# Learning Low-Rank Kernel Matrices

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

## Abstract

Kernel learning plays an important role in many machine learning tasks. However, algorithms for learning a kernel matrix often scale poorly, with running times that are cubic in the number of data points. In this paper, we propose efficient algorithms for learning low-rank kernel matrices; our algorithms scale linearly in the number of data points and quadratically in the rank of the kernel. We introduce and employ Bregman matrix divergences for rank-deficient matrices—these divergences are natural for our problem since they preserve the rank as well as positive semi-definiteness of the kernel matrix. Special cases of our framework yield faster algorithms for various existing kernel learning problems. Experimental results demonstrate the effectiveness of our algorithms in learning both low-rank and full-rank kernels.

## 1. Introduction

Kernel methods have played a major role in many recent machine learning algorithms. However, scalability is often a concern: given $n$ input data points, many kernel-based algorithms scale as $O(n^3)$. Furthermore, the kernel matrix requires $O(n^2)$ memory overhead, which may be prohibitive for large-scale learning tasks. Recently, research has been done on using low-rank kernel representations to improve scalability (Fine & Scheinberg, 2001). If the kernel matrix is assumed to be of rank $r$ (with $r < n$), we need only store the decomposition of the kernel matrix $K = GG^T$, where $G$ is $n \times r$. Many kernel-based learning algorithms can be reformulated in terms of $G$, and lead to algorithms that scale linearly with $n$. One of the main obstacles in using low-rank kernel matrices lies in obtaining a matrix with such a property; most standard kernel functions do not produce low-rank kernel matrices, in general.

The focus in this paper is on learning low-rank kernel matrices given distance and similarity constraints on the data. Learning a kernel matrix has been a topic of significant research, but most existing algorithms are not very efficient. Moreover, there is not much literature on learning low-rank kernel matrices. We propose to learn a low-rank kernel by minimizing the divergence to an initial low-rank kernel matrix while satisfying distance and similarity constraints as well as a low-rank constraint. However, low-rank constraints are non-convex, and optimization problems involving such constraints are intractable in general. By introducing specific matrix divergences, we show how to naturally obtain convexity of the optimization problem, leading to algorithms that are substantially more efficient than current kernel learning algorithms.

Our main contributions in this paper are:

- We employ *rank-preserving Bregman matrix divergences*, dissimilarity measures over matrices which are natural for learning low-rank kernel matrices, as they implicitly constrain the rank and maintain positive semi-definiteness during the updates of our algorithms.
- We develop projection algorithms based on the Burg matrix divergence and the von Neumann divergence that scale linearly with the number of data points.
- A special case of our formulation leads to the DefiniteBoost optimization problem of Tsuda et al. (2005). Our approach improves the running time of their algorithm by a factor of $n$, from $O(n^3)$ to $O(n^2)$ per projection. Additional special cases of our formulation lead to improved algorithms for nonlinear dimensionality reduction and the nearest correlation matrix problem.

Appearing in *Proceedings of the 23$^{rd}$ International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

- We experimentally demonstrate that our algorithms can be effectively used in classification and clustering 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 $\mathbf{a}_1, ..., \mathbf{a}_n$, a common step in kernel algorithms is to transform the data using a non-linear function $\psi$. This mapping, typically, represents a transformation of the data to a higher-dimensional feature space. A kernel function is a function $\kappa$ that gives the inner product between two vectors in the feature space:

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

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 $\mathbf{a}_i$, we form an $n \times n$ matrix $K$, called the kernel matrix, whose $(i, j)$ entry corresponds to $\kappa(\mathbf{a}_i, \mathbf{a}_j)$. 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 semi-definite (has non-negative eigenvalues). 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. To improve scalability, the use of low-rank kernel representations has been proposed. 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 decomposition $K = GG^T$, 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, this algorithm was shown to outperform other SVM training algorithms in terms of training time by several orders of magnitude. In clustering, the kernel $k$-means algorithm (Kulis et al., 2005) has a running time of $O(n^2)$ per iteration but can be improved to $O(nrk)$ time per iteration with a low-rank representation, where $k$ is the number of desired clusters. Low-rank kernel representations are often obtained using incomplete Cholesky decompositions (Fine & Scheinberg, 2001). Recently, work has been done on using labeled data to improve the low-rank decomposition (Bach & Jordan, 2005).

In this paper, our focus is on using distance and similarity constraints to learn a low-rank kernel matrix. The problem of learning a kernel matrix has been studied in various contexts. Lanckriet et al. (2004) study transductive learning of the kernel matrix and multiple kernel learning using semi-definite 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, semi-definite programming is used 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 the Burg divergence in addition to the von Neumann divergence.

## 3. Optimization Framework

### 3.1. Bregman Matrix Divergences

Let $\phi$ be a real-valued strictly convex function defined over a convex set $S = \text{dom}(\phi) \subseteq \mathbb{R}^m$ such that $\phi$ is differentiable on the relative interior of $S$. The *Bregman divergence* (Bregman, 1967) with respect to $\phi$ is defined as

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

For example, if $\phi(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$, then the resulting Bregman divergence is $D_\phi(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$. Alternately, if $\phi(\mathbf{x}) = \sum_i (x_i \log x_i - x_i)$, then the resulting Bregman divergence is the (unnormalized) relative entropy. Bregman divergences generalize many properties of squared loss and relative entropy.

We can naturally extend this definition to convex functions defined over matrices. In this case, given a strictly convex, differentiable function $\phi(X)$, the Bregman matrix divergence is defined to be

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

where $\text{tr}(A)$ denotes the trace of matrix $A$. Examples include $\phi(X) = \|X\|_F^2$, which leads to the squared Frobenius norm $\|X-Y\|_F^2$. We consider two other matrix divergences which are less well-known. Let the function $\phi$ compute the (negative) entropy of the eigenvalues of a positive semi-definite matrix. More specifically, if $X$ has eigenvalues $\lambda_1, ..., \lambda_n$, then $\phi(X) = \sum_i(\lambda_i \log \lambda_i - \lambda_i)$, which may be expressed as $\phi(X) = \text{tr}(X \log X - X)$, where $\log X$ is the matrix logarithm.[1] The resulting matrix divergence is

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

and is known as the von Neumann divergence (or quantum relative entropy in the physics literature). Another example is to take the Burg entropy of the eigenvalues, i.e. $\phi(X) = -\sum_i \log \lambda_i$; this may be expressed as $\phi(X) = -\log\det X$. The resulting matrix divergence is

$$D_{Burg}(X,Y) = \text{tr}(XY^{-1}) - \log\det(XY^{-1}) - n, \quad (2)$$

which we call the Burg matrix divergence (or the LogDet divergence).

## 3.2. Problem Description

We now give a formal statement of the problem. Given an input kernel matrix $K_0$, we attempt to solve the following for $K$:

$$\begin{array}{rl} \text{minimize} & D_\phi(K, K_0) \\ \text{subject to} & \text{tr}(KA_i) \le b_i,\ 1 \le i \le c, \\ & \text{rank}(K) \le r, \\ & K \succeq 0. \end{array}$$

Any of the linear inequality constraints above may be replaced with equalities. This problem is non-convex in general, due to the rank constraint. However, when the rank of $K_0$ does not exceed $r$, then this problem surprisingly turns out to be convex when we use rank-preserving Bregman matrix divergences (defined later in Section 4). As we will see later, another advantage of using the von Neumann and Burg divergences is that the algorithms used to solve the minimization problem implicitly maintain the positive semi-definiteness constraint.

Though our algorithms can handle general linear constraints of the form $\text{tr}(KA_i) \le b_i$, we will focus on two types of constraints. The first is a distance constraint. The squared Euclidean distance in feature space between the $i$th and the $j$th data points is given by $K_{ii} + K_{jj} - 2K_{ij}$. Given the constraint $K_{ii} + K_{jj} - 2K_{ij} \le b$, it can be represented as $\text{tr}(KA) \le b$, where $A = \mathbf{z}\mathbf{z}^T$, $z_i = 1$, $z_j = -1$, and all other entries of $\mathbf{z}$ are 0. The constraint $K_{ij} \le b$ can be written as $\text{tr}(KA) \le b$ using $A = \mathbf{x}\mathbf{y}^T$, with $x_j = 1$, $y_i = 1$, and all other entries of $\mathbf{x}$ and $\mathbf{y}$ are 0.

## 3.3. Bregman Projections

Consider the convex optimization problem presented above, without the rank constraint. (We will see how to handle the rank constraint in the next section.) To solve this problem, we use the method of cyclic projections (Bregman, 1967; Censor & Zenios, 1997), which we briefly summarize. Suppose we wish to minimize $f(X) = D_\phi(X, X_0)$ subject to linear equality and inequality constraints. In each iteration of the cyclic projection algorithm, we choose one constraint (assumed to be $\text{tr}(XA_i) = b_i$ or $\text{tr}(XA_i) \le b_i$). If constraint $i$ is an equality constraint, we project our current solution $X_t$ onto constraint $i$ to obtain $X_{t+1}$ by solving the following system of equations uniquely for $\alpha$ and $X_{t+1}$:

$$\begin{aligned} \nabla f(X_{t+1}) &= \nabla f(X_t) + \alpha A_i^T \quad (3) \\ \text{tr}(X_{t+1}A_i) &= b_i. \end{aligned}$$

If constraint $i$ is an inequality constraint, we maintain a non-negative dual variable $\lambda_i$ for that constraint. After solving the above system of equations for $\alpha$, we set $\alpha' = \min(\lambda_i, \alpha)$ and $\lambda_i = \lambda_i - \alpha'$. Then we update $X_{t+1}$ via

$$\nabla f(X_{t+1}) = \nabla f(X_t) + \alpha' A_i^T. \quad (4)$$

We cycle through constraints in such a way that all constraints are visited infinitely often in the limit. Both of our algorithms in Section 5 are based on this method. The main difficulty in using cyclic projections lies in solving the nonlinear system of equations efficiently. Details of the convergence of this method can be found in Censor and Zenios (1997). Note that Tsuda et al. (2005) do not handle inequality constraints correctly, as they fail to make the correction (4) based on the dual variables.

For simplicity, we assume that the constraint matrices are symmetric and rank one matrices: $A_i = \mathbf{z}_i\mathbf{z}_i^T$ (we briefly discuss extensions to higher-rank constraints in Section 5.3). By calculating the gradient for the Burg and the von Neumann matrix divergences, respectively, (3) simplifies to:

$$\begin{aligned} X_{t+1} &= \left(X_t^{-1} - \alpha \mathbf{z}_i\mathbf{z}_i^T\right)^{-1} \quad (5) \\ X_{t+1} &= \exp(\log(X_t) + \alpha \mathbf{z}_i\mathbf{z}_i^T), \quad (6) \end{aligned}$$

subject to $\text{tr}(X_{t+1}\mathbf{z}_i\mathbf{z}_i^T) = b_i$, or equivalently, $\mathbf{z}_i^T X_{t+1}\mathbf{z}_i = b_i$.

[1]If $X = V\Lambda V^T$ is the eigendecomposition of the positive-definite matrix $X$, then its 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.

## 4. Bregman Divergences for Rank-Deficient Matrices

The von Neumann and Burg matrix divergences, as well as the corresponding updates in Bregman's algorithm, are seemingly undefined for low-rank matrices. We now discuss extensions of these divergences to low-rank matrices.

Let the eigendecompositions of $X$ and $Y$ be $X = V\Lambda V^T$ and $Y = U\Theta U^T$. We express the Burg divergence $D_{Burg}(X,Y)$ given in (2) using the eigendecomposition of $X$ and $Y$:

$$\sum_{i,j} \frac{\lambda_i}{\theta_j}(\mathbf{v}_i^T\mathbf{u}_j)^2 - \sum_i \log\left(\frac{\lambda_i}{\theta_i}\right) - n.$$

Similarly, the von Neumann divergence $D_{vN}(X,Y)$ can be written as:

$$\sum_i \lambda_i \log\lambda_i - \sum_{i,j}(\mathbf{v}_i^T\mathbf{u}_j)^2\lambda_i\log\theta_j - \sum_i(\lambda_i - \theta_i).$$

Using continuity arguments, it is possible to show that the Burg divergence between $X$ and $Y$ is finite if and only if the range spaces of $X$ and $Y$ are the same. Similarly, the von Neumann divergence is finite if and only if the range space of $Y$ contains the range space of $X$ (for example, by defining $0\log 0 = 0$, we can verify this for the von Neumann divergence).

This fact has an important implication: when minimizing $D_{Burg}(K,K_0)$, the updates of our algorithm automatically maintain the range space of the input kernel at each iteration as long as the optimization problem is feasible. It follows that the rank of $K$ is equal to the rank of $K_0$ during all updates of the algorithm. Similarly, for the von Neumann divergence, the range space of $K_0$ must contain the range space of $K$, so we conclude that the rank of $K$ never exceeds the rank of $K_0$ during all updates of the algorithm. Thus, we implicitly maintain rank constraints in our optimization problem under both divergences.

We can re-write the divergences in terms of the reduced eigendecompositions of $X$ and $Y$ when they share the same range space. We consider only the top $r$ leading eigenvalues and eigenvectors of $X$ and $Y$, where $r$ is the rank of $Y$. Then the Burg matrix divergence $D_{Burg}(X,Y)$ can be written as

$$\sum_{i,j\le r} \frac{\lambda_i}{\theta_j}(\mathbf{v}_i^T\mathbf{u}_j)^2 - \sum_{i\le r} \log\left(\frac{\lambda_i}{\theta_i}\right) - r,$$

where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ and $\theta_1 \geq \theta_2 \geq ... \geq \theta_n$. The von Neumann divergence $D_{vN}(X,Y)$ is expressed using the reduced eigendecomposition as

$$\sum_{i\le r} \lambda_i \log\lambda_i - \sum_{i,j\le r}(\mathbf{v}_i^T\mathbf{u}_j)^2\lambda_i\log\theta_j - \sum_{i\le r}(\lambda_i - \theta_i).$$

## 5. Algorithms

### 5.1. Burg Divergence

#### 5.1.1. Matrix Updates

Consider minimizing $D_{Burg}(X, X_0)$, the Burg matrix divergence between $X$ and $X_0$. Recall the projection update rule (5). To accommodate low-rank kernels, we can write the update as:

$$X_{t+1} = (X_t^\dagger - \alpha\mathbf{z}_i\mathbf{z}_i^T)^\dagger,$$

where $X_t^\dagger$ is the pseudoinverse of the positive semi-definite matrix $X_t$ (Golub & Van Loan, 1996). Alternatively, we could have written this update using the more "procedural" reduced eigendecomposition of $X_t$. Note that the limit of $((X_t + \epsilon I)^{-1} - \alpha\mathbf{z}_i\mathbf{z}_i^T)^{-1}$ as $\epsilon \to 0$ leads to the above update, justifying our use of the pseudoinverse. Note that in the full-rank case, the above update is the same as (5). We apply the Sherman-Morrison inverse formula to this update, which can be extended to use the pseudoinverse of a positive semi-definite matrix:

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

Using this formula, we arrive at a simplified expression for $X_{t+1}$:

$$X_{t+1} = X_t + \frac{\alpha X_t\mathbf{z}_i\mathbf{z}_i^TX_t}{1 - \alpha\mathbf{z}_i^TX_t\mathbf{z}_i}.$$

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

$$\mathrm{tr}\left(\left(X_t + \frac{\alpha X_t\mathbf{z}_i\mathbf{z}_i^TX_t}{1 - \alpha\mathbf{z}_i^TX_t\mathbf{z}_i}\right)\mathbf{z}_i\mathbf{z}_i^T\right) = b_i.$$

Let $p = \mathbf{z}_i^TX_t\mathbf{z}_i$. Note that $\mathrm{tr}(X_t\mathbf{z}_i\mathbf{z}_i^T) = p$ and $\mathrm{tr}(X_t\mathbf{z}_i\mathbf{z}_i^TX_t\mathbf{z}_i\mathbf{z}_i^T) = p^2$. In the case of distance constraints, $p$ is the distance between the two data points corresponding to constraint $i$, and can be computed in constant time. Then we have:

$$\alpha = \frac{1}{p} - \frac{1}{b_i}.$$

If we let $\beta = \alpha/(1 - \alpha p)$, then our matrix update is given by

$$X_{t+1} = X_t + \beta X_t\mathbf{z}_i\mathbf{z}_i^TX_t.$$

This update is pleasantly surprising since the projection parameter for Bregman's algorithm does not usually admit a closed form solution (see Section 5.2.2 for the case of the von Neumann divergence).

### 5.1.2. Update for the Factored Matrix

If one stopped with the update given above, the cost is $O(n^2)$ per iteration. However, we can achieve a more efficient update for low-rank matrices by working on a suitable factored form of the matrix $X_t$. If $X_t$ can be written as $GG^T$, where $G$ is an $n \times r$ matrix, the update can be formulated using only the factored matrices. Let $\beta$ be as above. Then we have:

$$\begin{aligned} X_{t+1} &= GG^T + \beta GG^T \mathbf{z}_i \mathbf{z}_i^T GG^T \\ &= G(I + \beta G^T \mathbf{z}_i \mathbf{z}_i^T G)G^T. \end{aligned}$$

The matrix $I + \beta \tilde{\mathbf{z}}_i \tilde{\mathbf{z}}_i^T$, where $\tilde{\mathbf{z}}_i = G^T \mathbf{z}_i$, 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 $I + \beta \tilde{\mathbf{z}}_i \tilde{\mathbf{z}}_i^T$ is a rank-one perturbation of the identity, this update can be done in $O(r^2)$ time using a 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 $G_0$ is the initial Cholesky factor of $K_0$. Instead of updating $G$ explicitly at each iteration, we can just update $B$ as $BL$. The matrix $I + \beta G^T \mathbf{z}_i \mathbf{z}_i^T G$ is then $I + \beta B^T G_0^T \mathbf{z}_i \mathbf{z}_i^T G_0 B$. In the case of distance constraints, we can compute $G_0^T \mathbf{z}_i$ in $O(r)$ time as the difference of two rows of $G_0$. Furthermore, $\alpha$, $\beta$ and the Cholesky factorization can all be computed in $O(r^2)$ time. The multiplication $BL$ appears to cost $O(r^3)$ time, but the simple structure of $L$ and the fact that it depends on $O(r)$ parameters allow us to implement this multiplication in $O(r^2)$ time as well. Details are omitted due to lack of space.

### 5.1.3. Algorithm

The algorithm for distance inequality constraints using the Burg divergence is given as Algorithm 1. As discussed in the previous section, every projection can be done in $O(r^2)$ time. Thus, cycling through all $c$ projections requires $O(cr^2)$ time. The only dependence on $n$—the number of data points—occurs in steps 1 and 4. If we need to explicitly create the matrix $G_0$, this takes $O(nr)$ time, and the last step, multiplying $G = G_0 B$, takes $O(nr^2)$ time. Hence, the algorithm is linear in $n$ (and linear in $c$).

Convergence can be checked by using the dual variables $\lambda$. The cyclic projection algorithm can be viewed as a dual ascent algorithm, thus convergence can be measured as follows: after cycling through all constraints, we check to see how much $\lambda$ has changed after a full cycle. At convergence, this change (as measured with a vector norm) should be small.

Algorithm 1: Learning a low-rank kernel in Burg divergence under distance constraints.

```
KernelLearnBurg($r, \{A_i\}_{i=1}^c, G_0$)
Input: $r$: rank of desired kernel matrix, $\{A_i\}_{i=1}^c$: constraints, $G_0$: optional input kernel factor matrix of rank $r$
Output: $G$: output low-rank kernel factor matrix
1. If not given, create a random $n \times r$ matrix $G_0$.
2. Set $B = I_r$, $i = 1$, and $\lambda_j = 0 \;\; \forall$ constraints $j$.
3. Repeat until convergence:
    • Let $\mathbf{v}^T$ be row $i_1$ of $G_0$ minus row $i_2$ of $G_0$, where, corresponding to constraint $i$, points $i_1$ and $i_2$ are constrained to have squared Euclidean distance less than or equal to $b_i$.
    • Set the following variables:
        $\mathbf{w} \leftarrow B^T \mathbf{v}$
        $\alpha \leftarrow \min\left(\lambda_i, \frac{1}{\|\mathbf{w}\|_2^2} - \frac{1}{b_i}\right)$
        $\lambda_i \leftarrow \lambda_i - \alpha$
        $\beta \leftarrow \alpha/(1 - \alpha\|\mathbf{w}\|_2^2)$
    • Compute the Cholesky factorization $LL^T = I + \beta \mathbf{w}\mathbf{w}^T$.
    • Set $B \leftarrow BL$, $i \leftarrow \mathrm{mod}(i, c) + 1$.
4. Return $G = G_0 B$.
```

## 5.2. Von Neumann Divergence

In this section we develop a cyclic projection algorithm 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 equation (6), the projection update rule for constraint $i$. As with the Burg divergence, we can express the von Neumann divergence update using the reduced eigendecomposition of $X_t$. Let $X_t = V_t \Lambda_t V_t^T$, where $V_t$ is $n \times r$ and $\Lambda_t$ is $r \times r$. We must calculate $\exp(\log(X_t) + \alpha \mathbf{z}_i \mathbf{z}_i^T)$. When $X_t$ is of full rank, the definitions of the log and exp functions imply that $\exp(\log(X_t) + \alpha \mathbf{z}_i \mathbf{z}_i^T) = V_t \exp(\log(\Lambda_t) + \alpha V_t^T \mathbf{z}_i \mathbf{z}_i^T V_t) V_t^T$. This motivates the following natural extension of the update when $X_t$ is low rank:

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

We can verify that the matrix on the right-hand side is equal to the limit of $\exp(\log(X_t + \epsilon I) + \alpha \mathbf{z}_i \mathbf{z}_i^T)$ when $\epsilon \rightarrow 0$, thus justifying the update.

If the eigendecomposition of the exponent $\log(\Lambda_t) + \alpha V_t^T \mathbf{z}_i \mathbf{z}_i^T V_t$ is $U_t \Theta_t U_t^T$, then the eigendecomposition of $X_{t+1}$ is calculated by $V_{t+1} = V_t U_t$ and $\Lambda_{t+1} = \exp(\Theta_t)$. This special eigenvalue problem (diagonal plus rank-one update) can be solved in $O(r^2)$ time; see 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 a $r \times r$ orthogonal matrix, while $V_t$ and $\Lambda_t$ are defined as before. The matrix update may be written as:

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

yielding the following updates:

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

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

### 5.2.2. Computing the Projection Parameter

In the previous section, we assumed that we knew the projection parameter $\alpha$. Unlike in the case of the Burg divergence, this parameter does not have a closed form solution. Instead, we must compute $\alpha$ by solving the nonlinear system of equations given in (3).

The main computational difficulty in finding $\alpha$ is in calculating $\mathbf{z}_i^T X_{t+1} \mathbf{z}_i$ for a given $\alpha$. Using the approach from Section 5.2.1, we have that $\mathbf{z}_i^T X_{t+1} \mathbf{z}_i = \mathbf{z}_i^T V_{t+1} W_{t+1} \Lambda_{t+1} W_{t+1}^T V_{t+1}^T \mathbf{z}_i$, and for a given choice of $\alpha$, this trace can be computed in $O(r^2)$ time. We built a custom nonlinear solver that is optimized for this problem, as a result of which we can accurately compute $\alpha$ using only a few trace computations (rarely more than six trace evaluations per projection). Thus, the projection parameter can effectively be computed in $O(r^2)$ time.

### 5.2.3. Algorithm

The algorithm for distance inequality constraints using the von Neumann divergence is given as Algorithm 2. By using the fast multipole method, every projection can be done in $O(r^2)$ time. The asymptotic running time of this algorithm is the same as the algorithm that uses the Burg divergence.

Algorithm 2: Learning a low-rank kernel in von Neumann divergence under distance constraints.

```
KernelLearnVonNeumann($r, \{A_i\}_{i=1}^c, V_0, \Lambda_0$)
Input: $r$: rank of desired kernel matrix, $\{A_i\}_{i=1}^c$: constraints, $V_0, \Lambda_0$: optional input kernel of rank $r$ in factored form
Output: $V$: output low-rank kernel factor matrix
1. If not given, create a random $n \times r$ orthogonal matrix $V_0$ and $r \times r$ diagonal matrix $\Lambda_0$.
2. Set $W = I_r$, $\Lambda = \Lambda_0$, $i = 1$, and $\lambda_j = 0\ \forall$ constraints $j$.
3. Repeat until convergence:
    • Let $\mathbf{v}^T$ be row $i_1$ of $V_0$ minus row $i_2$ of $V_0$, where, corresponding to constraint $i$, points $i_1$ and $i_2$ are constrained to have squared Euclidean distance less than or equal to $b_i$.
    • Set the following variables:
        $\mathbf{w} \leftarrow W^T \mathbf{v}$
        $\alpha \leftarrow$ ComputeProj($\mathbf{v}, \mathbf{w}, W, \Lambda, b_i$)
        $\beta \leftarrow \min(\lambda_i, \alpha)$
        $\lambda_i \leftarrow \lambda_i - \beta$
    • Compute the eigendecomposition $U \Theta U^T = \Lambda + \beta \mathbf{w}\mathbf{w}^T$.
    • Set $W \leftarrow WU$, $\Lambda \leftarrow \exp(\Theta)$, $i \leftarrow \mathrm{mod}(i, c) + 1$.
4. Return $V = V_0 W \Lambda^{1/2}$.
```

## 5.3. Generalizations and Special Cases

In the previous two sections, we focused on the case of distance constraints. In this section, we briefly discuss generalizations to other constraints, and discuss important special cases of our optimization problem.

When the constraints are similarity constraints (i.e. $K_{ij} \leq b_i$), the updates can be modified easily (details are omitted due to lack of space). Other constraints are possible as well; for example, one could incorporate further distance constraints (such as $\|\mathbf{a}_i - \mathbf{a}_j\|_2^2 \leq \|\mathbf{a}_l - \mathbf{a}_m\|_2^2$) or further similarity constraints (such as $K_{ij} \leq K_{lm}$). Arbitrary linear constraints can also be applied, the cost per projection update will then be $O(nr)$.

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 of our approach is that we compute the exact projection, whereas Tsuda et al. (2005) compute an approximate projection. Computing an approximate projection may lead to a faster per-iteration cost, but it takes more iterations to converge to the optimal solution. We illustrate this further in Section 6.

Another important special case is the *nearest correlation matrix* problem (Higham, 2002), which is an important problem that arises in applications in finance. In this problem, we set the constraints to be $K_{ii} = 1$ for all $i$. Thus, we seek to find the nearest positive semi-definite matrix with unit diagonal. Our algorithms from this paper give new methods of finding low-rank correlation matrices. Previous algorithms scale cubically in $n$.

Our formulation can also be employed for nonlinear dimensionality reduction, as in 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. Experiments

To show the effectiveness of our algorithms, we present both clustering and classification results. We use two data sets from real-life applications:

1. `Digits` data: 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, and our subset contains 317 digits.

2. `GyrB` protein data: 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).

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 2-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. NMI measures the amount of statistical information shared by the random variables representing the cluster and class distributions.

Constraints for the `GyrB` data set were generated randomly as follows: two data points are chosen at random and a distance constraint is constructed. If the data points are in the same class, the constraint is of the form $d(i_1, i_2) \leq b_i$, where $b_i$ is the distance from the original kernel matrix, and for different classes, we construct $d(i_1, i_2) \geq b_i$ constraints. For the `Digits` data set, we added constraints of the form $d(i_1, i_2) \leq (1-\epsilon)b_i$ for same class pairs and $d(i_1, i_2) \geq (1+\epsilon)b_i$ for different class pairs ($b_i$ is the original distance and $\epsilon = .25$). This is very similar to “idealizing” the kernel, as in Kwok and Tsang (2003). Our convergence tolerance was $10^{-3}$.

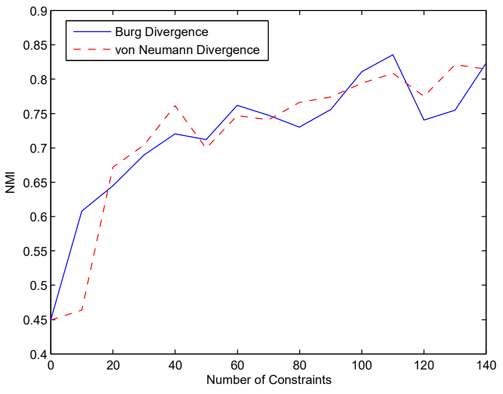

*Figure 1.* Results of clustering the `Digits` data set

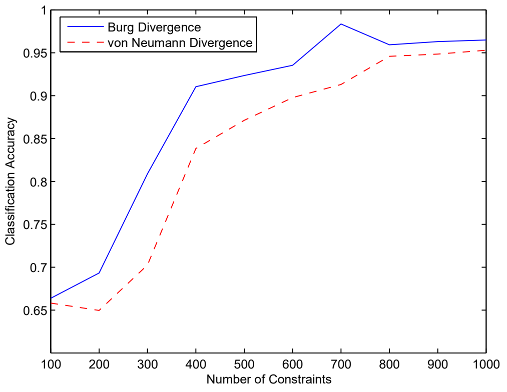

*Figure 2.* Classification accuracy with the `gyrB` data set

### 6.1. Results

We first ran our algorithms on the `Digits` data set to learn a rank-16 kernel matrix using the randomly generated constraints. The Gram matrix of the data set (a 317x317 rank-16 matrix formed using a linear kernel) was used as our initial kernel matrix. Figure 1 shows NMI values for the clustering 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 both divergences). We also recorded convergence data in terms of the number of cycles (sweeps) through all constraints; for the von Neumann divergence, convergence was attained from 11 sweeps for few constraints to 105 sweeps for many constraints, and for the Burg divergence, between 17 and 354 sweeps were needed for convergence. This experiment highlights that our algorithm can use constraints to learn a low-rank kernel matrix (rank 16 as opposed to a full rank of 317). It is noteworthy that the learned kernel performs better than the original kernel for clustering and classification.

As a second experiment, we performed a comparison to the DefiniteBoost algorithm of Tsuda et al. (2005) (modified to correctly handle inequality constraints) using the `GyrB` data set. Using only constraints, we attempt to learn a kernel matrix that achieves high classification accuracy. As in the experiments of Tsuda et al. (2005), we learned a full-rank kernel matrix starting from the scaled identity matrix. In our experiments, we observed that using approximate projections—as done in DefiniteBoost—increases the number of sweeps needed for convergence considerably. For example, starting with the scaled identity as the initial kernel matrix and 100 constraints, it took our von Neumann algorithm only 11 sweeps to converge, whereas it took 3220 sweeps 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 sweeps.

The slow convergence of the DefiniteBoost algorithm did not allow us to run it with a larger set of constraints. For the Burg and the von Neumann exact projection algorithms, the number of sweeps required for convergence never exceeded 600 on runs of up to 1000 constraints. Figure 2 depicts the classification accuracy achieved versus the number of constraints. 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 scaled identity matrix, and so did not encode any domain information.

## 7. Conclusions

In this paper, we have developed algorithms for learning low-rank kernel matrices. By exploiting the rank-preserving property of Bregman matrix divergences, we were able to obtain convexity of our optimization problem. Unlike previous kernel learning algorithms, which have running times that are cubic in the number of data points, our algorithms are highly 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 effectively learned low-rank and full-rank kernels for classification and clustering problems.

**Acknowledgements** 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 and the anonymous reviewers for their helpful suggestions.

## References

Bach, F., & Jordan, M. (2005). Predictive low-rank decomposition for kernel methods. *Proc. ICML-2005.*

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

Bregman, L. (1967). 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.

Censor, Y., & Zenios, S. (1997). *Parallel optimization.* Oxford University Press.

Demmel, J. D. (1997). *Applied numerical linear algebra.* Society for Industrial and Applied Mathematics.

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

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations.* Johns Hopkins University Press.

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

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

Kulis, B., Basu, S., Dhillon, I., & Mooney, R. (2005). Semi-supervised graph clustering: A kernel approach. *Proc. ICML-2005.*

Kwok, J., & Tsang, I. (2003). Learning with idealized kernels. *Proc. ICML-2003.*

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

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis.* Cambridge University Press.

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

Weinberger, K., Sha, F., & Saul, L. (2004). Learning a kernel matrix for nonlinear dimensionality reduction. *Proc. ICML-2004.*