Efficient Kernel Learning from Side Information Using ADMM

En-Liang Hu[†,‡] **James T. Kwok**[†]
[†]Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
[‡]Department of Mathematics
Yunnan Normal University, Yunnan, China
{jamesk, ynelhu }@cse.ust.hk

Abstract

Side information is highly useful in the learning of a nonparametric kernel matrix. However, this often leads to an expensive semidefinite program (SDP). In recent years, a number of dedicated solvers have been proposed. Though much better than off-the-shelf SDP solvers, they still cannot scale to large data sets. In this paper, we propose a novel solver based on the alternating direction method of multipliers (ADMM). The key idea is to use a low-rank decomposition of the kernel matrix $\mathbf{K} = \mathbf{V}^\top\mathbf{U}$, with the constraint that $\mathbf{V} = \mathbf{U}$. The resultant optimization problem, though non-convex, has favorable convergence properties and can be efficiently solved without requiring eigen-decomposition in each iteration. Experimental results on a number of real-world data sets demonstrate that the proposed method is as accurate as directly solving the SDP, but can be one to two orders of magnitude faster.

1 Introduction

Kernel methods have been highly successful in various aspects of machine learning, such as classification, regression, clustering, ranking, and dimensionality reduction [Schölkopf and Smola, 2002]. Because of the central role of the kernel, it is important to identify an appropriate kernel function or matrix for the task at hand. Over the past decade, there have been a large body of literature on this kernel learning problem [Lanckriet *et al.*, 2004; Bach *et al.*, 2004; Cortes *et al.*, 2009]. While a parametric form of the kernel [Chapelle *et al.*, 2003] or a combination of multiple kernels [Sonnenburg *et al.*, 2006] are often assumed, nonparametric kernel learning, which takes no such assumptions, is more flexible and has received significant interest in recent years [Li *et al.*, 2008; Wu *et al.*, 2009; Kulis *et al.*, 2009; Hu *et al.*, 2011; Zhuang *et al.*, 2011; Shang *et al.*, 2012].

To facilitate kernel learning, obviously one has to utilize information from the data. The most straightforward approach is to use class labels. However, obtaining label information may sometimes be expensive and time-consuming. In this paper, we focus on a weaker form of supervisory information, namely, the so-called *must-link* and *cannot-link* pairwise constraints [Wagstaff *et al.*, 2001]. These pairwise constraints, or *side information*, define whether the two patterns involved should belong to the same class or not. Another useful source of information, which is commonly used in semi-supervised learning, is the data manifold [Belkin *et al.*, 2006]. This encourages patterns that are locally nearby on the manifold to have similar predicted labels.

A notable nonparametric kernel learning approach that nicely integrates these two sources of information is the *pairwise constraint propagation* (PCP) algorithm [Li *et al.*, 2008]. Let n be the number of patterns, and $\mathcal{M}, \mathcal{C}$ be the sets of must-link and cannot-link pairs, respectively. PCP is formulated as the following optimization problem

$$\min_{\mathbf{K}\succeq\mathbf{0}} \ tr(\mathbf{KL}) + \frac{\gamma}{2}\sum_{(i,j)\in\mathcal{T}}(K_{ij} - T_{ij})^2, \tag{1}$$

where $\mathbf{K} = [K_{ij}] \in \mathbb{R}^{n\times n}$ is the kernel matrix to be learned (which has to be symmetric and positive semidefinite (psd), denoted $\mathbf{K} \succeq \mathbf{0}$), $\mathbf{L}$ is the graph Laplacian matrix of the data manifold, $\mathbf{T} = [T_{ij}]$ with

$$T_{ij} = \begin{cases} 1 & i = j, \\ 1 & (i,j) \in \mathcal{M}, \\ 0 & (i,j) \in \mathcal{C}, \end{cases} \tag{2}$$

$\mathcal{T} = \mathcal{M}\cup\mathcal{C}\cup\{(i,i)\}_{i=1}^n$, and γ is a regularization parameter. In words, the $tr(\mathbf{KL})$ term in (1) encourages $\mathbf{K}$ to be aligned with $\mathbf{L}$ and thus smooth on the manifold, while $(K_{ij} - T_{ij})^2$ encourages K_{ij} to be close to 1 for two must-link patterns i, j, and 0 for cannot-link patterns. Note from (2) that all the patterns are also encouraged to be normalized in the kernel representation (i.e., $K_{ii} \simeq 1$).

As in other nonparametric kernel learning algorithms, the optimization problem in (1) is a semidefinite program (SDP) [Vandenberghe and Boyd, 1996]. Though off-the-shelf solvers such as CSDP can be used [Li *et al.*, 2008], they are slow, particularly when n is large or the pairwise constraints are abundant. Recently, Laue [2012] proposed a more efficient hybrid solver that is based on gradually increasing the rank of the $\mathbf{K}$ solution. However, each time the rank is increased, it has to perform an approximate eigen-decomposition and nonlinear optimization. As will be demonstrated in Section 4.2, this can be inefficient on large kernel learning problems.

Recently, there have been a lot of interest in scaling up this SDP. In many machine learning problems, a prominent tool

that often drastically reduces the computational effort is low-rank approximation [Fine and Scheinberg, 2001]. In the context of kernel learning, this involves replacing the $n \times n$ matrix $\mathbf{K}$ by $\mathbf{V}^\top \mathbf{V}$, where $\mathbf{V} \in \mathbb{R}^{r \times n}$ and r is the rank. The solver in [Burer and Choi, 2006] can then be used, though the line search and computation of the coefficients in each iteration can be expensive. Alternatively, one can optimize $\mathbf{V}$ column-by-column using coordinate descent [Hu *et al.*, 2011]. The resultant algorithm is scalable, but the decoupling of variables as required in coordinate descent makes it unable to handle the (soft) constraints of $K_{ii} \simeq 1$ in (2).

Another popular low-rank approximation option is to replace $\mathbf{K}$ by $\mathbf{Q}\mathbf{U}\mathbf{Q}^\top$, as in [Li and Liu, 2009; Wu *et al.*, 2009] and the *modified fixed point continuation* (MFPC) algorithm [Shang *et al.*, 2012]. Here, $\mathbf{Q} \in \mathbb{R}^{n \times r}$ is a fixed matrix with orthogonal columns (usually the trailing eigenvectors of the graph Laplacian matrix $\mathbf{L}$), and $\mathbf{U} \in \mathbb{R}^{r \times r}$ is the psd matrix to be optimized. Typically, $r \ll n$, and thus optimization of $\mathbf{U}$ is much more manageable. Moreover, Wu *et al.* [2009] showed that problem (1) belongs to a special class of SDP called quadratic semidefinite program (QSDP). By exploiting this structure, they reformulated (1) as a semidefinite-quadratic-linear program (SQLP), which reduces the complexity from $O(n^9)$ for the standard QSDP formulation to $O(n^{6.5})$. However, empirically, this still does not scale well. As reported in [Wu *et al.*, 2009], kernel learning on a data set with 2000 instances and 100 pairwise constraints takes about 4000 seconds. Moreover, the fixed $\mathbf{Q}$ limits the ability of $\mathbf{K}$ to fit the data, leading to possibly deteriorated performance of the learned model.

Instead of solving problem (1) directly, Zhuang *et al.* [2011] noted that this, together with other similar machine learning problems, can be reformulated as optimizing the inner product of the matrix variable with another fixed matrix. By adding an additional constraint $tr(\mathbf{K}\mathbf{K}) \leq B$, where $B > 0$ is a user-defined constant, this optimization problem has a closed-form solution of $\mathbf{K}$. The resultant algorithm, called *simple nonparametric kernel learning* (SimpleNPKL), is significantly more efficient than existing algorithms. However, computing the closed-form $\mathbf{K}$ solution requires eigen-decomposition at each iteration, which can be expensive on large kernel learning problems with a large number of instances or constraints. Moreover, its performance is sometimes sensitive to the setting of B.

In this paper, our key idea is to decompose the matrix $\mathbf{K}$ as $\mathbf{V}^\top \mathbf{U}$, where $\mathbf{V}, \mathbf{U} \in \mathbb{R}^{r \times n}$, together with the constraint that $\mathbf{V} = \mathbf{U}$. Obviously, this is essentially the same as the commonly-used low-rank approximation $\mathbf{V}^\top \mathbf{V}$ discussed above [Burer and Choi, 2006; Hu *et al.*, 2011; Li and Liu, 2009], but with the important difference that $\mathbf{V}$ and $\mathbf{U}$ are now decoupled. Together with the QSDP structure in (1), this allows simple and efficient optimization based on the alternating direction method of multipliers (ADMM) [Boyd *et al.*, 2011], which has been popularly used in diverse fields such as machine learning, data mining and image processing. Experimental results show that the proposed method is as accurate as solving the SDP directly, but is significantly more efficient and scalable.

The rest of this paper is organized as follows. Section 2 first gives a brief introduction to ADMM. Section 3 then describes the proposed solver and its properties. Experimental results on a number of real-world data sets are presented in Section 4, and the last section gives some concluding remarks.

Notations: In the sequel, matrices and vectors are denoted in bold, with upper-case letters for matrices and lower-case for vectors. The transpose of a vector/matrix is denoted by the superscript $^\top$, $tr(\mathbf{A})$ denotes the trace of matrix $\mathbf{A}$, $\mathbf{A} \odot \mathbf{B}$ is the element-wise multiplication of matrices $\mathbf{A}$ and $\mathbf{B}$ (i.e., $[\mathbf{A} \odot \mathbf{B}]_{ij} = A_{ij}B_{ij}$), and $\mathbf{A} \bullet \mathbf{B} = tr(\mathbf{A}^\top \mathbf{B})$ is their inner product. Moreover, $\mathbf{I}$ is the identity matrix, and $\mathbf{0}$ (resp. $\mathbf{1}$) is the vector (or matrix) of all zeros (resp. ones).

2 Alternating Direction Method of Multipliers (ADMM)

ADMM is a simple but powerful algorithm first introduced in [Glowinski and Marrocco, 1975], and has recently been successfully used in diverse fields such as machine learning, data mining and image processing. The standard ADMM is for solving convex problems. Here, we consider the more general bi-convex problem in [Boyd *et al.*, 2011]:

$$\min_{\mathbf{x},\mathbf{y}} F(\mathbf{x}, \mathbf{y}) \ : \ G(\mathbf{x}, \mathbf{y}) = \mathbf{0}, \tag{3}$$

where $F(\cdot, \cdot)$ is bi-convex and $G(\cdot, \cdot)$ is bi-affine[1]. As in the method of multipliers, ADMM considers the augmented Lagrangian of (3): $\mathscr{L}(\mathbf{x}, \mathbf{y}, \mathbf{\Lambda}) = F(\mathbf{x}, \mathbf{y}) + \mathbf{\Lambda}^\top G(\mathbf{x}, \mathbf{y}) + \frac{\rho}{2} \|G(\mathbf{x}, \mathbf{y})\|^2$, where $\mathbf{\Lambda}$ is the vector of Lagrangian multipliers, and $\rho > 0$ is a penalty parameter. At the kth iteration, the values of $\mathbf{x}, \mathbf{y}$ and $\mathbf{\Lambda}$ (denoted $\mathbf{x}^k, \mathbf{y}^k$ and $\mathbf{\Lambda}^k$) are updated as

$$\begin{aligned} \mathbf{x}^{k+1} &= \arg\min_{\mathbf{x}} \mathscr{L}(\mathbf{x}, \mathbf{y}^k, \mathbf{\Lambda}^k), \\ \mathbf{y}^{k+1} &= \arg\min_{\mathbf{y}} \mathscr{L}(\mathbf{x}^{k+1}, \mathbf{y}, \mathbf{\Lambda}^k), \\ \mathbf{\Lambda}^{k+1} &= \mathbf{\Lambda}^k + \rho G(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}). \end{aligned}$$

Note that while the method of multipliers minimizes $\mathscr{L}(\mathbf{x}, \mathbf{y}, \mathbf{\Lambda}^k)$ w.r.t. $\mathbf{x}$ and $\mathbf{y}$ jointly, ADMM allows easier decomposition of the optimization problem by minimizing them in an alternating manner.

3 Kernel Learning by ADMM

In this section, we present a novel solver for problem (1) based on the ADMM. Section 3.1 shows that the update steps can be efficiently computed. A necessary condition for convergence is provided in Section 3.2, and Section 3.3 discusses the time complexity in each iteration.

As in [Wu *et al.*, 2009; Hu *et al.*, 2011; Li and Liu, 2009], we use a low-rank approximation on the kernel matrix $\mathbf{K}$. In other words, $\mathbf{K}$ is approximated as

$$\mathbf{K} \simeq \mathbf{V}^\top \mathbf{V}, \tag{4}$$

where $\mathbf{V} \in \mathbb{R}^{r \times n}$ and rank $r < n$. Problem (1) can then be rewritten as

$$\min_{\mathbf{V}} \ tr(\mathbf{K}\mathbf{L}) + \frac{\gamma}{2} \sum_{(i,j) \in \mathcal{T}} (K_{ij} - T_{ij})^2 \ : \ \mathbf{K} = \mathbf{V}^\top \mathbf{V}.$$

[1] In other words, for any fixed $\mathbf{x}, \mathbf{y}$, $F(\cdot, \mathbf{y})$ and $F(\mathbf{x}, \cdot)$ are convex; while $G(\cdot, \mathbf{y})$ and $G(\mathbf{x}, \cdot)$ are affine.

Recall that the graph Laplacian matrix $\mathbf{L}$ is symmetric [Luxburg, 2007], this can be further written as

$$\min_{\mathbf{V}-\mathbf{U}=\mathbf{0}} tr(\mathbf{V}\mathbf{L}\mathbf{U}^\top) + \frac{\gamma}{2}\sum_{(i,j)\in\mathcal{T}}(\mathbf{v}_i^\top\mathbf{u}_j - T_{ij})^2, \quad (5)$$

where $\mathbf{U} \in \mathbb{R}^{r\times n}$, and $\mathbf{v}_i$'s (resp. $\mathbf{u}_i$'s) are the columns of $\mathbf{V}$ (resp. $\mathbf{U}$). Note that the objective is bi-convex, and the constraint function $\mathbf{V}-\mathbf{U}$ is bi-affine.

3.1 ADMM Update Steps

Let $\mathbf{\Lambda}$ be the Lagrangian multiplier for the constraint $\mathbf{V}-\mathbf{U}=\mathbf{0}$. As in Section 2, the augmented Lagrangian of (5) is

$$\begin{aligned}\mathscr{L}(\mathbf{U},\mathbf{V},\mathbf{\Lambda}) &= tr(\mathbf{V}\mathbf{L}\mathbf{U}^\top) + \frac{\gamma}{2}\sum_{(i,j)\in\mathcal{T}}(\mathbf{v}_i^\top\mathbf{u}_j - T_{ij})^2 \\ &+ \mathbf{\Lambda}\bullet(\mathbf{V}-\mathbf{U}) + \frac{\rho}{2}\|\mathbf{V}-\mathbf{U}\|^2, \quad (6)\end{aligned}$$

and the update steps at the kth iteration of ADMM are

$$\mathbf{V}^{k+1} = \arg\min_{\mathbf{V}} \mathscr{L}(\mathbf{U}^k,\mathbf{V},\mathbf{\Lambda}^k), \quad (7)$$

$$\mathbf{U}^{k+1} = \arg\min_{\mathbf{U}} \mathscr{L}(\mathbf{U},\mathbf{V}^{k+1},\mathbf{\Lambda}^k), \quad (8)$$

$$\mathbf{\Lambda}^{k+1} = \mathbf{\Lambda}^k + \rho(\mathbf{V}^{k+1}-\mathbf{U}^{k+1}).$$

It can be easily shown that

$$\frac{\partial\mathscr{L}(\mathbf{U}^k,\mathbf{V},\mathbf{\Lambda}^k)}{\partial\mathbf{v}_i} = \mathbf{A}_i^k\mathbf{v}_i^k - \mathbf{c}_i^k, \quad (9)$$

$$\frac{\partial\mathscr{L}(\mathbf{U},\mathbf{V}^{k+1},\mathbf{\Lambda}^k)}{\partial\mathbf{u}_i} = \mathbf{B}_i^k\mathbf{u}_i^k - \mathbf{d}_i^k, \quad (10)$$

where

$$\mathbf{A}_i^k = \rho\mathbf{I} + \gamma\sum_{j\in\mathcal{T}_i}\mathbf{u}_j^k{\mathbf{u}_j^k}^\top, \quad (11)$$

$$\mathbf{B}_i^k = \rho\mathbf{I} + \gamma\sum_{j\in\mathcal{T}_i}\mathbf{v}_j^{k+1}{\mathbf{v}_j^{k+1}}^\top, \quad (12)$$

$$\mathbf{c}_i^k = \gamma\sum_{j\in\mathcal{T}_i}T_{ij}\mathbf{u}_j^k - \sum_{s=1}^n L_{is}\mathbf{u}_s^k + \rho\mathbf{u}_i^k - \mathbf{\Lambda}_i^k,$$

$$\mathbf{d}_i^k = \gamma\sum_{j\in\mathcal{T}_i}T_{ij}\mathbf{v}_i^{k+1} - \sum_{s=1}^n L_{is}\mathbf{v}_s^{k+1} + \rho\mathbf{v}_i^{k+1} + \mathbf{\Lambda}_i^k,$$

$\mathbf{\Lambda}_i^k$ is the ith column of $\mathbf{\Lambda}^k$, and $\mathcal{T}_i = \{j \mid (i,j)\in\mathcal{T}\}$.

On setting (9), (10) to zero, the solutions of (7) and (8) can be obtained as

$$\mathbf{v}_i^{k+1} = (\mathbf{A}_i^k)^{-1}\mathbf{c}_i^k,\ \mathbf{u}_i^{k+1} = (\mathbf{B}_i^k)^{-1}\mathbf{d}_i^k, \quad (13)$$

for $i = 1,\cdots,n$. Note from (11), (12) that $\mathbf{A}_i^k$ and $\mathbf{B}_i^k$ are always invertible.

To monitor convergence of the algorithm, we require the primal and dual residuals at iteration $k+1$

$$\Delta_{\text{primal}} = \left\|\mathbf{V}^{k+1}-\mathbf{U}^{k+1}\right\|,\ \Delta_{\text{dual}} = \rho\left\|\mathbf{V}^{k+1}-\mathbf{V}^k\right\| \quad (14)$$

to be small [Boyd *et al.*, 2011]. Moreover, to improve convergence, it is common to vary the penalty parameter ρ in each ADMM iteration. Specifically, following [Boyd *et al.*, 2011], we update ρ as

$$\rho \leftarrow \begin{cases} 2\rho & \Delta_{\text{primal}} > 10\Delta_{\text{dual}}, \\ \max(\rho/2, 10) & \Delta_{\text{dual}} > 10\Delta_{\text{primal}}. \end{cases} \quad (15)$$

The whole procedure is shown in Algorithm 1.

Algorithm 1 KL_{ADMM}: Kernel learning by ADMM.

Input: rank r, $\mathbf{V}^0 = [\mathbf{v}_1^0,\ldots,\mathbf{v}_n^0]$, $\mathbf{U}^0 = [\mathbf{u}_1^0,\ldots,\mathbf{u}_n^0]$, parameters ρ, ε and $IterMax$.
Output: $\mathbf{K} = {\mathbf{V}^k}^\top\mathbf{V}^k$ (or ${\mathbf{U}^k}^\top\mathbf{U}^k$).
$k \leftarrow 0$;
repeat
  for $i = 1$ **to** n **do**
    $\mathbf{v}_i^{k+1} \leftarrow (\mathbf{A}_i^k)^{-1}\mathbf{c}_i^k,\ \mathbf{u}_i^{k+1} \leftarrow (\mathbf{B}_i^k)^{-1}\mathbf{d}_i^k$;
  end for
  $\mathbf{\Lambda}^{k+1} \leftarrow \mathbf{\Lambda}^k + \rho(\mathbf{V}^{k+1}-\mathbf{U}^{k+1})$;
  compute the primal and dual residuals in (14);
  update ρ using (15);
  $k \leftarrow k+1$;
until $\max(\Delta_{\text{primal}}, \Delta_{\text{dual}}) < \varepsilon$ or $k > IterMax$;

3.2 Convergence

With the low-rank decomposition in (4), problem (5) is non-convex in $\mathbf{V}$ and so we can only consider local convergence [Boyd *et al.*, 2011]. As in [Xu *et al.*, 2011], we show below a necessary condition for local convergence.

Lemma 1 *Let $\{(\mathbf{U}^k,\mathbf{V}^k,\mathbf{\Lambda}^k)\}$ be a sequence generated by Algorithm 1. If the sequence $\{\mathbf{\Lambda}^k\}$ is bounded and satisfies*

$$\sum_{k=0}^{\infty}\left\|\mathbf{\Lambda}^{k+1}-\mathbf{\Lambda}^k\right\| < \infty, \quad (16)$$

then $\mathbf{V}^k-\mathbf{V}^{k+1}\to\mathbf{0}$ and $\mathbf{U}^k-\mathbf{U}^{k+1}\to\mathbf{0}$.

Proof 1 *First, observe that $\mathscr{L}(\mathbf{U},\mathbf{V},\mathbf{\Lambda})$ is bounded, which follows from the boundedness of $\mathbf{\Lambda}$ and*

$$\begin{aligned}\mathscr{L}(\mathbf{U},\mathbf{V},\mathbf{\Lambda}) &= tr(\mathbf{V}\mathbf{L}\mathbf{U}^\top) + \frac{\gamma}{2}\sum_{(i,j)\in\mathcal{T}}(\mathbf{v}_i^\top\mathbf{u}_j - T_{ij})^2 \\ &+ \frac{\rho}{2}\left\|\mathbf{V}-\mathbf{U}+\frac{\mathbf{\Lambda}}{\rho}\right\|^2 - \frac{1}{2\rho}\|\mathbf{\Lambda}\|^2.\end{aligned}$$

Moreover, $\mathscr{L}$ is strongly convex w.r.t. each coordinate variable of $\mathbf{v}_i$ and $\mathbf{u}_i$. For any $\mathbf{v}_i$, it holds that

$$\begin{aligned}&\mathscr{L}(\mathbf{v}_i+\triangle\mathbf{v}_i,\cdot) - \mathscr{L}(\mathbf{v}_i,\cdot) \\ &\geq \partial_{\mathbf{v}_i}\mathscr{L}(\mathbf{v}_i,\cdot)^\top\triangle\mathbf{v}_i + \rho\|\triangle\mathbf{v}_i\|^2 \quad (17)\end{aligned}$$

for any $\triangle\mathbf{v}_i$, where $\mathscr{L}(\mathbf{v}_i,\cdot)$ denotes that all the variables in (7) except $\mathbf{v}_i$ are fixed. Moreover, $\mathbf{v}_i^\star$ is a minimizer of $\mathscr{L}(\mathbf{v}_i,\cdot)$ if

$$\partial_{\mathbf{v}_i}\mathscr{L}(\mathbf{v}_i^\star,\cdot)^\top\triangle\mathbf{v}_i \geq 0. \quad (18)$$

Note that $\mathbf{v}_i^{k+1}$ is a minimizer of $\mathscr{L}(\mathbf{v}_i,\cdot)$ at the kth iteration. Then, on combining (17) and (18), we have

$$\mathscr{L}(\mathbf{v}_i^k,\cdot) - \mathscr{L}(\mathbf{v}_i^{k+1},\cdot) \geq \rho\left\|\mathbf{v}_i^k-\mathbf{v}_i^{k+1}\right\|^2. \quad (19)$$

Similarly, one can show that

$$\mathscr{L}(\mathbf{u}_i^k,\cdot) - \mathscr{L}(\mathbf{u}_i^{k+1},\cdot) \geq \rho\left\|\mathbf{u}_i^k-\mathbf{u}_i^{k+1}\right\|^2. \quad (20)$$

Using (17) – (20) and $\mathbf{\Lambda}^{k+1} - \mathbf{\Lambda}^k = \rho(\mathbf{V}^{k+1} - \mathbf{U}^{k+1})$*, we have*

$$\begin{aligned}
&\mathscr{L}(\mathbf{U}^k, \mathbf{V}^k, \mathbf{\Lambda}^k) - \mathscr{L}(\mathbf{U}^{k+1}, \mathbf{V}^{k+1}, \mathbf{\Lambda}^{k+1}) \\
&= \mathscr{L}(\mathbf{U}^k, \mathbf{V}^k, \mathbf{\Lambda}^k) - \mathscr{L}(\mathbf{U}^k, \mathbf{V}^{k+1}, \mathbf{\Lambda}^k) \\
&\quad + \mathscr{L}(\mathbf{U}^k, \mathbf{V}^{k+1}, \mathbf{\Lambda}^k) - \mathscr{L}(\mathbf{U}^{k+1}, \mathbf{V}^{k+1}, \mathbf{\Lambda}^k) \\
&\quad + \mathscr{L}(\mathbf{U}^{k+1}, \mathbf{V}^{k+1}, \mathbf{\Lambda}^k) - \mathscr{L}(\mathbf{U}^{k+1}, \mathbf{V}^{k+1}, \mathbf{\Lambda}^{k+1}) \\
&\geq \rho \sum_{i=1}^{n} \left\| \mathbf{v}_i^k - \mathbf{v}_i^{k+1} \right\|^2 + \rho \sum_{i=1}^{n} \left\| \mathbf{u}_i^k - \mathbf{u}_i^{k+1} \right\|^2 \\
&\quad - \frac{1}{\rho} \left\| \mathbf{\Lambda}^k - \mathbf{\Lambda}^{k+1} \right\|^2 \\
&= \rho \left(\left\| \mathbf{V}^k - \mathbf{V}^{k+1} \right\|^2 + \left\| \mathbf{U}^k - \mathbf{U}^{k+1} \right\|^2 \right) \\
&\quad - \frac{1}{\rho} \left\| \mathbf{\Lambda}^k - \mathbf{\Lambda}^{k+1} \right\|^2 .
\end{aligned}$$

Taking summation of the above inequality from 1 to ∞*, and recall that* $\mathscr{L}$ *is bounded, then*

$$\rho \sum_{k=1}^{\infty} \left(\left\| \mathbf{V}^k - \mathbf{V}^{k+1} \right\|^2 + \left\| \mathbf{U}^k - \mathbf{U}^{k+1} \right\|^2 \right) - \frac{1}{\rho} \sum_{k=1}^{\infty} \left\| \mathbf{\Lambda}^k - \mathbf{\Lambda}^{k+1} \right\|^2 < \infty.$$

Since the second term on the left is bounded, it follows that $\sum_{k=1}^{\infty} \left(\left\| \mathbf{V}^k - \mathbf{V}^{k+1} \right\|^2 + \left\| \mathbf{U}^k - \mathbf{U}^{k+1} \right\|^2 \right) < \infty$*, from which we immediately have* $\mathbf{V}^k - \mathbf{V}^{k+1} \to \mathbf{0}$ *and* $\mathbf{U}^k - \mathbf{U}^{k+1} \to \mathbf{0}$*.* ■

Proposition 1 *Let* $\{(\mathbf{U}^k, \mathbf{V}^k)\}$ *be a sequence generated by Algorithm 1. Then any accumulation point of* $\{(\mathbf{U}^k, \mathbf{V}^k)\}$ *satisfies the Karush-Kuhn-Tucker (KKT) condition for problem (5).*

Proof 2 *From (13), we have*

$$\begin{aligned}
\mathbf{A}_i^k(\mathbf{v}_i^{k+1} - \mathbf{v}_i^k) &= \mathbf{c}_i^k - \mathbf{A}_i^k \mathbf{v}_i^k, &(21)\\
\mathbf{B}_i^k(\mathbf{u}_i^{k+1} - \mathbf{u}_i^k) &= \mathbf{d}_i^k - \mathbf{B}_i^k \mathbf{u}_i^k, &(22)\\
\mathbf{\Lambda}^{k+1} - \mathbf{\Lambda}^k &= \rho(\mathbf{V}^{k+1} - \mathbf{U}^{k+1}). &(23)
\end{aligned}$$

Lemma 1 implies that the left- and right-hand sides of (21), (22), (23) all go to zero, i.e.,

$$\mathbf{c}_i^k - \mathbf{A}_i^k \mathbf{v}_i^k \to \mathbf{0},\ \mathbf{d}_i^k - \mathbf{B}_i^k \mathbf{u}_i^k \to \mathbf{0},\ \mathbf{U}^k - \mathbf{V}^k \to \mathbf{0}, \quad (24)$$

and thus the KKT conditions for problem (5), namely,

$$\begin{aligned}
\mathbf{A}_i \mathbf{v}_i - \mathbf{c}_i &= \mathbf{0},\\
\mathbf{B}_i \mathbf{u}_i - \mathbf{d}_i &= \mathbf{0},\\
\mathbf{V} - \mathbf{U} &= \mathbf{0}
\end{aligned}$$

are satisfied. ■

In particular, note from (24) that although the algorithm does not guarantee $\mathbf{U}^k = \mathbf{V}^k$ at each iteration k, the difference goes to zero as the iteration proceeds. This will also be experimentally demonstrated in Section 4.1.

3.3 Time Complexity

As $\mathbf{A}_i^k, \mathbf{B}_i^k$ are of size $r \times r$, computing their inverses in (13) takes $O(r^3)$ time. In practice, we often have $r > \mathcal{T}_i$. For example, please refer to the experimental setup in Section 4. By rewriting $\mathbf{A}_i^k$ as $\gamma \left(\frac{\rho}{\gamma} \mathbf{I} + \mathbf{P}_i \mathbf{P}_i^\top \right)$, where $\mathbf{P}_i = [\mathrm{u}_{j_1}^k, \ldots, \mathrm{u}_{j_{|\mathcal{T}_i|}}^k] \in \mathbb{R}^{r \times |\mathcal{T}_i|}$, and using the Sherman-Morrison-Woodbury (SMW) identity [Golub and Van Loan, 1996], we have

$$\begin{aligned}
(\mathbf{A}_i^k)^{-1} &= \frac{1}{\gamma} \left(\frac{\rho}{\gamma} \mathbf{I} + \mathbf{P}_i \mathbf{P}_i^\top \right)^{-1} \\
&= \frac{1}{\rho} \left(\mathbf{I} - \mathbf{P}_i (\frac{\rho}{\gamma} \mathbf{I} + \mathbf{P}_i^\top \mathbf{P}_i)^{-1} \mathbf{P}_i^\top \right), \quad (25)
\end{aligned}$$

where $\frac{\rho}{\gamma} \mathbf{I} + \mathbf{P}_i^\top \mathbf{P}_i$ is of size $|\mathcal{T}_i| \times |\mathcal{T}_i|$. Hence, computing $(\mathbf{A}_i^k)^{-1}$ takes only $O(|\mathcal{T}_i|^3 + r|\mathcal{T}_i|)$ time. The same applies to the computing of $(\mathbf{B}_i^k)^{-1}$. The time complexity for each iteration of Algorithm 1 is then

$$O\left(rn + \sum_{i=1}^{n} \left(|\mathcal{T}_i|^3 + r|\mathcal{T}_i| \right) \right). \quad (26)$$

In practice, $|\mathcal{T}_i|$ is usually very small, and (26) essentially scales linearly with n.

4 Experiments

As in [Hoi *et al.*, 2007; Li *et al.*, 2008], we study the performance of the proposed approach in the context of data clustering. Specifically, a kernel matrix is learned from the pairwise constraints, which is then used for clustering by the kernel k-means algorithm. Experiments are performed on a number of benchmark data sets (Table 1) that have been commonly used for nonparametric kernel learning [Hoi *et al.*, 2007; Hu *et al.*, 2011; Zhuang *et al.*, 2011].

Table 1: Data sets used in the experiment.

data set	#classes	#patterns	#features	#constraints	avg($\lvert\mathcal{T}_i\rvert$)	r
chessboard	2	100	2	340	2.2	25
dbl-spiral	2	100	3	340	2.2	25
glass	6	214	9	728	2.2	37
heart	2	270	13	918	2.2	42
iris	3	150	4	510	2.2	31
protein	6	116	20	396	2.2	27
sonar	2	208	60	708	2.2	37
soybean	4	47	35	161	2.2	17
wine	3	178	12	606	2.2	34
adult-a1a	2	1,605	123	5,457	2.2	103
adult-a2a	2	2,265	123	7,701	2.2	123
adult-a3a	2	3,185	123	10,829	2.2	146
adult-a4a	2	4,781	123	16,257	2.2	179
adult-a5a	2	6,414	123	21,808	2.2	208
adult-a6a	2	11,220	123	38,148	2.2	275

The constructions of the data manifold and pairwise constraints follow that in [Hoi *et al.*, 2007; Hu *et al.*, 2011;

Zhuang *et al.*, 2011]. Specifically, the similarity between patterns $\mathbf{x}_i, \mathbf{x}_j$ is defined as $\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$, where σ is equal to half of the average distance between each pattern and its top-10 nearest neighbor. The 50-nearest-neighbor graph is then used for the manifold of the *adult* data sets, and the 5-nearest-neighbor graph for the others. As for the pairwise constraints, we randomly select $0.6n$ pairs of patterns that belong to the same class to construct the must-link pairs, another $0.6n$ pairs of patterns that belong to different classes to construct the cannot-link pairs, and another n must-link (i, i) pairs for each pattern i. On all the data sets, the average number of constraints (i.e., $|\mathcal{T}_i|$) associated with a random pattern i is always around 2.

As for the rank of the kernel matrix, we follow [Zhuang *et al.*, 2011; Hu *et al.*, 2011] and set its value to be the largest r satisfying $r(r+1)/2 \leq m$, where m is the total number of constraints in $\mathcal{T}$. Table 1 shows the rank values (i.e., r) used. As can be seen, they are all much larger than $|\mathcal{T}_i| \simeq 2$, i.e., the average number of constraints associated with a random pattern i. This justifies the use of the SMW identity to compute the matrix inverse in Section 3.3.

The proposed KL_{ADMM} algorithm[2] is compared with the following solvers discussed in Section 1. Similar to KL_{ADMM}, all are based on a rank-r approximation of $\mathbf{K}$.

1. The hybrid algorithm [Laue, 2012], with the rank of the $\mathbf{K}$ solution increasing up to r;
2. SimpleNPKL[3] [Zhuang *et al.*, 2011], which is extended for the quadratic loss in (1);
3. MFPC [Shang *et al.*, 2012], which is based on the low-rank representation $\mathbf{QUQ}^\top$, where the columns of $\mathbf{Q}$ are the r trailing eigenvectors of the graph Laplacian $\mathbf{L}$;
4. The SQLP solver [Wu *et al.*, 2009], which is also based on the same low-rank representation as MFPC.

All the algorithms are implemented in MATLAB and run on a PC with i7-3370 3.4GHz CPU and 32G RAM.

The following performance evaluation measures are used:

1. percentage of pattern pairs that are correctly clustered together [Hoi *et al.*, 2007]:

$$\text{accuracy} = \sum_{i>j} \frac{I(I(c_i = c_j) = I(\hat{c}_i = \hat{c}_j))}{0.5n(n-1)},$$

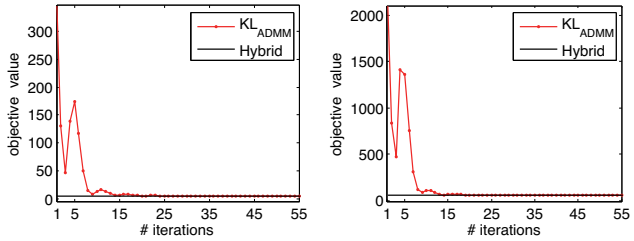
   where c_i (resp. $\hat{c}_i$) is the true (resp. predicted) cluster label of pattern $\mathbf{x}_i$, and $I(\cdot)$ is the indicator function that returns 1 when the argument is true, and 0 otherwise;
2. CPU time for learning the kernel matrix $\mathbf{K}$.

4.1 Accuracy and Speed

Results on the clustering accuracy and CPU time are shown in Table 2. SQLP, which shares the same low-rank model as MFPC, has comparable accuracy as MFPC but is much slower (often by one to two orders of magnitude). Hence, its results are not reported here because of the lack of space. Moreover, some solvers become very slow on the larger data sets and so are not run to completion.

As can be seen, KL_{ADMM} is much faster than SimpleNPKL, which in turn is faster than the hybrid algorithm. In particular, on the largest data set (*adult-a4a*) that SimpleNPKL can handle, KL_{ADMM} is about 30 times faster. MFPC, though is almost as fast as KL_{ADMM}, has much inferior accuracy, and cannot be improved even by increasing the size of $\mathbf{U}$. As discussed in Section 1, for efficient optimization, MFPC has to fix the $\mathbf{Q}$ matrix in the low-rank representation ($\mathbf{QUQ}^\top$) of $\mathbf{K}$. This severely restricts the ability of $\mathbf{K}$ to adapt to the data.

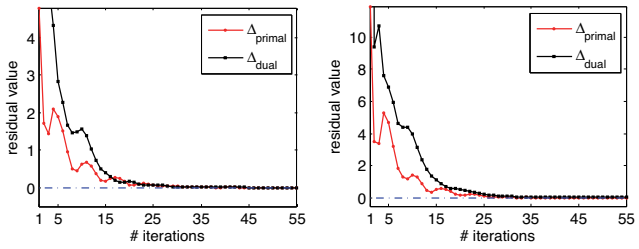
On the other hand, KL_{ADMM}, though involves non-convex optimization, is as accurate as the hybrid algorithm which converges to the globally optimal solution of the SDP problem [Laue, 2012]. Figure 1 shows the speed in which the objective value obtained by KL_{ADMM} converges towards the globally optimal objective of the hybrid algorithm. As can be seen, KL_{ADMM} often converges in fewer than 25 iterations.



(a) soybean. (b) glass.

Figure 1: Convergence of the objective value of KL_{ADMM} on the *soybean* and *glass* data sets.

Finally, Figure 2 shows the progress of KL_{ADMM}'s primal and dual residuals with the number of iterations. Recall from (14) that the primal residual Δ_{primal} measures the difference between $\mathbf{V}^k$ and $\mathbf{U}^k$. As can be seen, this difference quickly goes to zero, as is discussed in Section 3.2.



(a) soybean. (b) glass.

Figure 2: The primal's and dual's residual value for KL_{ADMM} on the *soybean* and *glass* data sets.

[2] $\mathbf{V}^0$ and $\mathbf{U}^0$ is initialized randomly, $\rho = 100$, and $IterMax$ set to 500 (and to 100 on the *adult* datasets).

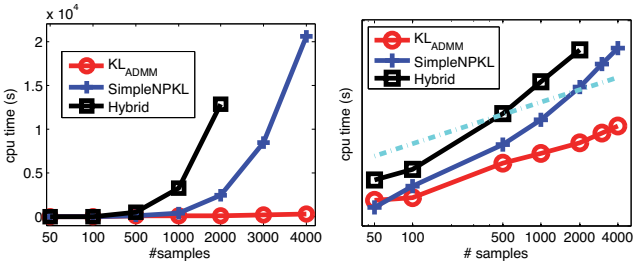
[3] Recall that an additional B parameter is needed to constrain $tr(\mathbf{KK})$. Preliminary results show that the solution quality can be sensitive to B. Here, we set $B = 100n$, which performs better in our experiments than the value suggested in [Zhuang *et al.*, 2011].

Table 2: Comparison on clustering accuracy (%) and CPU time (second) on the data sets. The best and comparable results (according to the pairwise t-test with 95% confidence) are highlighted.

	clustering accuracy (%)				CPU time (sec)			
data set	KL_{ADMM}	SimpleNPKL	hybrid	MFPC	KL_{ADMM}	SimpleNPKL	hybrid	MFPC
chessboard	**91.28±4.17**	89.09±5.33	**91.28±4.22**	71.09±4.87	3.08±0.05	17.06±0.34	38.95±3.57	**1.22±0.23**
dbl-spiral	**99.70±0.96**	98.71±1.27	**99.70±0.96**	**99.68±2.31**	**0.74±0.23**	10.44±0.40	15.61±2.62	1.32±0.06
glass	83.56±2.02	**84.24±1.87**	**83.70±1.73**	77.25±1.39	17.56±0.18	71.38±1.75	231.43±22.12	**4.03±0.78**
heart	**78.88±2.79**	78.03±3.93	**79.00±2.75**	58.34±4.09	37.14±0.16	113.85±2.73	400.60±37.63	**5.64±0.45**
iris	**98.69±1.08**	97.55±1.05	**98.69±1.10**	97.36±0.88	3.44±0.19	35.26±8.26	53.48±3.13	**1.90±0.60**
protein	**90.34±2.21**	87.93±1.97	**90.27±2.20**	85.39±1.87	3.91±0.18	22.44±0.56	54.34±4.23	**1.66±0.17**
sonar	**91.54±2.49**	89.53±2.34	**91.59±2.54**	71.84±3.39	16.28±0.21	62.35±2.20	174.27±15.22	**3.39±0.13**
soybean	**98.73±3.84**	96.10±5.07	**98.73±4.40**	93.80±5.85	0.77±0.18	1.96±0.09	5.80±0.73	**0.20±0.11**
wine	**84.11±2.29**	**83.79±2.36**	**84.14±2.54**	76.37±2.69	6.89±0.20	45.58±1.20	118.70±11.06	**2.82±0.28**
adult-a1a	**95.85±1.35**	85.68±2.06	-	66.29±1.83	**54.99±0.78**	365.68±26.33	-	120.42±14.32
adult-a2a	**95.65±0.58**	85.70±1.46	-	65.46±1.07	**92.02±1.64**	988.12±23.13	-	157.08±2.06
adult-a3a	**93.16±0.93**	83.68±1.07	-	62.07±2.31	**161.60±3.25**	2800.0±120.04	-	332.34±1.81
adult-a4a	**93.23±0.46**	82.97±0.62	-	61.94±2.82	**319.29±2.71**	8767.8±334.44	-	821.76±71.70
adult-a5a	**92.93±0.44**	-	-	63.21±0.88	1724.9±3.23	-	-	**1407.0±135.92**
adult-a6a	**93.36±0.30**	-	-	65.17±1.62	7661.7±4.20	-	-	**4786.6±207.36**

4.2 Scalability

In this section, we compare the scalability of the various algorithms, except for MFPC and SQLP which have been shown to have much inferior accuracy (Section 4.1). Experiments are performed on a synthetic 10-dimensional binary classification data set. Patterns from the first class are generated from the normal distribution $\mathcal{N}(\mathbf{1}, \mathbf{I})$, while those from the second class are from $\mathcal{N}(-\mathbf{1}, \mathbf{I})$.



(a) plot using the normal scale. (b) log-log plot.

Figure 3: CPU time vs the number of patterns.

In the first experiment, we fix the number of constraints at 1,000. Figure 3(a) shows the CPU time as the number of patterns n increases. As in Section 4.1, KL_{ADMM} scales much better than the others. The curves are redrawn as a log-log plot in Figure 3(b). A line corresponding to a linear scaling between time and n is also shown. As can be seen, KL_{ADMM} scales almost linearly in n, which agrees with the analysis in Section 3.3. In the second experiment, we fix n at 1,000 and gradually increase the number of constraints (recall that the rank r also increases according to our experimental setup). Results are shown in Figure 4. As can be seen, KL_{ADMM} is again much more efficient than the other methods.

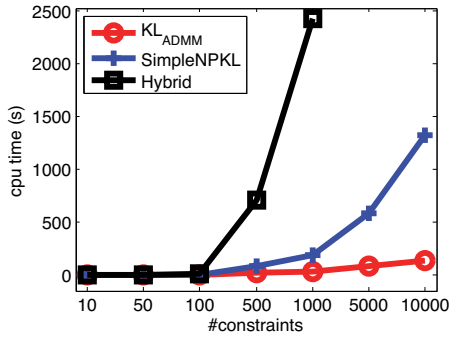


Figure 4: CPU time vs the number of constraints.

5 Conclusion

In this paper, we proposed an efficient solver for nonparametric low-rank kernel learning. Using ADMM, the proposed KL_{ADMM} decouples the original optimization problem into computationally inexpensive subproblems. Unlike existing solvers which perform a large eigen-decomposition in each iteration, KL_{ADMM} only requires operations on very small matrices. Experimental results on a number of real-world data sets demonstrate that KL_{ADMM} is as accurate as directly solving the SDP, but is much faster than existing solvers. Its CPU time scales linearly with the number of patterns, and can be used on much larger kernel learning problems.

In the future, the proposed solver will be extended to optimize other nonparametric kernel learning models.

Acknowledgment

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant No. 614012), the National Natural Science Foundation of China (No. 61165012) , the Natural Science Foundations Fund (Grant No. 2011FZ074) and the Department of Education Fund (No. 2011Y305) of Yunnan Province.

References

[Bach *et al.*, 2004] F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the Twenty-First International Conference on Machine*, Banff, Alberta, Canada, July 2004.

[Belkin *et al.*, 2006] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

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

[Burer and Choi, 2006] S. Burer and C. Choi. Computational enhancements in low-rank semidefinite programming. *Optimization Methods and Software*, 21(3):493–512, 2006.

[Chapelle *et al.*, 2003] O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2003.

[Cortes *et al.*, 2009] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning non-linear combinations of kernels. In *Advances in Neural Information Processing Systems*, 2009.

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

[Glowinski and Marrocco, 1975] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualite, d'une classe de problems de dirichlet non lineares. *Revue Francaise d'Automatique, Informatique, et Recherche Opérationelle*, 9:41–76, 1975.

[Golub and Van Loan, 1996] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Hopkins University Press, Baltimore, Maryland, USA, 3rd edition, 1996.

[Hoi *et al.*, 2007] S.C.H. Hoi, R. Jin, and M.R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 361–368, Corvallis, Oregon, USA, June 2007.

[Hu *et al.*, 2011] E.-L. Hu, B. Wang, and S. Chen. BCD-NPKL: Scalable non-parametric kernel learning using block coordinate descent. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, pages 209–216, Bellevue, WA, USA, June 2011.

[Kulis *et al.*, 2009] B. Kulis, M. Sustik, and I. Dhillon. Low-rank kernel learning with Bregman matrix divergences. *Journal of Machine Learning Research*, 10:341–376, 2009.

[Lanckriet *et al.*, 2004] G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[Laue, 2012] S. Laue. A hybrid algorithm for convex semidefinite optimization. In *Proceedings of the Twenty-Nineth International Conference on Machine Learning*, pages 1206–4608, Edinburgh, Scotland, June 2012.

[Li and Liu, 2009] Z. Li and J. Liu. Constrained clustering by spectral regularization. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 421–428, Miami, Florida, USA, June 2009.

[Li *et al.*, 2008] Z. Li, J. Liu, and X. Tang. Pairwise constraint propagation by semidefinite programming for semi-supervised classification. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 576–583, Helsinki, Finland, July 2008.

[Luxburg, 2007] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[Schölkopf and Smola, 2002] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[Shang *et al.*, 2012] F. Shang, L.C. Jiao, and F. Wang. Semi-supervised learning with mixed knowledge information. In *Proceedings of the Eighteenth Conference on Knowledge Discovery and Data Mining*, pages 732–740, Beijing, China, August 2012.

[Sonnenburg *et al.*, 2006] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, July 2006.

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

[Wagstaff *et al.*, 2001] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, Williamstown, MA, USA, June 2001.

[Wu *et al.*, 2009] X.-M. Wu, A. M.-C. So, Z. Li, and S.-Y.R. Li. Fast graph Laplacian regularized kernel learning via semidefinite-quadratic-linear programming. In *Advances in Neural Information Processing Systems*, pages 1964–1972, 2009.

[Xu *et al.*, 2011] Y. Xu, W. Yin, Z. Wen, and Y. Zhang. An alternating direction algorithm for matrix completion with nonnegative factors. Technical Report TR11-03, Department of Computational and Applied Mathematics, Rice University, 2011.

[Zhuang *et al.*, 2011] J. Zhuang, I.W. Tsang, and S.C.H. Hoi. A family of simple non-parametric kernel learning algorithms. *Journal of Machine Learning Research*, 12:1313–1347, 2011.