes from restricting the convex function ϕ to the range spaces of the matrices. We motivate our approach using Corollary 2, before formalizing it in Section 4.2. Subsequently, we discuss the computation of Bregman projections for low-rank matrices in Section 4.3.

4.1 Motivation

If X and Y have eigendecompositions $X = V\Lambda V^T$ and $Y = U\Theta U^T$, respectively, then whenever X or Y is of low-rank, some eigenvalues λ_i of X or θ_j of Y are equal to zero. Consequently, if we could apply Corollary 2, the $D_\varphi(\lambda_i, \theta_j)$ terms that involve zero eigenvalues need careful treatment. More specifically, for the von Neumann divergence we have:

$$D_\varphi(\lambda_i, \theta_j) = \lambda_i \log\lambda_i - \lambda_i \log\theta_j - \lambda_i + \theta_j. \tag{11}$$

Using the convention that $0\log 0 = 0$, $D_\varphi(\lambda_i, \theta_j)$ equals 0 when both λ_i and θ_j are 0, but is infinite when θ_j is 0 but λ_i is non-zero. Similarly, with the LogDet divergence, we have

$$D_\varphi(\lambda_i, \theta_j) = \frac{\lambda_i}{\theta_j} - \log\frac{\lambda_i}{\theta_j} - 1. \tag{12}$$

In cases where $\lambda_i = 0$ and $\theta_j \neq 0$, or $\lambda_i \neq 0$ and $\theta_j = 0$, $D_\varphi(\lambda_i, \theta_j)$ is infinite.

For finiteness of the corresponding matrix divergence we require that $v_i^T u_j = 0$ whenever $D_\varphi(\lambda_i, \theta_j)$ is infinite so a cancellation will occur (via appropriate continuity arguments) and the divergence will be finite. This leads to properties of rank-deficient X and Y that are required for the matrix divergence to be finite. The following observations are discussed more formally in Section 4.2.

Observation 1 *The von Neumann divergence $D_{vN}(X,Y)$ is finite iff* $\text{range}(X) \subseteq \text{range}(Y)$.

The term $\lambda_i \log \theta_j$ from (11) is $-\infty$ if $\theta_j = 0$ but $\lambda_i \neq 0$. By using Corollary 2 and distributing the $(\boldsymbol{v}_i^T \boldsymbol{u}_j)^2$ term into the scalar divergence, we obtain: $\lambda_i (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \log \theta_j$. Thus, if $\boldsymbol{v}_i^T \boldsymbol{u}_j = 0$ when $\theta_j = 0$, then $\lambda_i (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \log \theta_j = 0$ (using $0 \log 0 = 0$), and the divergence is finite. When $\theta_j = 0$, the corresponding eigenvector $\boldsymbol{u}_j$ is in the null space of Y; therefore, for finiteness of the divergence, every vector $\boldsymbol{u}_j$ in the null space of Y is orthogonal to any vector $\boldsymbol{v}_i$ in the range space of X. This implies that the null space of Y contains the null space of X or, equivalently, $\text{range}(X) \subseteq \text{range}(Y)$.

Observation 2 *The LogDet divergence $D_{\ell d}(X,Y)$ is finite iff* $\text{range}(X) = \text{range}(Y)$.

To show the observation, we adopt the conventions that $\log \frac{0}{0} = 0$ and $\frac{0}{0} = 1$ in (12), which follow by continuity assuming that the rate at which the numerator and denominator approach zero is the same. Then, distributing the $(\boldsymbol{v}_i^T \boldsymbol{u}_j)^2$ term into D_φ, we have that $\boldsymbol{v}_i$ and $\boldsymbol{u}_j$ must be orthogonal whenever $\lambda_i = 0, \theta_j \neq 0$ or $\lambda_i \neq 0, \theta_j = 0$. This in turn says that: a) every vector $\boldsymbol{u}_j$ in the null space of Y must be orthogonal to every vector $\boldsymbol{v}_i$ in the range space of X and, b) every vector $\boldsymbol{v}_i$ in the null space of X must be orthogonal to every vector $\boldsymbol{u}_j$ in the range space of Y. It follows that the range spaces of X and Y are equal.

Assuming the eigenvalues of X and Y are listed in non-increasing order, we can write the low-rank equivalent of (3) simply as:

$$D_\phi(X,Y) = \sum_{i,j \leq r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 D_\varphi(\lambda_i, \theta_j),$$

where $r = \text{rank}(X)$.

If we now revisit our optimization problem formulated in (4), where we aim to minimize $D_\phi(X, X_0)$, we see that the LogDet and von Neumann divergences naturally maintain rank constraints if the problem is feasible. For the LogDet divergence, the equality of the range spaces of X and X_0 implies that when minimizing $D_\phi(X, X_0)$, we maintain $\text{rank}(X) = \text{rank}(X_0)$, assuming that there is a feasible X with a finite $D_\phi(X, X_0)$. Similarly, for the von Neumann divergence, the property that $\text{range}(X) \subseteq \text{range}(X_0)$ implies $\text{rank}(X) \leq \text{rank}(X_0)$.

4.2 Formalization Via Restrictions on the Range Space

The above section demonstrated informally that for $D_\phi(X,Y)$ to be finite the range spaces of X and Y must be equal for the LogDet divergence, and the range space of X must be contained in the range space of Y for the von Neumann divergence.

We now formalize the generalization to low-rank matrices. Let W be an orthogonal $n \times r$ matrix, such that its columns span the range space of Y. We will use the following simple and well known lemma later on:

Lemma 3 *Let Y be a symmetric $n \times n$ matrix with* $\text{rank}(Y) \leq r$*, and let W be an $n \times r$ column orthogonal matrix ($W^T W = I$) with* $\text{range}(Y) \subseteq \text{range}(W)$*. Then $Y = WW^T YWW^T$.*

Proof Extend W to a full $n \times n$ orthogonal matrix and denote it by W_f. Notice that the last $(n-r)$ rows of $W_f^T Y$ and the last $(n-r)$ columns of YW_f consist of only zeros. It follows that the matrix $Y_1 = W_f^T Y W_f$ contains only zeros except for the $r \times r$ sub-matrix in the top left corner, which coincides with $\hat{Y} = W^T Y W$. Observe that $Y = W_f Y_1 W_f^T = W \hat{Y} W^T = WW^T YWW^T$ to finish the proof. ■

We are now ready to extend the domain of the von Neumann and LogDet divergences to low-rank matrices.

Definition 4 *Consider the positive semidefinite $n \times n$ matrices X and Y that satisfy* $\text{range}(X) \subseteq \text{range}(Y)$ *when considering the von Neumann divergence, and* $\text{range}(X) = \text{range}(Y)$ *when considering the LogDet divergence. Let W be an $n \times r$ column orthogonal matrix such that range$(Y) \subseteq$ range(W) and define:*

$$D_\phi(X,Y) = D_{\phi,W}(X,Y) = D_\phi(W^T X W, W^T Y W), \tag{13}$$

where D_ϕ is either the von Neumann or the LogDet divergence.

The first equality in (13) implicitly assumes that the right hand side is not dependent on the choice of W.

Lemma 5 *In Definition 4, $D_{\phi,W}(X,Y)$ is independent of the choice of W.*

Proof All the $n \times r$ orthogonal matrices with the same range space as W can be expressed as a product WQ, where Q is an $r \times r$ orthogonal matrix. Introduce $\hat{X} = W^T X W$ and $\hat{Y} = W^T Y W$ and substitute WQ in place of W in (13):

$$\begin{aligned} D_{\phi,WQ}(X,Y) &= D_\phi((WQ)^T X (WQ), (WQ)^T Y (WQ)) \\ &= D_\phi(Q^T \hat{X} Q, Q^T \hat{Y} Q) \\ &= D_\phi(\hat{X}, \hat{Y}) = D_{\phi,W}(X,Y). \end{aligned}$$

The first equality is by Definition 4, and the third follows from the fact that the von Neumann and the LogDet divergences are invariant under any orthogonal similarity transformation[2] (see Proposition 11 in Appendix A). ■

We now show that Definition 4 is consistent with Corollary 2, demonstrating that our low-rank formulation agrees with the informal discussion given earlier.

Theorem 6 *Let the positive semidefinite matrices X and Y have eigendecompositions $X = V \Lambda V^T$, $Y = U \Theta U^T$ and let* $\text{range}(X) \subseteq \text{range}(Y)$. *Let the rank of Y equal r. Assuming that the eigenvalues of X and Y are sorted in non-increasing order, that is, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r$ and $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_r$, then*

$$D_{vN}(X,Y) = \sum_{i,j \leq r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 (\lambda_i \log \lambda_i - \lambda_i \log \theta_j - \lambda_i + \theta_j).$$

Proof Denote the upper left $r \times r$ submatrices of Λ and Θ by Λ_r and Θ_r respectively, and the corresponding reduced eigenvector matrices for X and Y by V_r and U_r. Picking W in (13) to equal U_r, we get:

$$D_{vN}(X,Y) = D_{vN}(U_r^T X U_r, U_r^T Y U_r) = D_{vN}((U_r^T V_r) \Lambda_r (V_r^T U_r), \Theta_r).$$

The arguments on the right hand side are $r \times r$ matrices and Θ_r is not rank-deficient. We can now apply Corollary 2 to get the result. ■

2. In fact, in the case of the LogDet divergence we have invariance under any invertible congruence transformation, see Proposition 12 in Appendix A.

Theorem 7 *Let the positive semidefinite matrices X and Y have eigendecompositions $X = V\Lambda V^T$, $Y = U\Theta U^T$, and let* $\text{range}(X) = \text{range}(Y)$ *and assume that the eigenvalues of X and Y are sorted in decreasing order. Then:*

$$D_{\ell d}(X,Y) = \sum_{i,j \le r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \left(\frac{\lambda_i}{\theta_j} - \log \frac{\lambda_i}{\theta_j} - 1 \right).$$

Proof Similar to the proof of Theorem 6. Note that the range spaces must coincide in this case, because the determinant of XY^{-1} should not vanish for the restricted transformations, which agrees with the $\text{range}(X) = \text{range}(Y)$ restriction. ■

We next show that the optimization problem and Bregman's projection algorithm for low-rank matrices can be cast as a full rank problem in a lower dimensional space, namely the range space. This equivalence implies that we do not have to rederive the convergence proofs and other properties of Bregman's algorithm in the low-rank setting.

Consider the optimization problem (4) for low-rank X_0, and denote a suitable orthogonal matrix as required in Definition 4 by W. The matrix of the eigenvectors of the reduced eigendecomposition of X_0 is a suitable choice. Consider the following matrix mapping:

$$M \longrightarrow \hat{M} = W^T M W.$$

By Lemma 3, the mapping is one-to-one on the set of symmetric matrices with range space contained in $\text{range}(W)$. We now apply the mapping to all matrices in the optimization problem (4) to obtain:

$$\begin{array}{ll} \text{minimize} & D_\phi(\hat{X}, \hat{X}_0) \\ \text{subject to} & \text{tr}(\hat{X}\hat{A}_i) \le b_i,\ 1 \le i \le c \\ & \text{rank}(\hat{X}) \le r \\ & \hat{X} \succeq 0. \end{array} \tag{14}$$

The rank constraint is automatically satisfied when $\text{rank}(X_0) = r$ and the problem is feasible. Clearly, $\hat{X} \succeq 0$ if and only if $X \succeq 0$. By Definition 4, $D_\phi(\hat{X}, \hat{X}_0) = D_\phi(X, X_0)$. Finally, the next lemma verifies that the constraints are equivalent as well.

Lemma 8 *Given a column orthogonal $n \times r$ matrix W satisfying* $\text{range}(X) \subseteq \text{range}(W)$*, it follows that* $\text{tr}(\hat{X}\hat{A}_i) = \text{tr}(XA_i)$*, where $\hat{X} = W^T X W$, $\hat{A}_i = W^T A_i W$.*

Proof Choose a reduced rank-r eigendecomposition of X to be $V\Lambda V^T$ such that the columns of V form an orthogonal basis of $\text{range}(W)$. Note that Λ will be singular when $\text{rank}(X)$ is less than r. There exists an $r \times r$ orthogonal Q that satisfies $W = VQ$, and so:

$$\begin{aligned} \text{tr}(\hat{X}\hat{A}_i) &= \text{tr}((W^T X W)(W^T A_i W)) = \text{tr}(Q^T V^T V \Lambda V^T V Q Q^T V^T A_i V Q) \\ &= \text{tr}(VQQ^T \Lambda Q Q^T V^T A_i) = \text{tr}(V\Lambda V^T A_i) = \text{tr}(XA_i). \end{aligned}$$

■

If we assume that the optimization problem (4) has a (rank-deficient) solution with finite divergence measure, then the corresponding full-rank optimization problem (14) also has a solution. Conversely, by Lemma 3, for a solution $\hat{X}$ of (14), there is a unique solution of (4), namely $X = W\hat{X}W^T$ (with finite $D_\phi(X, X_0)$) that satisfies the range space restriction.

4.3 Bregman Projections for Rank-deficient Matrices

Lastly, we derive the explicit updates for Bregman's algorithm in the low-rank setting. From now on we omit the constraint index for simplicity. Recall (7), which we used to calculate the projection update for Bregman's algorithm and apply it to the mapped problem (14):

$$\begin{aligned} \nabla\phi(\hat{X}_{t+1}) &= \nabla\phi(\hat{X}_t) + \alpha\hat{A} \\ \operatorname{tr}(\hat{X}_{t+1}\hat{A}) &= b. \end{aligned}$$

In case of the von Neumann divergence this leads to the update $\hat{X}_{t+1} = \exp(\log(\hat{X}_t) + \alpha\hat{A})$. The discussion in Section 4.2 and induction on t implies that $X_{t+1} = W\hat{X}_{t+1}W^T$ (with W as in Definition 4), or explicitly:

$$X_{t+1} = W\exp(\log(W^T X_t W) + \alpha(W^T A W))W^T.$$

If we choose $W = V_t$ from the reduced eigendecomposition $X_t = V_t\Lambda_t V_t^T$, then the update is written as:

$$X_{t+1} = V_t\exp(\log(\Lambda_t) + \alpha V_t^T A V_t)V_t^T, \tag{15}$$

which we will use in the von Neumann kernel learning algorithm. Note that the limit of $\exp(\log(X_t + \varepsilon I) + \alpha A)$ as ε approaches zero yields the same formula, which becomes clear if we apply a basis transformation that puts X_t in diagonal form.

The same argument applies to the Bregman projection for the LogDet divergence. In this case we arrive at the update:

$$X_{t+1} = V_t((V_t^T X_t V_t)^{-1} - \alpha(V_t^T A V_t))^{-1}V_t^T, \tag{16}$$

using Lemma 3 and the fact that $\operatorname{range}(X_{t+1}) = \operatorname{range}(X_t)$. The right hand side of (16) can be calculated without any eigendecomposition as we will show in Section 5.1.1.

5. Algorithms

In this section, we derive efficient algorithms for solving the optimization problem (5) for low-rank matrices using the LogDet and von Neumann divergences.

5.1 LogDet Divergence

We first develop a cyclic projection algorithm to solve (5) when the matrix divergence is the LogDet divergence.

5.1.1 MATRIX UPDATES

Consider minimizing $D_{\ell d}(X, X_0)$, the LogDet divergence between X and X_0. As given in (16), the projection update rule for low-rank X_0 and a rank-one constraint matrix $A = \boldsymbol{z}\boldsymbol{z}^T$ is:

$$X_{t+1} = V_t((V_t^T X_t V_t)^{-1} - \alpha(V_t^T \boldsymbol{z}\boldsymbol{z}^T V_t))^{-1}V_t^T,$$

where the eigendecomposition of X_t is $V_t\Lambda_t V_t^T$. Recall the Sherman-Morrison inverse formula (Sherman and Morrison, 1949; Golub and Van Loan, 1996):

$$(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^T A^{-1}}{1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u}}.$$

Applying this formula to the middle term of the projection update, we arrive at a simplified expression for X_{t+1}:

$$\begin{aligned} X_{t+1} &= V_t\left(V_t^T X_t V_t + \frac{\alpha V_t^T X_t V_t V_t^T z z^T V_t V_t^T X_t V_t}{1-\alpha z^T V_t V_t^T X_t V_t V_t^T z}\right)V_t^T \\ &= V_t\left(\Lambda_t + \frac{\alpha \Lambda_t V_t^T z z^T V_t \Lambda_t}{1-\alpha z^T V_t \Lambda_t V_t^T z}\right)V_t^T \\ &= V_t\Lambda_t V_t^T + \frac{\alpha V_t \Lambda_t V_t^T z z^T V_t \Lambda_t V_t^T}{1-\alpha z^T X_t z} \\ &= X_t + \frac{\alpha X_t z z^T X_t}{1-\alpha z^T X_t z}. \end{aligned} \tag{17}$$

Since X_{t+1} must satisfy the constraint, i. e., $\mathrm{tr}(X_{t+1} z z^T) = z^T X_{t+1} z = b$, we can solve the following equation for α:

$$z^T\left(X_t + \frac{\alpha X_t z z^T X_t}{1-\alpha z^T X_t z}\right) z = b. \tag{18}$$

Let $p = z^T X_t z$. Note that in the case of distance constraints, p is the distance between the two data points. When $p \neq 0$, elementary arguments reveal that there is exactly one solution for α provided that $b \neq 0$. The unique solution, in this case, can be expressed as:

$$\alpha = \frac{1}{p} - \frac{1}{b}. \tag{19}$$

If we let

$$\beta = \alpha/(1-\alpha p), \tag{20}$$

then our matrix update is given by

$$X_{t+1} = X_t + \beta X_t z z^T X_t. \tag{21}$$

This update is pleasant and surprising; usually the projection parameter for Bregman's algorithm does not admit a closed form solution (see Section 5.2.2 for the case of the von Neumann divergence). In the case where $p = 0$, (18) rewrites to $b = p/(1-\alpha p)$, which has a solution if and only if $b = 0$ (while α is arbitrary). It follows that z is in the null space of X_t, and by (17), $X_{t+1} = X_t$.

The following lemma confirms the expectation that we remain in the positive semidefinite cone and that the range space is unchanged.

Lemma 9 *Given a positive semidefinite matrix X_t, the matrix X_{t+1} from the update in (21) is positive semidefinite with* $\mathrm{range}(X_{t+1}) = \mathrm{range}(X_t)$*, assuming that (6) is feasible.*

Proof Factor the positive semidefinite matrix X_t as GG^T, where G is an $n \times r$ matrix and $r = \mathrm{rank}(X_t)$. The LogDet update produces:

$$X_{t+1} = GG^T + \beta GG^T z z^T GG^T = G(I + \beta G^T z z^T G)G^T.$$

We deduce immediately that $\mathrm{range}(X_{t+1}) \subseteq \mathrm{range}(G) = \mathrm{range}(X_t)$.

In order to prove that X_{t+1} is positive semidefinite and that the range space does not shrink, it suffices to show that all eigenvalues of $\beta G^T zz^T G$ are strictly larger than -1, implying that $I + \beta G^T zz^T G$ remains positive definite. The only non-zero eigenvalue of the rank-one $\beta G^T zz^T G$ equals $\mathrm{tr}(\beta G^T zz^T G) = \beta z^T GG^T z = \beta p$. According to (19) and (20) we calculate $\beta p = \frac{b}{p} - 1$, and noting that $b, p > 0$ completes the proof. ■

The update for X_{t+1} can alternatively be written using the pseudoinverse (Golub and Van Loan, 1996):

$$X_{t+1} = (X_t^\dagger - \alpha \tilde{A}_i)^\dagger,$$

where $\tilde{A}_i = WW^T A_i WW^T$ for an orthogonal matrix W satisfying $\mathrm{range}(W) = \mathrm{range}(X_t)$.[3]

Lemma 10 *Let $\tilde{A}_i = WW^T A_i WW^T$ and W be an orthogonal $n \times r$ matrix such that* $\mathrm{range}(W) = \mathrm{range}(X_t)$. *Then the following updates are equivalent:*

$$\begin{aligned} a)\ X_{t+1} &= W((W^T X_t W)^{-1} - \alpha(W^T A_i W))^{-1} W^T \\ b)\ X_{t+1} &= (X_t^\dagger - \alpha \tilde{A}_i)^\dagger \end{aligned}$$

Proof Since $\mathrm{range}(W) = \mathrm{range}(X_t)$, by Lemma 3 we have $\hat{X}_t = W^T X_t W$ and $X_t = W\hat{X}_t W^T$, where $\hat{X}_t$ is an invertible $r \times r$ matrix. Note that $W\hat{X}_t^{-1}W^T$ satisfies the properties required for the Moore-Penrose pseudoinverse, for example:

$$X_t(W\hat{X}_t^{-1}W^T)X_t = W\hat{X}_t W^T W\hat{X}_t^{-1}W^T W\hat{X}_t W^T = W\hat{X}_t W^T = X_t.$$

Thus $X_t^\dagger = W\hat{X}_t^{-1}W^T$ and we finish the proof by rewriting:

$$\begin{aligned} (X_t^\dagger - \alpha\tilde{A}_i)^\dagger &= (W\hat{X}_t^{-1}W^T - \alpha WW^T A_i WW^T)^\dagger = \left(W(\hat{X}_t^{-1} - \alpha W^T A_i W)W^T\right)^\dagger \\ &= W((W^T X_t W)^{-1} - \alpha(W^T A_i W))^{-1} W^T. \end{aligned}$$

■

5.1.2 UPDATE FOR THE FACTORED MATRIX

A naive implementation of the update given in (21) costs $O(n^2)$ operations per iteration. However, we can achieve a more efficient update for low-rank matrices by working on a suitable factored form of the matrix X_t resulting in an $O(r^2)$ algorithm. Both the reduced eigendecomposition and the Cholesky factorization are possible candidates for the factorization; we prefer the latter because the resulting algorithm does not have to rely on iterative methods.

The positive semidefinite rank-r matrix X_t can be factored as GG^T, where G is an $n \times r$ matrix, and thus the update can be written as:

$$X_{t+1} = G(I + \beta G^T zz^T G)G^T.$$

The matrix $I + \beta\tilde{z}_i\tilde{z}_i^T$, where $\tilde{z}_i = G^T z$, is an $r \times r$ matrix. To update G for the next iteration, we factor this matrix as LL^T; then our new G is updated to GL. Since $I + \beta\tilde{z}_i\tilde{z}_i^T$ is a rank-one

3. In Kulis et al. (2006), we had inadvertently used A_i instead of $\tilde{A}_i$.

Algorithm 1 CHOLUPDATEMULT(α, x, B). Right multiplication of a lower triangular $r \times r$ matrix B with the Cholesky factor of $I + \alpha \boldsymbol{x}\boldsymbol{x}^T$ in $O(r^2)$ time.

Input: $\alpha, \boldsymbol{x}, B$, with $I + \alpha \boldsymbol{x}\boldsymbol{x}^T \succeq 0$, B is lower triangular
Output: $B \leftarrow BL$, with $LL^T = I + \alpha \boldsymbol{x}\boldsymbol{x}^T$

$\alpha_1 = \alpha$
for $i = 1$ to r **do**
  $t = 1 + \alpha_i x_i^2$
  $h_i = \sqrt{t}$
  $\alpha_{i+1} = \alpha_i / t$
  $t = B_{ii}$
  $s = 0$
  $B_{ii} = B_{ii} h_i$
  for $j = i - 1$ **downto** 1 **do**
    $s = s + t x_{j+1}$
    $t = B_{ij}$
    $B_{ij} = (B_{ij} + \alpha_{j+1} x_j s) h_j$
  end for
end for

perturbation of the identity, this update can be done in $O(r^2)$ time using a standard Cholesky rank-one update routine.

To increase computational efficiency, we note that $G = G_0 B$, where B is the product of all the L matrices from every iteration and $X_0 = G_0 G_0^T$. Instead of updating G explicitly at each iteration, we simply update B to BL. The matrix $I + \beta G^T \boldsymbol{z}\boldsymbol{z}^T G$ is then $I + \beta B^T G_0^T \boldsymbol{z}\boldsymbol{z}^T G_0 B$. In the case of distance constraints, we can compute $G_0^T \boldsymbol{z}$ in $O(r)$ time as the difference of two rows of G_0. The multiplication update BL appears to have $O(r^3)$ complexity, dominating the run time. In the next section we derive an algorithm that combines the Cholesky rank-one update with the matrix multiplication into a single $O(r^2)$ routine.

5.1.3 FAST MULTIPLICATION WITH A CHOLESKY UPDATE FACTOR

We efficiently combine the Cholesky rank-one update with the matrix multiplication in CHOLUPDATEMULT given by Algorithm 1. A simple analysis of this algorithm reveals that it requires $3r^2 + 2r$ floating point operations (flops). This is opposed to the usual $O(r^3)$ time needed by matrix multiplication. We devote this section to the development of this fast multiplication algorithm.

Recall the algorithm used for the Cholesky factorization of an $r \times r$ matrix A (see Demmel, 1997, page 78):

for $j = 1$ **to** r **do**
  $l_{jj} = (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{1/2}$
  for $i = j + 1$ **to** r **do**
    $l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj}$
  end for
end for

We will derive a corresponding algorithm in a manner similar to Demmel (1997), while exploiting the special structure present in our problem. Let us denote $I_r + \alpha_1 \boldsymbol{x}\boldsymbol{x}^T$ by A ($\alpha_1 = \alpha$) and write it as a product of three block matrices:

$$A = \begin{bmatrix} \sqrt{1+\alpha_1 x_1^2} & 0 \\ \frac{\alpha_1 x_1}{\sqrt{1+\alpha_1 x_1^2}} \boldsymbol{x}_{2:r} & I_{r-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{A}_{22} \end{bmatrix} \begin{bmatrix} \sqrt{1+\alpha_1 x_1^2} & \frac{\alpha_1 x_1}{\sqrt{1+\alpha_1 x_1^2}} \boldsymbol{x}_{2:r}^T \\ 0 & I_{r-1} \end{bmatrix}.$$

It follows that $\tilde{A}_{22} + \frac{\alpha_1^2 x_1^2}{1+\alpha_1 x_1^2} \boldsymbol{x}_{2:r}\boldsymbol{x}_{2:r}^T = A_{22} = I_{r-1} + \alpha_1 \boldsymbol{x}_{2:r}\boldsymbol{x}_{2:r}^T$, leading to:

$$\tilde{A}_{22} = I_{r-1} + \frac{\alpha_1}{1+\alpha_1 x_1^2} \boldsymbol{x}_{2:r}\boldsymbol{x}_{2:r}^T.$$

Introduce $\alpha_2 = \frac{\alpha_1}{1+\alpha_1 x_1^2}$ and proceed by induction. We extract the following algorithm which calculates L satisfying $LL^T = A$:

for $j = 1$ **to** r **do**
  $t = 1 + \alpha_j x_j^2$
  $l_{jj} = \sqrt{t}$
  $\alpha_{j+1} = \alpha_j / t$
  $t = \alpha_j x_j / l_{jj}$
  for $i = j+1$ **to** r **do**
    $l_{ij} = t x_i$
  end for
end for

The above algorithm uses $\frac{1}{2}r^2 + \frac{13}{2}r$ flops to calculate L, while $\frac{1}{2}r^3 + O(r^2)$ are needed for the general algorithm. However, we do not necessarily have to calculate L explicitly, since the parameters α_i together with $\boldsymbol{x}$ implicitly determine L. Notice that the cost of calculating $\alpha_1, \alpha_2, \ldots, \alpha_r$ is linear in r.

Next we show how to calculate $\boldsymbol{u}^T L$ for a given vector $\boldsymbol{u}$ without explicitly calculating L and arrive at an $O(r)$ algorithm for this vector-matrix multiplication. The elements of $\boldsymbol{v}^T = \boldsymbol{u}^T L$ are equal to:

$$\begin{aligned}
v_1 &= u_1\sqrt{1+\alpha_1 x_1^2} + \frac{\alpha_2}{\sqrt{1+\alpha_1 x_1^2}} x_1(u_2 x_2 + u_3 x_3 + \ldots u_r x_r) \\
v_2 &= u_2\sqrt{1+\alpha_2 x_2^2} + \frac{\alpha_3}{\sqrt{1+\alpha_2 x_2^2}} x_2(u_3 x_3 + \ldots u_r x_r) \\
&\vdots \\
v_{r-1} &= u_{r-1}\sqrt{1+\alpha_{r-1} x_{r-1}^2} + \frac{\alpha_r}{\sqrt{1+\alpha_{r-1} x_{r-1}^2}} x_{r-1} u_r x_r \\
v_r &= u_r\sqrt{1+\alpha_r x_r^2}.
\end{aligned}$$

We can avoid the recalculation of some intermediate results if we evaluate v_r first, followed by v_{r-1} up to v_1. This strategy leads to the following algorithm:

Algorithm 2 Learning a low-rank kernel matrix in LogDet divergence under distance constraints.

Input: G_0: input kernel factor matrix, that is, $X_0 = G_0 G_0^T$, r: desired rank, $\{A_i\}_{i=1}^c$: distance constraints, where $A_i = (e_{i_1} - e_{i_2})(e_{i_1} - e_{i_2})^T$

Output: G: output low-rank kernel factor matrix, that is, $X = GG^T$

1: Set $B = I_r$, $i = 1$, and $\nu_j = 0 \ \forall$ constraints j.
2: **repeat**
3: $\quad \boldsymbol{v}^T = G_0(i_1,:) - G_0(i_2,:)$
4: $\quad \boldsymbol{w} = B^T \boldsymbol{v}$
5: $\quad \alpha = \min\left(\nu_i, \frac{1}{\|\boldsymbol{w}\|_2^2} - \frac{1}{b_i}\right)$
6: $\quad \nu_i \leftarrow \nu_i - \alpha$
7: $\quad \beta = \alpha/(1 - \alpha\|\boldsymbol{w}\|_2^2)$
8: $\quad$ Call CHOLUPDATEMULT$(\beta, \boldsymbol{w}, B)$ to factor $I + \beta \boldsymbol{w}\boldsymbol{w}^T = LL^T$ and update $B \leftarrow BL$.
9: $\quad$ Set $i \leftarrow \mathrm{mod}(i+1, c)$.
10: **until** convergence of ν
11: **return** $G = G_0 B$

$v_r = u_r\sqrt{1 + \alpha_r x_r^2}$
s = 0;
for $j = r - 1$ **downto** 1 **do**
$\quad s = s + u_{j+1}x_{j+1}$
$\quad t = \sqrt{1 + \alpha_j x_j^2}$
$\quad v_j = u_j t + \alpha_{j+1} x_j s/t$
end for

Exactly $11r - 6$ flops are required by the above algorithm, and therefore we can readily multiply an $r \times r$ matrix by L using $11r^2 - 6r$ flops. Even fewer flops are sufficient to implement the matrix multiplication if we observe that the square root expression above is repeatedly calculated for each row, since it depends only on x_j and α_j. Additionally, when multiplying with a lower triangular matrix, the presence of zeros allows further simplifications. Taking these considerations into account we arrive at the previously presented Algorithm 1, which uses exactly $3r^2 + 2r$ flops.

5.1.4 KERNEL LEARNING ALGORITHM

We are now ready to present the overall algorithm for distance inequality constraints using the LogDet divergence, see Algorithm 2. As discussed in the previous section, every projection can be done in $O(r^2)$ time. Thus, cycling through all c constraints requires $O(cr^2)$ time, but the total number of Bregman iterations may be large. The only dependence on the number of data points n occurs in steps 3 and 11. The last step, multiplying $G = G_0 B$, takes $O(nr^2)$ time.

As discussed earlier, convergence is guaranteed since we have mapped the original low-rank problem into a full-rank problem in a lower-dimensional space. Convergence can be checked by using the dual variables ν. The cyclic projection algorithm can be viewed as a dual coordinate ascent algorithm, thus convergence can be measured as follows: after cycling through all constraints, we check to see how much ν has changed after a full cycle. At convergence, this change (as measured with a vector norm) should be small.

5.2 Von Neumann Divergence

In this section we develop a cyclic projection algorithm to solve (5) when the matrix divergence is the von Neumann divergence.

5.2.1 MATRIX UPDATES

Consider minimizing $D_{vN}(X, X_0)$, the von Neumann divergence between X and X_0. Recall the projection update (15) for constraint i:

$$X_{t+1} = V_t \exp(\log(\Lambda_t) + \alpha V_t^T z z^T V_t) V_t^T.$$

If the eigendecomposition of the exponent $\log(\Lambda_t) + \alpha V_t^T z z^T V_t$ is $U_t \Theta_{t+1} U_t^T$, then the eigendecomposition of X_{t+1} is given by $V_{t+1} = V_t U_t$ and $\Lambda_{t+1} = \exp(\Theta_{t+1})$. This special eigenvalue problem (diagonal plus rank one update) can be solved in $O(r^2)$ time; see Golub (1973), Gu and Eisenstat (1994) and Demmel (1997). This means that the matrix multiplication $V_{t+1} = V_t U_t$ becomes the most expensive step in the computation, yielding $O(nr^2)$ complexity.

We reduce this cost by modifying the decomposition slightly. Let $X_t = V_t W_t \Lambda_t W_t^T V_t^T$ be the factorization of X_t, where W_t is an $r \times r$ orthogonal matrix, initially $W_0 = I_r$, while V_t and Λ_t are defined as before. The matrix update becomes

$$X_{t+1} = V_t W_t \exp(\log \Lambda_t + \alpha W_t^T V_t^T z z^T V_t W_t) W_t^T V_t^T,$$

yielding the following formulae:

$$V_{t+1} = V_t,\ W_{t+1} = W_t U_t,\ \Lambda_{t+1} = \exp(\Theta_{t+1}),$$

where $\log \Lambda_t + \alpha W_t^T V_t^T z z^T V_t W_t = U_t \Theta_{t+1} U_t^T$. For a general rank-one constraint the product $V_t^T z$ is calculated in $O(nr)$ time, but for distance constraints $O(r)$ operations are sufficient. The calculation of $W_t^T V_t^T z$ and computing the eigendecomposition $U_t \Theta_{t+1} U_t^T$ will both take $O(r^2)$ time. The matrix product $W_t U_t$ appears to cost $O(r^3)$ time, but in fact a right multiplication by U_t^T can be approximated very accurately in $O(r^2 \log r)$ and even in $O(r^2)$ time using the fast multipole method—see Barnes and Hut (1986) and Greengard and Rokhlin (1987).

Since we repeat the above update calculations until convergence, we can avoid calculating the logarithm of Λ_t at every step by maintaining $\Theta_t = \log \Lambda_t$ throughout the algorithm.

5.2.2 COMPUTING THE PROJECTION PARAMETER

In the previous section, we assumed that the projection parameter α has already been calculated. In contrast to the LogDet divergence, this parameter does not have a closed form solution. Instead, we must compute α by solving the nonlinear system of equations given by (7).

In the von Neumann case and in the presence of distance constraints, the problem amounts to finding the unique root of the function

$$f(\alpha) = z^T V_t W_t \exp(\Theta_t + \alpha W_t^T V_t^T z z^T V_t W_t) W_t^T V_t^T z - b.$$

It can be shown that $f(\alpha)$ is monotone in α, see Sustik and Dhillon (2008).

Using the approach from the previous section, $f(\alpha)$ can be computed in $O(r^2)$ time. One natural choice to find the root of $f(\alpha)$ is to apply Brent's general root finding method (Brent, 1973),

Algorithm 3 Learning a low-rank kernel matrix in von Neumann matrix divergence under distance constraints.

Input: V_0, Λ_0: input kernel factors, i. e., $X_0 = V_0 \Lambda_0 V_0^T$, r: rank of desired kernel matrix, $\{A_i\}_{i=1}^c$: distance constraints, where $A_i = (e_{i_1} - e_{i_2})(e_{i_1} - e_{i_2})^T$

Output: V, Λ: output low-rank kernel factors, i. e., $X = V \Lambda V^T$

Set $W = I_r$, $\Theta = \log \Lambda_0$, $i = 1$, and $\nu_j = 0 \ \forall$ constraints j.
repeat
  $\boldsymbol{v}^T = V_0(i_1,:) - V_0(i_2,:)$
  $\boldsymbol{w} = W^T \boldsymbol{v}$
  Call ZEROFINDER$(\Theta, \boldsymbol{w}, b_i)$ to determine α.
  $\beta = \min(\nu_i, \alpha)$
  $\nu_i \leftarrow \nu_i - \beta$.
  Compute eigendecomposition of $\Theta + \beta \boldsymbol{w}\boldsymbol{w}^T = U \tilde{\Theta} U^T$, $\Theta \leftarrow \tilde{\Theta}$.
  Update $W \leftarrow WU$.
  $i =$ mod $(i+1, c)$
until convergence of ν
return $V = V_0 W$, $\Lambda = \exp(\Theta)$

which does not need the calculation of the derivative of $f(\alpha)$. We have built an even more efficient custom root finder that is optimized for this problem. We rarely need more than six evaluations per projection to accurately compute α. A more detailed description of our root finder is beyond the scope of this paper, see Sustik and Dhillon (2008) for details.

5.2.3 KERNEL LEARNING ALGORITHM

The algorithm for distance inequality constraints using the von Neumann divergence is given as Algorithm 3. By using the fast multipole method every projection can be done in $O(r^2)$ time. Note that the asymptotic running time of this algorithm is the same as the LogDet divergence algorithm, although the root finding step makes Algorithm 3 slightly slower than Algorithm 2.

5.3 Limitations of our Approach

We conclude this section by briefly mentioning some limitations of our approach. First, though we are able to learn low-rank kernel matrices in our framework, the initial kernel matrix must be low-rank. As a result, we cannot use our methods to reduce the dimensionality of our data. Secondly, the method of Bregman projections may require iterating many cycles through all constraints to reach convergence. Although we have heavily optimized each iteration in this paper, it may be beneficial to search for an algorithm with faster convergence.

6. Discussion

We now further develop several aspects of the algorithms presented in the previous section. In particular, we discuss ways to move beyond the transductive setting in our framework for learning kernels, we briefly overview some generalizations of the problem formulation, we highlight how

special cases of our method are related to existing techniques, and we briefly analyze connections between our approach and semidefinite programming.

6.1 Handling New Data Points

The kernel learning algorithms of Section 5 are restricted to learning in the transductive setting; that is, we assume that we have all the data points up front, but labels or other forms of supervision are only available for some of the data points. This approach suffers when new, unseen data points are added, since this would require re-learning the entire kernel matrix.

Though we do not consider this situation in depth in the current paper, we can circumvent it. When learning a kernel matrix minimizing either the LogDet divergence or the von Neumann divergence, the range space restriction implies that the learned kernel K has the form $K = G_0 BB^T G_0^T$, where B is $r \times r$ and the input kernel is $K_0 = G_0 G_0^T$. We can view the matrix B as a linear transformation applied to our input data vectors G_0 and therefore we can apply this linear transformation to new points. In particular, recent work (Davis et al., 2007) has shown that the learning algorithms considered in this paper can equivalently be viewed as learning a *Mahalanobis distance function* given constraints on the data, which is simply the Euclidean distance after applying a linear transformation over the input data.

Since our algorithms can be viewed as Mahalanobis distance learning techniques, it is natural to compare against other existing metric learning algorithms. Such methods include metric learning by collapsing classes (MCML) (Globerson and Roweis, 2005), large-margin nearest neighbor metric learning (LMNN) (Weinberger et al., 2005), and many others. In the experimental results section, we provide some results comparing our algorithms with these existing methods.

6.2 Generalizations

There are several simple ways to extend the algorithms developed in Section 5. In this section, we introduce slack variables and discuss more general constraints than the distance constraints discussed earlier.

6.2.1 SLACK VARIABLES

In many cases, especially if the number of constraints is large, no feasible solution will exist to the Bregman divergence optimization problem given in (4). When no feasible solution exists, a common approach is to incorporate *slack variables*, which allows constraints to be violated but penalizes such violations.

There are many ways to introduce slack variables into the optimization problem (4). We add a new vector variable $\boldsymbol{b}$ with coordinates representing perturbed right hand sides of the linear constraints, and use the corresponding vector divergence to measure the deviation from the original constraints described by the input vector $\boldsymbol{b}_0$. The resulting optimization problem is as follows:

$$\begin{aligned} \text{minimize}_{X,\boldsymbol{b}} \quad & D_\phi(X, X_0) + \gamma D_\varphi(\boldsymbol{b}, \boldsymbol{b}_0) \\ \text{subject to} \quad & \operatorname{tr}(XA_i) \leq \boldsymbol{e}_i^T \boldsymbol{b},\ 1 \leq i \leq c \\ & X \succeq 0. \end{aligned} \tag{22}$$

Note that we use $D_\varphi(\boldsymbol{b}, \boldsymbol{b}_0)$ as the penalty for constraints as it is computationally simple; other choices of constraint penalty are possible, as long as the resulting objective function is convex.

The $\gamma > 0$ parameter governs the tradeoff between satisfying the constraints and minimizing the divergence between X and X_0. Note that we have also removed the implicit rank constraint for simplicity.

We form the Lagrangian to solve the above problem; recall the similar system of equations (7) in Section 3.4. Define $\mathcal{L}(X_{t+1}, \boldsymbol{b}_{t+1}, \alpha_i) = D_\phi(X_{t+1}, X_t) + \gamma D_\varphi(\boldsymbol{b}_{t+1}, \boldsymbol{b}_t) + \alpha_i(\boldsymbol{e}_i^T \boldsymbol{b}_{t+1} - \mathrm{tr}(X_{t+1}A_i))$, and set the gradients to zero with respect to X_{t+1}, $\boldsymbol{b}_{t+1}$ and α_i to get the following update equations:

$$\begin{aligned} \nabla\phi(X_{t+1}) &= \nabla\phi(X_t) + \alpha_i A_i, \\ \nabla\varphi(\boldsymbol{b}_{t+1}) &= \nabla\varphi(\boldsymbol{b}_t) - \frac{\alpha_i}{\gamma}\boldsymbol{e}_i, \\ \mathrm{tr}(X_{t+1}A_i) &= \boldsymbol{e}_i^T \boldsymbol{b}_{t+1}. \end{aligned} \tag{23}$$

In particular, for the LogDet divergence we arrive at the following updates:

$$X_{t+1}^{-1} = X_t^{-1} - \alpha_i A_i, \quad \boldsymbol{e}_i^T \boldsymbol{b}_{t+1} = \frac{\gamma \boldsymbol{e}_i^T \boldsymbol{b}_t}{\gamma + \alpha_i \boldsymbol{e}_i^T \boldsymbol{b}_t}, \quad \mathrm{tr}(X_{t+1}A_i) - \boldsymbol{e}_i^T \boldsymbol{b}_{t+1} = 0.$$

Assuming $A_i = \boldsymbol{z}_i \boldsymbol{z}_i^T$, we can still compute α_i in closed form as

$$\alpha_i = \frac{\gamma}{\gamma+1}\left(\frac{1}{p} - \frac{1}{b_i}\right), \text{ where } p = \boldsymbol{z}_i^T X_t \boldsymbol{z}_i, b_i = \boldsymbol{e}_i^T \boldsymbol{b}_t.$$

The ith element of $\boldsymbol{b}_t$ is updated to $\gamma b_i/(\gamma + \alpha_i b_i)$ and the matrix update is calculated as in the non-slack case.

In case of the von Neumann divergence the update rules turn out to be:

$$\log X_{t+1} = \log X_t + \alpha_i A_i, \quad \boldsymbol{e}_i^T \boldsymbol{b}_{t+1} = \boldsymbol{e}_i^T \boldsymbol{b}_t e^{-\frac{\alpha_i}{\gamma}}, \quad \mathrm{tr}(X_{t+1}A_i) - \boldsymbol{e}_i^T \boldsymbol{b}_{t+1} = 0.$$

The projection parameter α_i is not available in closed form, instead we calculate it as the root of the non-linear equation:

$$\log \mathrm{tr}(\exp(\log X_t + \alpha_i A_i) A_i) + \frac{\alpha_i}{\gamma} - \log \boldsymbol{e}_i^T \boldsymbol{b}_t = 0.$$

The scale invariance of the LogDet divergence implies that $D_{\ell d}(\boldsymbol{b}, \boldsymbol{b_0}) = D_{\ell d}(\boldsymbol{b}/\boldsymbol{b_0}, \mathbf{1})$ where the division is elementwise and therefore we implicitly measure "relative" error, that is the magnitude of the coordinates of vector $\boldsymbol{b}_0$ are naturally taken into account. For the von Neumann divergence scale invariance does not hold, but one may alternatively use $D_{vN}(\boldsymbol{b}/\boldsymbol{b_0}, \mathbf{1})$ as the penalty function.

6.2.2 OTHER CONSTRAINTS

In earlier sections, we focused on the case of distance constraints. We now briefly discuss generalizing to other constraints.

When the constraints are similarity constraints (i.e., $K_{jl} \leq b_i$), the updates can be easily modified. As discussed earlier, we must retain symmetry of the A_i constraint matrices, so for similarity constraints, we require $A_i = \frac{1}{2}(\boldsymbol{e}_j \boldsymbol{e}_l^T + \boldsymbol{e}_l \boldsymbol{e}_j^T)$. Notice that the constraint matrix now has rank two. This complicates the algorithms slightly; for example, with the LogDet divergence, the Sherman-Morrison formula must be applied twice, leading to a more complicated update rule (albeit one that still has a closed-form solution and can be computed in $O(r^2)$ time).

Other constraints are possible as well; for example, one could incorporate distance constraints such as $\|\boldsymbol{a}_j - \boldsymbol{a}_k\|_2^2 \leq \|\boldsymbol{a}_l - \boldsymbol{a}_m\|_2^2$, or further similarity constraints such as $K_{jk} \leq K_{lm}$. These are sometimes referred to as relative comparison constraints, and are useful in ranking algorithms (Schultz and Joachims, 2003); for these also, the cost per projection remains $O(r^2)$. Finally, arbitrary linear constraints can also be applied, the cost per projection update will then be $O(nr)$.

6.3 Special Cases

If we use the von Neumann divergence, let $r = n$, and set $b_i = 0$ for all constraints, we exactly obtain the DefiniteBoost optimization problem from Tsuda et al. (2005). In this case, our algorithm computes the projection update in $O(n^2)$ time. In contrast, the algorithm from Tsuda et al. (2005) computes the projection in a more expensive manner in $O(n^3)$ time. Another difference with our approach is that we compute the exact projection, whereas Tsuda et al. (2005) compute an approximate projection. Though computing an approximate projection may lead to a faster per-iteration cost, it takes many more iterations to converge to the optimal solution. We illustrate this further in the experimental results section.

The online-PCA problem discussed in Warmuth and Kuzmin (2006) employs a similar update based on the von Neumann divergence. As with the DefiniteBoost algorithm, the projection is not an exact Bregman projection; however, the projection can be computed in the same way as in our von Neumann kernel learning projection. As a result, the cost of an iteration of online-PCA can be improved from $O(n^3)$ to $O(n^2)$ with our approach.

Another special case is the *nearest correlation matrix* problem (Higham, 2002) that arises in financial applications. A correlation matrix is a positive semidefinite matrix with unit diagonal. For this case, we set the constraints to be $K_{ii} = 1$ for all i. Our algorithms from this paper give new divergence measures and methods for finding low-rank correlation matrices. Previous algorithms scale cubically in n, whereas our method scales linearly with n and quadratically with r for low-rank correlation matrices.

Our formulation can also be employed to solve a problem similar to that of Weinberger et al. (2004). The enforced constraints on the kernel matrix (centering and isometry) are linear, and thus can be encoded into our framework. The only difference is that Weinberger et al. maximize the trace of K, whereas we minimize a matrix divergence. Comparisons between these approaches is a potential area of future research.

6.4 Connections to Semidefinite Programming

In this section, we present a connection between minimizing the LogDet divergence and the solution to a semidefinite programming problem (SDP). As an example, we consider the min-balanced-cut problem.

Suppose we are given an n-vertex graph, whose adjacency matrix is A. Let $L = \mathrm{diag}(Ae) - A$ be the Laplacian of A, where $\boldsymbol{e}$ is the vector of all ones. The semidefinite relaxation to the minimum balanced cut problem (Lang, 2005) results in the following SDP:

$$\begin{aligned} \min_X \quad & \operatorname{tr}(LX) \\ \text{subject to} \quad & \operatorname{diag}(X) = \boldsymbol{e} \\ & \operatorname{tr}(X\boldsymbol{e}\boldsymbol{e}^T) = 0 \\ & X \succeq 0. \end{aligned}$$

Let $L^\dagger$ denote the pseudoinverse of the Laplacian, and let V be an orthonormal basis for the range space of L. Let r denote the rank of L; it is well known that $n-r$ equals the number of connected components of the graph. Consider the LogDet divergence between X and $\varepsilon L^\dagger$:

$$\begin{aligned} D_{\ell d}(X, \varepsilon L^\dagger) &= D_{\ell d}(V^T X V, V^T(\varepsilon L^\dagger)V) \\ &= \operatorname{tr}(V^T X V (V^T(\varepsilon L^\dagger)V)^{-1}) - \log\det(V^T X V (V^T(\varepsilon L^\dagger)V)^{-1}) - r \\ &= \operatorname{tr}\left(V^T X V \left(\frac{1}{\varepsilon} V^T L V\right)\right) - \log\det\left(V^T X V \left(\frac{1}{\varepsilon} V^T L V\right)\right) - r \\ &= \frac{1}{\varepsilon}\operatorname{tr}(LX) - \log\det\left(\frac{1}{\varepsilon} V^T X V V^T L V\right) - r \\ &= \frac{1}{\varepsilon}\operatorname{tr}(LX) - \log\det(V^T X V V^T L V) - r - \log(\varepsilon^{-r}). \end{aligned}$$

The fourth line uses Lemma 8 to replace $\operatorname{tr}(V^T X V(\frac{1}{\varepsilon} V^T L V))$ with $\frac{1}{\varepsilon}\operatorname{tr}(LX)$. If we aim to minimize this divergence, we can drop the last two terms, as they are constants. In this case, we have:

$$\begin{aligned} \operatorname{argmin}_X D_{\ell d}(X, \varepsilon L^\dagger) &= \operatorname{argmin}_X \frac{1}{\varepsilon}\operatorname{tr}(LX) - \log\det(V^T X V V^T L V) \\ &= \operatorname{argmin}_X \operatorname{tr}(LX) - \varepsilon \log\det(V^T X V V^T L V). \end{aligned}$$

As ε becomes small, the $\operatorname{tr}(LX)$ term dominates the objective function.

Now consider the constraints from the min balanced cut SDP. The $\operatorname{diag}(X) = \boldsymbol{e}$ constraint is exactly the constraint from the nearest correlation matrix problem (that is, $X_{ii} = 1$ for all i). The $X \succeq 0$ constraint is implicitly satisfied when LogDet is used, leaving the balancing constraint $\boldsymbol{e}^T X \boldsymbol{e} = 0$. Recall that $\operatorname{null}(X) = \operatorname{null}(L^\dagger)$, and from standard spectral graph theory, that $L\boldsymbol{e} = 0$. Therefore $L^\dagger \boldsymbol{e} = 0$, which further implies

$$\boldsymbol{e}^T L^\dagger \boldsymbol{e} = 0 \Rightarrow \boldsymbol{e}^T X \boldsymbol{e} = 0$$

by the fact that the LogDet divergence preserves the range space of $L^\dagger$. Thus, the null space restriction in LogDet divergence naturally yields the constraint $\boldsymbol{e}^T X \boldsymbol{e} = 0$ and so the min balanced cut SDP problem on A is equivalent (for sufficiently small ε) to finding the nearest correlation matrix to $\varepsilon L^\dagger$ under the LogDet divergence.

Many other semidefinite programming problems can be solved in a similar manner to the one given above for the min balanced cut problem. It is beyond the scope of this paper to determine the practicality of such an optimization method; our aim here is simply to demonstrate an intriguing relationship between semidefinite programming and minimizing the LogDet divergence. For further information on this relationship, see Kulis et al. (2007a).

6.5 Changing the Range Space

The key property of the LogDet and von Neumann divergences which allow the algorithms to learn low-rank kernel matrices is their range space preservation property, discussed in Section 4. While this property leads to efficient algorithms for learning low-rank kernel matrices, it also forces the learned matrices to maintain the range space of the input matrix, which is a limitation of our method.

Allowing the range space to change is potentially a very useful tool, but is still an open research question. One possibility worth exploring is to augment the input matrix X_0 with a matrix capturing the complementary space, that is, the input would be $X_0 + \varepsilon N$, where N captures the null space of X_0. Even if there does not exist a solution to the optimization problem of minimizing $D_\phi(X, X_0)$, there is always a solution to minimizing $D_\phi(X, X_0 + \varepsilon N)$. Details of this approach are to be pursued as future work.

A related approach to circumvent this problem is to apply an appropriate kernel function over the input data, the result of which is that the algorithm learns a non-linear transformation of the data over the input space. Recently, Davis et al. (2007) discussed how one can generalize to new data points with the LogDet divergence even after applying such a kernel function, making this approach practical in many situations.

7. Experiments

In this section, we present a number of results using our algorithms, and provide references to recent work which has applied our algorithms to large-scale learning tasks. We first begin with the basic algorithm, and present results on transductive learning—the scenario where all data is provided upfront but labels are available for only a subset of the data—and clustering. We then discuss some results comparing our methods to existing metric learning algorithms.

We run the kernel learning algorithms in MATLAB, with some routines written in C and compiled with the MEX compiler. An efficient implementation of the special matrix multiplication appearing in step 9 of Algorithm 3 could achieve further run-time improvements. Unfortunately, an efficient and accurate implementation—based on the fast multipole method—is not readily available.

7.1 Transductive Learning and Clustering Results

To show the effectiveness of our algorithms, we present results from clustering as well as classification. We consider several data sets from real-life applications:

1. `Digits:` a subset of the *Pendigits* data from the UCI repository that contains handwritten samples of the digits 3, 8, and 9. The raw data for each digit is 16-dimensional; this subset contains 317 digits and is a standard data set for semi-supervised clustering (e.g., see, Bilenko et al., 2004).

2. `GyrB:` a protein data set: a 52×52 kernel matrix among bacteria proteins, containing three bacteria species. This matrix is identical to the one used to test the DefiniteBoost algorithm in Tsuda et al. (2005).

3. `Spambase:` a data set from the UCI repository containing 4601 email messages, classified as spam or not spam—1813 of the emails are spam (39.4 percent). This data has 57 attributes.

4. Nursery: data set developed to rank applications for nursery schools. This set contains 12960 instances with 8 attributes, and 5 class labels.

For classification, we compute accuracy using a k-nearest neighbor classifier ($k = 5$) that computes distance in the feature space, with a 50/50 training/test split and two-fold cross validation. Our results are averaged over 20 runs. For clustering, we use the kernel k-means algorithm and compute accuracy using the Normalized Mutual Information (NMI) measure, a standard technique for determining quality of clusters, which measures the amount of statistical information shared by the random variables representing the cluster and class distributions (Strehl et al., 2000). If C is the random variable denoting the cluster assignments of the points, and K is the random variable denoting the underlying class labels on the points then the NMI measure is defined as:

$$NMI = \frac{I(C;K)}{(H(C)+H(K))/2}$$

where $I(X;Y) = H(X) - H(X|Y)$ is the mutual information between the random variables X and Y, $H(X)$ is the Shannon entropy of X, and $H(X|Y)$ is the conditional entropy of X given Y (Cover and Thomas, 1991). The normalization by the average entropy of C and K makes the value of NMI stay between 0 and 1.

We consider two different experiments, each with their own constraint selection procedure. One experiment is to learn a kernel matrix *only using constraints*. In this case, we generate constraints from some target kernel matrix, and judge performance on the learned kernel matrix as we provide an increasing number of constraints. This setup was employed for the GyrB data set allowing us to compare to the methods and results of Tsuda et al. (2005). Constraints were generated randomly as follows: two data points are chosen at random and a distance constraint is constructed as follows. If the data points are in the same class, the constraint is of the form $d(i, j) \leq b$, where b is the corresponding distance between points i and j from the original kernel matrix, and if the data points are from different classes, then the constraint is of the form $d(i, j) \geq b$.

For the remaining data sets, we were interested in adding supervised class information to *improve* an existing low-rank kernel matrix. In this case, we took our initial low-rank kernel to be a linear kernel over the original data matrix and we added constraints of the form $d(i, j) \leq (1-\varepsilon)b$ for same class pairs and $d(i, j) \geq (1+\varepsilon)b$ for different class pairs (b is the original distance and $\varepsilon = .25$). This is very similar to "idealizing" the kernel, as in Kwok and Tsang (2003). Our convergence tolerance was set to 10^{-3}, and we incorporated slack variables for the larger data sets Spambase and Nursery (the smaller data sets did not require slack variables to converge). Note that there are other methods for choosing constraints; for example, instead of using $(1-\varepsilon)b$ for the right-hand side for same class pairs, one could instead choose a global value u for all same class constraints (and analogously for different class pairs).

We first show that the low-rank kernels learned by our algorithms attain good clustering and classification accuracies. We ran our algorithms on the Digits data set to learn a rank-16 kernel matrix using randomly generated constraints. The Gram matrix of the original Digits data was used as our initial (rank-16) kernel matrix. The left plot of Figure 1 shows clustering NMI values with increasing constraints. Adding just a few constraints improves the results significantly, and both of the kernel learning algorithms perform comparably. Classification accuracy using the k-nearest neighbor method was also computed for this data set: marginal classification accuracy gains were observed with the addition of constraints (an increase from 94 to 97 percent accuracy for

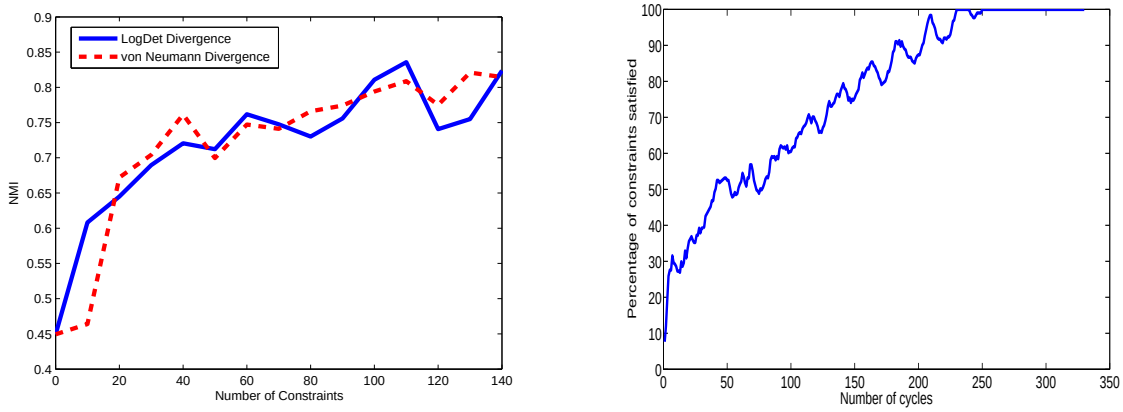

Figure 1: (Left) Normalized Mutual Information results of clustering the `Digits` data set—both algorithms improve with a small number of constraints (Right) Percentage of constraints satisfied (to a tolerance of 10^{-3}) in Bregman's algorithm as a function of the number of cycles, using the LogDet divergence

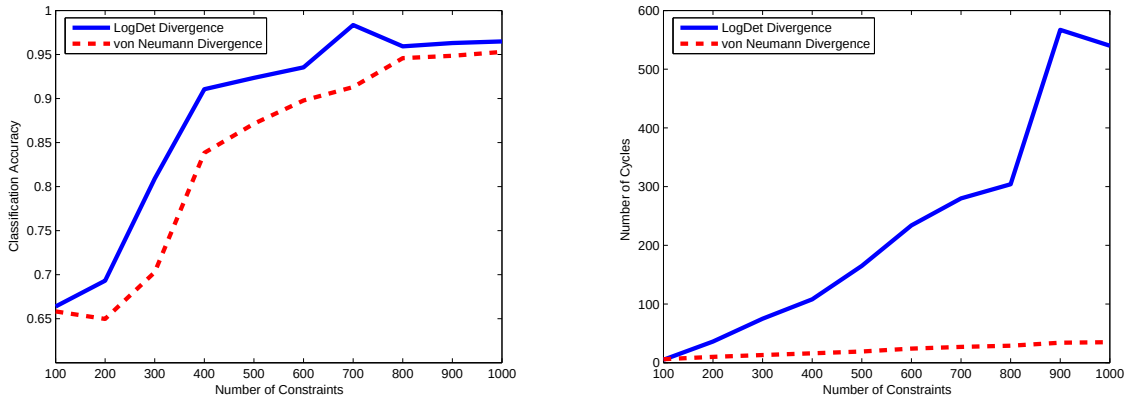

Figure 2: Classification accuracy (left) and convergence (right) on the `GyrB` data set

both divergences). We also recorded convergence data on `Digits` in terms of the number of cycles through all constraints; for the von Neumann divergence, convergence was attained in 11 cycles for 30 constraints and in 105 cycles for 420 constraints, while for the LogDet divergence, between 17 and 354 cycles were needed for convergence. This experiment highlights how our algorithm can use constraints to learn a low-rank kernel matrix. It is noteworthy that the learned kernel performs better than the original kernel for clustering as well as classification. Furthermore, in the right plot of Figure 1, we plot the number of constraints satisfied within a tolerance of 10^{-3} (300 total constraints) as a function of the number of cycles through all constraints, when using the LogDet divergence. Similar results are obtained with the von Neumann divergence. The number of cycles required for convergence to high accuracy can potentially be large, but for typical machine learning tasks, high accuracy solutions are not necessary and low to medium accuracy solutions can be achieved much faster.

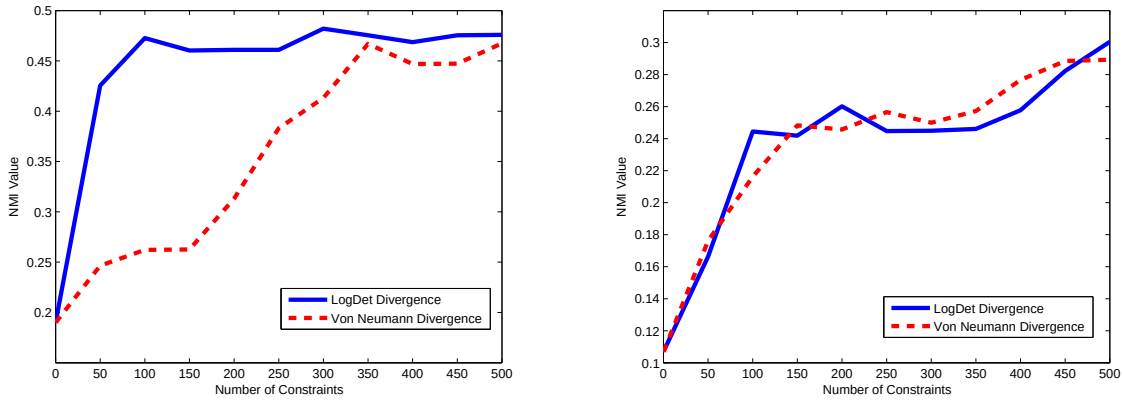

Figure 3: Clustering accuracy on `Spambase` (left) and `Nursery` (right)

As a second experiment, we performed experiments on `GyrB` and comparisons to the DefiniteBoost algorithm of Tsuda et al. (2005) (modified to correctly handle inequality constraints). Using only constraints, we attempt to learn a kernel matrix that achieves high classification accuracy. In order to compare to DefiniteBoost, we learned a full-rank kernel matrix starting from the scaled identity matrix as in Tsuda et al. (2005). In our experiments, we observed that using approximate projections—as done in DefiniteBoost—considerably increases the number of cycles needed for convergence. For example, starting with the scaled identity as the initial kernel matrix and 100 constraints, it took our von Neumann algorithm only 11 cycles to converge, whereas it took 3220 cycles for the DefiniteBoost algorithm to converge. Since the optimal solutions are the same for approximate versus exact projections, we converge to the same kernel matrix as DefiniteBoost but in far fewer iterations. Furthermore, our algorithm has the ability to learn low-rank kernels (as in the case of the `Digits` example). Figure 2 depicts the classification accuracy and convergence of our LogDet and von Neumann algorithms on this data set. The slow convergence of the DefiniteBoost algorithm did not allow us to run it with a larger set of constraints. For the LogDet and the von Neumann exact projection algorithms, the number of cycles required for convergence never exceeded 600 on runs of up to 1000 constraints on `GyrB`. The classification accuracy on the original matrix is .948, and so our learned kernels achieve even higher accuracy than the target kernel with a sufficient number of constraints. These results highlight that excellent classification accuracy can be obtained using a kernel that is learned using only distance constraints. Note that the starting kernel was the identity matrix, and so did not encode any domain information.

On the larger data sets `Spambase` and `Nursery`, we found similar improvements in clustering accuracy while learning rank-57 and rank-8 kernel matrices, respectively, using the same setup as in the `Digits` experiment. Figures 3 and 4 show the clustering accuracy and convergence on these data sets. Note that both data sets are too large for DefiniteBoost to handle, and furthermore, storing the full kernel matrix would require significant memory overhead (for example, MATLAB runs out of memory while attempting to store the full kernel matrix for `Nursery`); this scenario highlights the memory overhead advantage of using low-rank kernels. We see that on the `Spambase` data set, the LogDet divergence algorithm produces clustering accuracy improvements with very little supervision, but the number of cycles to converge is much larger than the von Neumann algorithm (for 500 constraints, the von Neumann algorithm requires only 29 cycles to converge). The slower

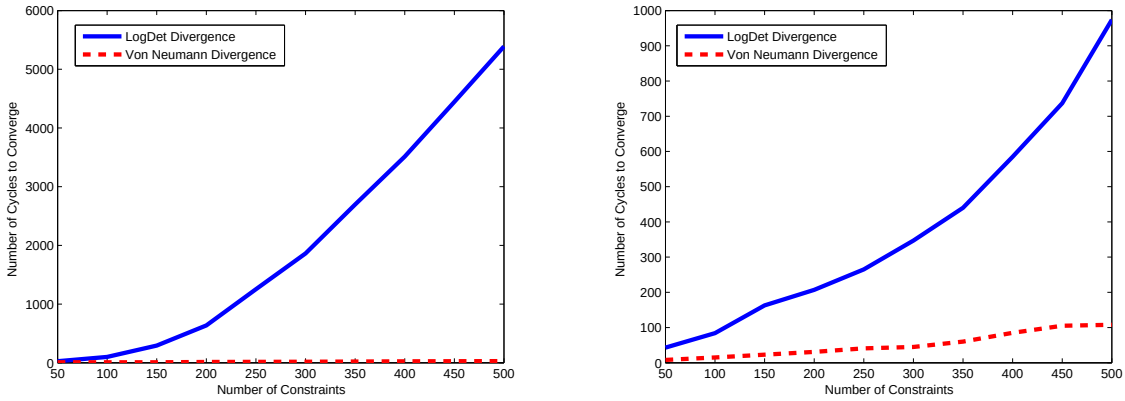

Figure 4: Convergence on `Spambase` (left) and `Nursery` (right)

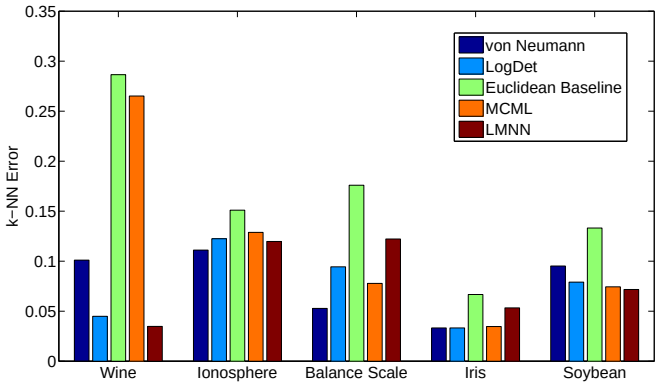

Figure 5: Classification error rates for k-nearest neighbor classification via different learned metrics over UCI data sets. The von Neumann and LogDet algorithms are competitive with existing state of the art metric learning methods, and significantly outperform the baseline.

convergence of the LogDet divergence algorithm is a topic of future work—note, however, that even though the number of cycles for LogDet to converge is higher than for von Neumann, the overall running time is often lower due to the efficiency of each LogDet iteration and the lack of a suitable implementation of the fast multiple method. On the `Nursery` data set, both algorithms perform similarly in terms of clustering accuracy, and the discrepancy in terms of cycles to converge between the algorithms is less drastic. We stress that other methods for learning kernel matrices, such as using the squared Frobenius norm in place of the LogDet or von Neumann divergences, would lead to learning full-rank matrices and would require significantly more memory and computational resources.

7.2 Metric Learning and Large-Scale Experiments

We now move beyond the transductive setting in order to compare with existing metric learning algorithms. As briefly discussed in Section 6.1, learning a low-rank kernel with the same range space as the input kernel is equivalent to learning a linear transformation of the input data, so we now compare against some existing methods for learning linear transformations. In particular, we compare against two popular Mahalanobis metric learning algorithms: metric learning by collapsing classes (MCML) (Globerson and Roweis, 2005) and large-margin nearest neighbor metric learning (LMNN) (Weinberger et al., 2005). As a baseline, we consider the squared Euclidean distance.

For each data set, we use two-fold cross validation to learn a linear transformation of our training data. This transformation is used in a k-nearest neighbor classifier on our test data, and the resulting error is reported. For the LogDet and von Neumann algorithms, we cross-validate the γ parameter and use the constraint selection procedure described in Davis et al. (2007). In particular, we generate constraints of the form $d(i, j) \leq u$ for same-class pairs and $d(i, j) \geq \ell$ for different-class pairs, where u and ℓ are chosen based on the 5th and 95th percentile of all distances in the training data. We randomly choose $40c^2$ constraints, where c is the number of classes in the data.

We ran on a number of standard UCI data sets to compare with existing methods. In Figure 5, we display k-NN accuracy ($k = 4$) over five data sets. Overall, we see that both the LogDet and von Neumann algorithms compare favorably with existing methods.

In terms of large-scale experiments, we also ran on the MNIST data set, a handwritten digits data set. This data has 60,000 training points and 10,000 test points. After deskewing the images and performing dimensionality reduction to the first 100 principal components of MNIST, we ran our methods successfully with 10,000 and 100,000 constraints, chosen as in the metric learning experiments above. The baseline error for a 1-nearest neighbor search over the top 100 principal components after deskewing is 2.35%. For 10,000 constraints, the LogDet algorithm ran in 4.8 minutes and achieved a test error of 2.29%, while the von Neumann algorithm ran in 7.7 minutes and achieved 2.30% error. For 100,000 constraints, LogDet ran in 11.3 minutes and achieved an error of 2.18% while von Neumann ran in 71.4 minutes and achieved an error of 2.17%.

Finally, we note that our methods have recently been applied to very large problems in the computer vision domain. We refer the reader to Jain et al. (2008), which adapts the algorithms discussed in this paper to three large-scale vision experiments: human body pose estimation, feature indexing for 3-d reconstruction, and object classification. For pose estimation, the size of the data was 500,000 images and for feature indexing, there were 300,000 image patches. These experiments validate the use of our methods on large-scale data, and also demonstrate that they can be used to outperform state of the art methods in computer vision.

8. Conclusions

In this paper, we have developed algorithms for using Bregman matrix divergences for low-rank matrix nearness problems. In particular, we have developed a framework for using the LogDet divergence and the von Neumann divergence when the initial matrices are low-rank; this is achieved via a restriction of the range space of the matrices. Unlike previous kernel learning algorithms, which have running times that are cubic in the number of data points, our resulting algorithms are efficient—both algorithms have running times linear in the number of data points and quadratic in the rank of the kernel. Furthermore, our algorithms can be used in conjunction with a number of kernel-based learning algorithms that are optimized for low-rank kernel representations. The

experimental results demonstrate that our algorithms are effective in learning low-rank and full-rank kernels for classification and clustering problems on large-scale data sets that arise in diverse applications.

There is still much to be gained from studying these divergences. Algorithmically, we have considered only projections onto a single constraint, but it is worth pursuing other approaches to solve the optimization problems such as methods that optimize with respect to multiple constraints simultaneously. We discussed how the LogDet divergence exhibits connections to Mahalanobis metric learning and semidefinite programming, and there is significant ongoing work in these areas. Finally, we hope that our framework will lead to new insights and algorithms for other machine learning problems where matrix nearness problems need to be solved.

Acknowledgments

This research was supported by NSF grant CCF-0431257, NSF Career Award ACI-0093404, and NSF-ITR award IIS-0325116. We thank Koji Tsuda for the protein data set and Kilian Weinberger for the image deskewing code.

Appendix A. Properties of Bregman Matrix Divergences

In this section, we highlight some important properties of Bregman matrix divergences.

Proposition 11 *Let Q be a square, orthogonal matrix, that is, $Q^TQ = QQ^T = I$. Then for all spectral Bregman matrix divergences, $D_\phi(Q^TXQ, Q^TYQ) = D_\phi(X,Y)$.*

Proof This follows from Lemma 1 by observing that if $X = V\Lambda V^T$ and $Y = U\Theta U^T$, the ith eigenvector of Q^TXQ is $Q^T\boldsymbol{v}_i$ and the jth eigenvector of Q^TYQ is $Q^T\boldsymbol{u}_j$. The eigenvalues remain unchanged by the orthogonal similarity transformation. Thus, the term $((Q^T\boldsymbol{v}_i)^T(Q^T\boldsymbol{u}_j))^2$ in Lemma 1 simplifies to $(\boldsymbol{v}_i^T\boldsymbol{u}_j)^2$ using the fact that $QQ^T = I$. ■

Proposition 12 *Let M be a square, non-singular matrix, and let X and Y be $n \times n$ positive definite matrices. Then $D_{\ell d}(M^TXM, M^TYM) = D_{\ell d}(X,Y)$.*

Proof First observe that if X is positive definite, then M^TXM is positive definite. Then:

$$\begin{aligned} D_{\ell d}(M^TXM, M^TYM) &= \operatorname{tr}(M^TXM(M^TYM)^{-1}) - \log\det(M^TXM(M^TYM)^{-1}) - n \\ &= \operatorname{tr}(M^TXMM^{-1}Y^{-1}M^{-T}) - \log\det(M^TXMM^{-1}Y^{-1}M^{-T}) - n \\ &= \operatorname{tr}(XY^{-1}) - \log\det(XY^{-1}) - n \\ &= D_{\ell d}(X,Y) \end{aligned}$$

■

Note that, as a corollary, we have that the LogDet divergence is *scale-invariant*, that is, $D_{\ell d}(X,Y) = D_{\ell d}(cX,cY)$, for any positive scalar c. Furthermore, this proposition may be extended to the case when X and Y are positive semidefinite, using the definition of the LogDet divergence for rank-deficient matrices.

Proposition 13 *Let Y be a positive definite matrix, and let X be any positive semidefinite matrix. Then $D_{\ell d}(X,Y) = D_{vN}(I, Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) = D_{\ell d}(Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}, I)$*

Proof The first equality by expanding the definition of the von Neumann divergence:

$$\begin{aligned} D_{vN}(I, Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) &= \operatorname{tr}(I\log I - I\log(Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) - I + Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) \\ &= \operatorname{tr}(Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) - \operatorname{tr}(\log(Y^{-\frac{1}{2}}XY^{-\frac{1}{2}})) - \operatorname{tr}(I) \\ &= \operatorname{tr}(XY^{-1}) - \log\det(Y^{-\frac{1}{2}}XY^{-\frac{1}{2}}) - n \\ &= \operatorname{tr}(XY^{-1}) - \log\det(XY^{-1}) - n \\ &= D_{\ell d}(X,Y), \end{aligned}$$

where we have used the fact that $\operatorname{tr}(\log A) = \log\det(A)$. The second equality follows by applying Proposition 12 to $D_{\ell d}(X,Y)$ with $M = Y^{-\frac{1}{2}}$. ■

Appendix B. Necessity of Dual Variable Corrections

In Section 3.4, we presented the method of Bregman projections and showed that, in the presence of inequality constraints, a dual variable correction is needed to guarantee convergence to the globally optimal solution. In some recent machine learning papers such as Tsuda et al. (2005), this correction has been omitted from the algorithm. We now briefly demonstrate why such corrections are needed.

A very simple example illustrating the failure of Bregman projections without corrections is in finding of the nearest 2×2 (positive definite) matrix X to the identity matrix that satisfies a single linear constraint:

$$\begin{aligned} \text{minimize}_X \quad & D_\phi(X, I) \\ \text{subject to} \quad & X_{11} + X_{22} - 2X_{12} \geq 1 \\ & X \succeq 0. \end{aligned}$$

The starting (identity) matrix satisfies the linear constraint, but a single projection step (to the corresponding equality constraint) without corrections will produce a suboptimal matrix, regardless of the divergence used. On the other hand, employing corrections leads to $\alpha_i' = 0$, and the input matrix remains unchanged.

It is also not sufficient to repeatedly perform Bregman projections only on violated constraints (without any corrections) until all constraints are satisfied—this approach is used by Tsuda et al. (2005). To demonstrate this more involved case, we consider an example over vectors, where the goal is to minimize the relative entropy to a given vector $\boldsymbol{x}_0$ under linear constraints. Note that the argument carries over to the matrix case, but is more difficult to visualize.

$$\begin{aligned} \text{minimize}_{\boldsymbol{x}} \quad & KL(\boldsymbol{x}, \boldsymbol{x}_0) \\ \text{subject to} \quad & \boldsymbol{x}^T \begin{bmatrix} 0.0912 & 0.9385 & -0.4377 \end{bmatrix} \geq 0.0238 \\ & \boldsymbol{x}^T \begin{bmatrix} 0.6020 & 0.6020 & -0.4377 \end{bmatrix} \geq 0.2554 \\ & \boldsymbol{x}^T \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = 1. \end{aligned} \tag{24}$$

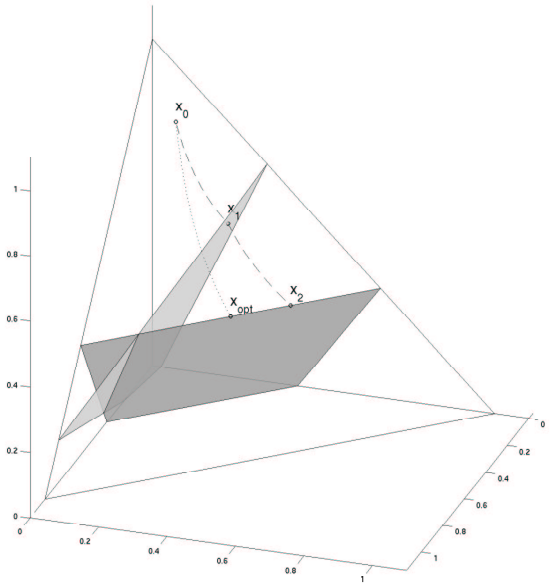

Figure 6: Termination points for the relative entropy example described by (24). The shaded regions illustrate the hyperplanes associated with the two inequality constraints of the optimization problem. The optimal solution is the maximum entropy vector $x_{opt} = \left[\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\right]$, but without corrections we arrive at $x_2 = \left[\frac{1}{5}\ \frac{7}{15}\ \frac{1}{3}\right]$ after two projection steps starting from $x_0 = [0.1\ 0.1\ 0.8]$.

In the above problem, $KL(\boldsymbol{x}, \boldsymbol{x}_0)$ refers to the relative entropy, defined earlier. The first two constraints are linear inequality constraints while the third constraint forces $\boldsymbol{x}$ to remain on the unit simplex so that it is a probability vector. Let $\boldsymbol{x}_0$ equal $[0.1\ 0.1\ 0.8]$. If we process the constraints in the above order without corrections, after two projection steps (one projection onto each of the first two constraints), we arrive at the vector

$$\left[\frac{1}{5} \quad \frac{7}{15} \quad \frac{1}{3}\right],$$

which satisfies all three constraints. Therefore this vector is returned as the optimal solution if no corrections are applied. In comparison, a proper execution of Bregman's algorithm employing corrections arrives at

$$\left[\frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3}\right],$$

which is the maximum entropy vector. It is also worth noting that if we project to the second constraint first, then we arrive at the optimal solution immediately. By appropriately modifying the constraints and the starting point in this example, the incorrect algorithm can be made to converge arbitrarily close to $[0\ 0\ 1]$, a minimum entropy vector, thus drastically demonstrating the necessity of dual variable corrections.

References

F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. In *Proc. 22nd International Conference on Machine Learning (ICML)*, 2005.

J. Barnes and P. Hut. A hierarchical $O(n \log n)$ force calculation algorithm. *Nature*, 324:446–449, 1986.

M. Bilenko, S. Basu, and R. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

L. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Comp. Mathematics and Mathematical Physics*, 7:200–217, 1967.

R.P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ., 1973. ISBN 0-13-022335-2.

Y. Censor and S. Zenios. *Parallel Optimization*. Oxford University Press, 1997.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In *Advances in Neural Information Processing Systems (NIPS) 14*, 2002.

J. Davis, B. Kulis, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *Proc. 24th International Conference on Machine Learning (ICML)*, 2007.

J. D. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

I. S. Dhillon and J. A. Tropp. Matrix nearness problems with Bregman divergences. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1120–1146, 2007.

I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means, spectral clustering and normalized cuts. In *Proc. 10th ACM SIGKDD Conference*, 2004.

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

R. Fletcher. A new variational result for quasi-Newton formulae. *SIAM Journal on Optimziation*, 1 (1), February 1991.

A. Globerson and S. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems (NIPS) 18*, 2005.

G. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15:318–334, 1973.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73: 325–348, 1987.

M. Gu and S. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15:1266–1276, 1994.

N. Higham. Computing the nearest correlation matrix—a problem from finance. *IMA J. Numerical Analysis*, 22(3):329–343, 2002.

P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

W. James and C. Stein. Estimation with quadratic loss. In *Proc. Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 361–379. Univ. of California Press, 1961.

B. Kulis, M. A. Sustik, and I. S. Dhillon. Learning low-rank kernel matrices. In *Proc. 23rd International Conference on Machine Learning (ICML)*, 2006.

B. Kulis, S. Sra, S. Jegelka, and I. S. Dhillon. Scalable semidefinite programming using convex perturbations. Technical Report TR-07-47, University of Texas at Austin, September 2007a.

B. Kulis, A. Surendran, and J. Platt. Fast low-rank semidefinite programming for embedding and clustering. In *Proc. 11th International Conference on AI and Statistics (AISTATS)*, 2007b.

J. Kwok and I. Tsang. Learning with idealized kernels. In *Proc. 20th International Conference on Machine Learning (ICML)*, 2003.

G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

K. Lang. Fixing two weaknesses of the spectral method. In *Advances in Neural Information Processing Systems (NIPS) 18*, 2005.

M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems (NIPS) 16*, 2003.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix. *Annals of Mathematical Statistics*, 20, 1949.

A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI)*, 2000.

M. A. Sustik and I. S. Dhillon. On some modified root-finding problems. Working manuscript, 2008.

L. Torresani and K. Lee. Large margin component analysis. In *Advances in Neural Information Processing Systems (NIPS) 19*, 2006.

K. Tsuda, G. Rátsch, and M. Warmuth. Matrix exponentiated gradient updates for online learning and Bregman projection. *Journal of Machine Learning Research*, 6:995–1018, 2005.

M. K. Warmuth and D. Kuzmin. Randomized PCA algorithms with regret bounds that are logarithmic in the dimension. In *Advances in Neural Information Processing Systems (NIPS) 20*, 2006.

K. Weinberger, F. Sha, and L. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proc. 21st International Conference on Machine Learning (ICML)*, 2004.

K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS) 18*, 2005.

K. Weinberger, F. Sha, Q. Zhu, and L. Saul. Graph Laplacian methods for large-scale semidefinite programming, with an application to sensor localization. In *Advances in Neural Information Processing Systems (NIPS) 19*, 2006.