Efficient Hyperkernel Learning Using Second-Order Cone Programming

Ivor Wai-Hung Tsang and James Tin-Yau Kwok

***Abstract*—The kernel function plays a central role in kernel methods. Most existing methods can only adapt the kernel parameters or the kernel matrix based on empirical data. Recently, Ong *et al.* introduced the method of hyperkernels which can be used to learn the kernel function directly in an inductive setting. However, the associated optimization problem is a semidefinite program (SDP), which is very computationally expensive, even with the recent advances in interior point methods. In this paper, we show that this learning problem can be equivalently reformulated as a second-order cone program (SOCP), which can then be solved more efficiently than SDPs. Comparison is also made with the kernel matrix learning method proposed by Lanckriet *et al.* Experimental results on both classification and regression problems, with toy and real-world data sets, show that our proposed SOCP formulation has significant speedup over the original SDP formulation. Moreover, it yields better generalization than Lanckriet *et al.*'s method, with a speed that is comparable, or sometimes even faster, than their quadratically constrained quadratic program (QCQP) formulation.**

***Index Terms*—Hyperkernel, kernel learning, second-order cone program (SOCP), semidefinite program (SDP).**

I. Introduction

In RECENT years, kernels have been successfully used in various aspects of machine learning, such as classification, regression, clustering, ranking, and principal component analysis [3]–[5]. The basic idea is to map the data in the input space $\mathcal{X}$ to a feature space via some nonlinear map φ, and then apply a linear method there. It is now well-known that the computational procedure depends only on the inner products[1] $\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$ in the feature space (where $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}$), which can be obtained efficiently from a suitable *kernel* function $k(\cdot,\cdot)$. Besides, kernel methods have the important computational advantage that no nonconvex nonlinear optimization is involved. Thus, the use of kernels provides elegant nonlinear generalizations of many existing linear algorithms.

Because of the central role of the kernel, a poor kernel choice can lead to significantly impaired performance. Typically, the practitioner has to decide the kernel function before learning starts, with common choices being the polynomial kernel, Gaussian kernel and Laplacian kernel. The associated kernel parameters (such as the order in the polynomial kernel and the width in the Gaussian or Laplacian kernel) can then be determined by optimizing a quality functional of the kernel [1], such as kernel target alignment [6], generalization error bounds [7], [5], Bayesian probabilities [8], [9], cross-validation error [10], and class separability [11].

Instead of adapting only the kernel parameters, recent developments also adapt the form of the kernel itself [1], [6], [12]–[16]. In a transductive setting, as all information on the feature space is encoded in the kernel matrix (with entries for both the training and test patterns), one can bypass the learning of kernel function by just learning the kernel *matrix* instead. As the kernel matrix must be positive semidefinite (psd), Lanckriet *et al.* [14], [2] used semidefinite programming (SDP) to optimize a cost function (such as the hard/soft margin of the resultant SVM classifier) on the training set over the set of psd matrices. To avoid overfitting, capacity of the search space has to be controlled. Inspired from a generalization bound for transduction, Lanckriet *et al.* [2] constrained the kernel matrix to be in a convex subset of psd matrices with a fixed trace. When the kernel-target alignment [6] is used as the cost function instead, this method can also be shown to be a generalization of the kernel matrix learning method proposed in [6]. Other kernel matrix learning methods, such as using boosting to optimize a weighted combination of base kernels [13] and the use of Bayesian inference with a hierarchical model [16], have also been recently proposed.

However, transduction requires knowing the test patterns in advance and, thus, may not always be appropriate. In an induction setting, a novel approach that learns the kernel *function* directly is the method of hyperkernels [1], [17]–[18]. By introducing the notion of a hyper-RKHS, the desired kernel function can be obtained by minimizing a regularized quality functional, in which the capacity of the search space is explicitly penalized by a regularization term. It can be shown that the kernel function obtained is always a linear combination of a finite number of prespecified hyperkernel evaluations. Often, this is further constrained to be a positive linear combination so as to ensure that the resultant kernel is always a valid kernel. As will be detailed in Section II, learning with hyperkernels involves optimizing two sets of variables. The first set of variables, $\{\alpha_1, \ldots, \alpha_m\}$ where m is the number of training samples, are coefficients in the kernel expansion, while the second set, $\{\beta_1, \ldots, \beta_{m^2}\}$, are coefficients in the hyperkernel expansion. Originally, these two sets have to be optimized separately in an alternating manner [1]. Recently, simultaneous optimization of both sets of variables is made possible by formulating this as a SDP problem [17].

However, even with the recent advances in interior point methods, solving SDP problems is still very computationally

Manuscript received January 17, 2005; revised January 17, 2005.

The authors are with the Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (e-mail: ivor@cs.ust.hk; jamesk@cs.ust.hk).

Digital Object Identifier 10.1109/TNN.2005.860848

[1]In this paper, vector/matrix transpose (in both the input and feature spaces) is denoted by the superscript ^{T}.

expensive. In the method of hyperkernels, this is further aggravated by the fact that $O(m^2)$ variables, instead of $O(m)$ variables as in other kernel methods, are involved. In [17] and [1], this problem is partially alleviated by reducing the number of variables using a low rank approximation (e.g., [19] and [20]) on the hyperkernel matrix. While this often makes the SDP problem more tractable, an alternative formulation faster than SDP is still very desirable.

On the other hand, as the kernel function obtained by the hyperkernel method is a positive linear combination of hyperkernel evaluations, the corresponding kernel matrix is, consequently, a positive linear combination of some prespecified matrices. Rather than regarding this as a derived property, one can treat it as a constraint in the kernel learning process and then apply Lanckriet *et al.*'s method [2] in such an induction setting.[2] In particular, the computational problem reduces to a quadratically constrained quadratic program (QCQP), which can be solved much faster than SDP problems. Recently, Bousquet and Herrmann [21] proposed an even faster method based on the use of gradient descent. Note that the hyperkernel method cannot be similarly reduced to a QCQP because it uses the hyperkernel prior, while [21] and [2] use the trace of the kernel matrix for capacity control. However, although these QCQP-based and descent-based methods have significant speed advantages over the SDP-based hyperkernel method, our experiments in Section V show that the hyperkernel method has better generalization performance on all the real-world data sets tested.

In this paper, we attempt to improve the hyperkernel method so that its generalization performance is as good as that of the original SDP formulation, but with a speed that is closer to the QCQP-based method of [2]. In particular, we will show that the hyperkernel method can be equivalently formulated as a second-order cone program (SOCP). SOCPs are convex optimization problems in which a linear function is minimized over the intersection of an affine linear manifold with the Cartesian product of second-order cones. Moreover, interior-point methods for SOCP have a much better worst-case complexity and run far more efficiently in practice than those for SDP problems [22], [2].

The rest of this paper is organized as follows. Section II reviews the kernel learning method using hyperkernels, and Section III gives a short introduction on SOCP. Section IV shows that the optimization problem associated with hyperkernels can be equivalently formulated as a SOCP problem, which is then followed by a number of SOCP examples in the context of various kernel methods. Experimental results on both toy and real-world data sets are presented in Section V, and the last section gives some concluding remarks. A preliminary version of this paper has appeared in [23].

In the sequel, $\mathbf{A} \succeq 0$ means that the matrix $\mathbf{A}$ is symmetric and positive semidefinite (psd), and $\mathbf{v} = [v_1, \ldots, v_k]^T \geq \mathbf{0}$ means that $v_i \geq 0$ for $i = 1, \ldots, k$. Moreover, $\mathbb{R}_+^k, \mathbb{R}_{++}^k$ denote the sets of nonnegative and positive vectors in $\mathbb{R}^k$, respectively, and $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ denotes the training set, with $\mathbf{x}_i$s in some input space $\mathcal{X}$ and the corresponding y_is in $\mathbb{R}$.

[2]However, readers are cautioned that capacity control in this method is based on a generalization bound for the transduction, but not induction, setting.

II. Learning With Hyperkernels

In this section, we review the method of hyperkernels as introduced in [1], [17]–[18]. Section II-A first introduces the concepts of reproducing kernel Hilbert space (RKHS) and regularized risk functional minimization, which are then extended to the hyper-RKHS and regularized quality functional minimization in Section II-B.

A. RKHS

Given a nonempty set $\mathcal{X}$ and a Hilbert space $\mathcal{H}_k$ of functions $f : \mathcal{X} \to \mathbb{R}$, $\mathcal{H}_k$ is a RKHS [24], [25] with kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ if

1) k has the reproducing property: $\langle f, k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}_k} = f(\mathbf{x}), \forall f \in \mathcal{H}_k, \forall \mathbf{x} \in \mathcal{X}$, where $\langle\cdot,\cdot\rangle_{\mathcal{H}_k}$ denotes the dot product in $\mathcal{H}_k$; in particular, $\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot)\rangle_{\mathcal{H}_k} = k(\mathbf{x}, \mathbf{x}'), \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$;
2) k spans $\mathcal{H}_k$.

The *regularized risk functional* is a sum of the empirical risk corresponding to a loss function $\ell : (\mathcal{X} \times \mathbb{R}^2)^m \to \mathbb{R}^+ \cup \{\infty\}$, and a regularizer $\Omega : [0, \infty) \to \mathbb{R}$ which is a strictly monotonic increasing function. Minimizing this regularized risk leads to the primal

$$\min_{f \in \mathcal{H}_k} \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \frac{\lambda}{2} \Omega(\|f\|_{\mathcal{H}_k}) \tag{1}$$

where $\|f\|_{\mathcal{H}_k}$ is the RKHS norm of f and $\lambda \in \mathbb{R}_{++}$ is a user-defined constant that trades off the empirical risk with the complexity of f. By the representer theorem, the minimizer f admits a representation of the form $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})$, where $\alpha_i \in \mathbb{R}$ for $1 \leq i \leq m$.

The number of variables in the primal problem (1) depends on the dimensionality of the kernel-induced feature space. For many nonlinear kernels, this can be very large (sometimes even infinite) and solving the primal directly is infeasible. Hence, most kernel methods solve the dual instead, in which the number of dual variables is only dependent on the size of the training set. Usually, the dual is a quadratic programming (QP) problem of the form

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \ & \boldsymbol{\alpha}^T \mathbf{v} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{G}(\mathbf{K}) \boldsymbol{\alpha} \\ \text{s.t.} \ & \mathbf{A}\boldsymbol{\alpha} + \mathbf{b} \geq \mathbf{0} \end{aligned} \tag{2}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_m]^T, \mathbf{v} \in \mathbb{R}^m, \mathbf{A} \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^n$, and $\mathbf{G}(\mathbf{K}) \in \mathbb{R}^{m \times m}$ is a matrix-valued function of the kernel matrix $\mathbf{K} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{ij} \in \mathbb{R}^{m \times m}$ and is psd.

B. Hyper-RKHS

Denote $\underline{\mathcal{X}} = \mathcal{X} \times \mathcal{X}$. Analogous to a RKHS discussed in Section II-A, the Hilbert space $\underline{\mathcal{H}}$ of functions $k : \underline{\mathcal{X}} \to \mathbb{R}$ is a hyper-RKHS if there exists a hyperkernel $\underline{k} : \underline{\mathcal{X}} \times \underline{\mathcal{X}} \to \mathbb{R}$ such that:

1) $\underline{k}$ has the reproducing property: $\langle k, \underline{k}(\underline{\mathbf{x}}, \cdot)\rangle_{\underline{\mathcal{H}}} = k(\underline{\mathbf{x}}), \forall k \in \underline{\mathcal{H}}, \forall \underline{\mathbf{x}} \in \underline{\mathcal{X}}$, where $\langle\cdot,\cdot\rangle_{\underline{\mathcal{H}}}$ denotes the dot product in $\underline{\mathcal{H}}$; in particular, $\langle \underline{k}(\underline{\mathbf{x}}, \cdot), \underline{k}(\underline{\mathbf{x}}', \cdot)\rangle_{\underline{\mathcal{H}}} = \underline{k}(\underline{\mathbf{x}}, \underline{\mathbf{x}}'), \forall \underline{\mathbf{x}}, \underline{\mathbf{x}}' \in \underline{\mathcal{X}}$;
2) $\underline{k}$ spans $\underline{\mathcal{H}}$;

3) for any fixed $\underline{\mathbf{x}} \in \underline{\mathcal{X}}$, the hyperkernel $\underline{k}$ is a valid kernel in its second argument; in other words, the function $k(\mathbf{x}, \mathbf{x}') = \underline{k}(\underline{\mathbf{x}}, (\mathbf{x}, \mathbf{x}'))$ with $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ is a kernel.

The suitability of a kernel k in a particular training data set is measured by the *regularized quality functional*, which is a sum of the regularized risk functional and the norm $\|k\|_{\underline{\mathcal{H}}}$ of k in $\underline{\mathcal{H}}$. The desired kernel function is then obtained by minimizing this functional over the entire space $\underline{\mathcal{H}}$ of kernels, leading to the primal

$$\min_{k \in \underline{\mathcal{H}}} \min_{f \in \mathcal{H}_k} \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \frac{\lambda}{2} \Omega(\|f\|_{\mathcal{H}_k}) + \frac{\lambda_Q}{2} \|k\|^2_{\underline{\mathcal{H}}} \quad (3)$$

where $\lambda_Q \in \mathbb{R}_{++}$ is another user-defined constant. By the representer theorem for hyper-RKHS, the minimizer k admits a representation of the form

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i,j=1}^{m} \beta_{ij} \underline{k}((\mathbf{x}_i, \mathbf{x}_j), (\mathbf{x}, \mathbf{x}')) \quad (4)$$

where $\beta_{ij} \in \mathbb{R}$ for $1 \leq i, j \leq m$. To ensure that k in (4) is a valid kernel, the expansion coefficients β_{ij}s are further constrained to be nonnegative. By defining $\underline{\mathbf{K}}_{ij} = [\underline{k}((\mathbf{x}_i, \mathbf{x}_j), (\mathbf{x}_k, \mathbf{x}_l)]_{kl} \in \mathbb{R}^{m \times m}$, (4) can be written in matrix form as

$$\mathbf{K} = \sum_{i,j=1}^{m} \beta_{ij} \underline{\mathbf{K}}_{ij}, \beta_{ij} \geq 0. \quad (5)$$

It is obvious that $\underline{\mathbf{K}}_{ij} \succeq 0$ from Property 3 of hyperkernels.

Combining (2) with the representer theorem for hyper-RKHS, the dual of (3) can be obtained as

$$\begin{aligned} \min_{\boldsymbol{\beta}} \max_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T \mathbf{v} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{G}(\mathbf{K}) \boldsymbol{\alpha} + \frac{\lambda_Q}{2} \boldsymbol{\beta}^T \underline{\mathbf{K}} \boldsymbol{\beta} \\ \text{s.t.} \quad & \mathbf{A}\boldsymbol{\alpha} + \mathbf{b} \geq \mathbf{0} \\ & \lambda^T \boldsymbol{\beta} = c \\ & \boldsymbol{\beta} \geq \mathbf{0} \\ & \mathbf{K} = \text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta}) \end{aligned} \quad (6)$$

where $\boldsymbol{\beta} = [\beta_{11}, \beta_{12}, \ldots, \beta_{1m}, \ldots, \beta_{m1}, \ldots, \beta_{mm}]^T, \underline{\mathbf{K}} = [\underline{k}((\mathbf{x}_i, \mathbf{x}_j), (\mathbf{x}_k, \mathbf{x}_l))]_{ijkl} \in \mathbb{R}^{m^2 \times m^2}$, and $\mathbf{K} = \text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta})$ means reshaping the $m^2 \times 1$ vector $\underline{\mathbf{K}}\boldsymbol{\beta}$ to a $m \times m$ matrix $\mathbf{K}$. Notice that an additional constraint $\lambda^T \boldsymbol{\beta} = c$, where $\lambda = [1, \ldots, 1]^T \in \mathbb{R}^{m^2}$ and c is a constant, is imposed to avoid arbitrary scaling of the resultant $\mathbf{K}$ matrix. Finally, this can then be formulated and solved as a SDP problem [17].

III. Second-Order Cone Program (SOCP)

In recent years, it has been found that many optimization problems, such as robust linear programming, robust least-squares and problems involving sums or maxima of norms, can all be formulated as SOCP [22], [26], [27]. In engineering, SOCP has also found a wide variety of applications such as filter design, antenna array weight design, and grasping force optimization in robotics. Mathematically, SOCPs are a class of convex optimization problems in which a linear function is minimized over the intersection of an affine linear manifold with the Cartesian product of *second-order cones*. The second-order cone (also known as the *quadratic cone*, *Lorentz cone*, or *ice-cream cone*) $\mathcal{Q}$ is the norm cone[3] for the Euclidean norm

$$\mathcal{Q} = \{\mathbf{x} = [x_0, \tilde{\mathbf{x}}^T]^T : x_0 \geq \|\tilde{\mathbf{x}}\|\}$$

where $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]^T \in \mathbb{R}^n, \tilde{\mathbf{x}} = [x_1, \ldots, x_{n-1}]^T \in \mathbb{R}^{n-1}$ and $\|\cdot\|$ denotes the standard Euclidean norm. In general, $\mathcal{Q}$ can be the Cartesian product of several such cones, i.e., $\mathcal{Q} = \mathcal{Q}_{n_1} \times \cdots \times \mathcal{Q}_{n_r}$ where each $\mathcal{Q}_{n_i} \subseteq \mathbb{R}^{n_i}$. The cone $\mathcal{Q}$ also induces a partial order on $\mathbb{R}^n$, as

$$\mathbf{x} \succeq_{\mathcal{Q}} \mathbf{y} \Leftrightarrow \mathbf{x} - \mathbf{y} \in \mathcal{Q}.$$

Inequalities of these forms are called *second-order cone inequalities*.

Given $\mathbf{B}_i \in \mathbb{R}^{m \times n_i}, \mathbf{d} \in \mathbb{R}^m, \mathbf{c}_i \in \mathbb{R}^{n_i}$ and $\mathbf{x}_i \in \mathbb{R}^{n_i}$, the standard primal form of a SOCP problem is

$$\begin{aligned} \min_{\mathbf{x}_1, \ldots, \mathbf{x}_r} \quad & \mathbf{c}_1^T \mathbf{x}_1 + \cdots + \mathbf{c}_r^T \mathbf{x}_r \\ \text{s.t.} \quad & \mathbf{B}_1 \mathbf{x}_1 + \cdots + \mathbf{B}_r \mathbf{x}_r = \mathbf{d} \\ & \mathbf{x}_i \succeq_{\mathcal{Q}} \mathbf{0}. \end{aligned} \quad (7)$$

The corresponding dual is

$$\begin{aligned} \max_{\mathbf{y}, \mathbf{z}_1, \ldots, \mathbf{z}_r} \quad & \mathbf{d}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{B}_i^T \mathbf{y} + \mathbf{z}_i = \mathbf{c}_i \\ & \mathbf{z}_i \succeq_{\mathcal{Q}} \mathbf{0} \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^m$ and each $\mathbf{z}_i \in \mathbb{R}^{n_i}$. Denote $\mathbf{x} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_r^T)^T, \mathbf{z} = (\mathbf{z}_1^T, \ldots, \mathbf{z}_r^T)^T$ and $\mathbf{c} = (\mathbf{c}_1^T, \ldots, \mathbf{c}_r^T)^T$. A duality theory, very similar to that for linear programs (LPs), has been developed for SOCPs. In particular, the strong duality theorem [22] guarantees that if the primal and dual problems have strictly feasible solutions (i.e., $\mathbf{x}_i \succ_{\mathcal{Q}} \mathbf{0}$ and $\mathbf{z}_i \succ_{\mathcal{Q}} \mathbf{0}, \forall i$), then both have optimal solutions (denoted $\mathbf{x}^*$ and $(\mathbf{y}^*, \mathbf{z}^*)$, respectively), and the duality gap is zero (i.e., $\mathbf{c}^T \mathbf{x}^* = \mathbf{d}^T \mathbf{y}^*$).

Moreover, it is well-known that standard optimization problems, such as LPs, convex QPs, and QCQPs, can all be solved as SOCPs. In turn, SOCP is a special case of SDP[4] [28]. While SOCPs can be solved as SDPs, it is, however, not recommended to do so [22], [26]. Interior-point methods for SOCPs (such as MOSEK [29]) have a much better worst-case complexity than solving the same problem as SDPs [20], and in practice they also run far more efficiently.

IV. SOCP Formulation

In Section IV-A, we first show that the general optimization problem in (6) can be equivalently formulated as a SOCP problem. The specific SOCP formulations for kernel learning in

[3]A set $\mathcal{K}$ is called a *cone* if, for every $\mathbf{x} \in \mathcal{K}$ and $\theta \geq 0$, we have $\theta\mathbf{x} \in \mathcal{K}$. A set is a *convex cone* if it is convex and a cone, which means that for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{K}$ and $\theta_1, \theta_2 \geq 0$, we have $\theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 \in \mathcal{K}$.

[4]SDP problems are a class of convex optimization problems in which a linear function is minimized over the intersection of an affine set and the cone of positive semidefinite matrices.

the context of various kernel methods are then discussed in Section IV-B. Finally, Section IV-C compares the worst-case time complexities of the SDP and SOCP formulations.

A. General Formulation

The Lagrangian for (2) is

$$\mathcal{L}(\boldsymbol{\alpha},\gamma)=\boldsymbol{\alpha}^T\mathbf{v}-\frac{1}{2}\boldsymbol{\alpha}^T\mathbf{G}(\mathbf{K})\boldsymbol{\alpha}+\gamma^T(\mathbf{A}\boldsymbol{\alpha}+\mathbf{b}) \quad (8)$$

where $\gamma\in\mathbb{R}^n_+$. By duality, we have $\max_{\boldsymbol{\alpha}}\min_{\gamma\geq\mathbf{0}}\mathcal{L}(\boldsymbol{\alpha},\gamma)=\min_{\gamma\geq\mathbf{0}}\max_{\boldsymbol{\alpha}}\mathcal{L}(\boldsymbol{\alpha},\gamma)$. Setting the derivative of (8) w.r.t. $\boldsymbol{\alpha}$ to zero, we obtain the optimal solution of $\boldsymbol{\alpha}$ as[5]

$$\begin{aligned}\frac{\partial\mathcal{L}(\boldsymbol{\alpha},\gamma)}{\partial\boldsymbol{\alpha}}&=\mathbf{v}-\mathbf{G}(\mathbf{K})\boldsymbol{\alpha}+\mathbf{A}^T\gamma\\ &\Rightarrow\boldsymbol{\alpha}=\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\gamma+\mathbf{v}). \quad (9)\end{aligned}$$

Substituting (9) and the Karush–Kuhn–Tucker (KKT) condition that $\gamma^T(\mathbf{A}^T\boldsymbol{\alpha}+\mathbf{b})=0$ back into (6), we then obtain

$$\begin{aligned}\min_{\boldsymbol{\beta},\gamma}\quad&\frac{1}{2}(\mathbf{A}^T\gamma+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\gamma+\mathbf{v})\\ &+\gamma^T\mathbf{b}+\frac{\lambda_Q}{2}\boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta}\\ \text{s.t.}\quad&\mathbf{1}^T\boldsymbol{\beta}=c\\ &\boldsymbol{\beta},\gamma\geq\mathbf{0}\\ &\mathbf{K}=\text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta})\\ =\min_{\boldsymbol{\beta},\gamma,t_1,t_2}\quad&\frac{1}{2}t_1+\gamma^T\mathbf{b}+\frac{\lambda_Q}{2}t_2 \quad (10)\\ \text{s.t.}\quad&t_1\geq(\mathbf{A}^T\gamma+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\gamma+\mathbf{v}) \quad (11)\\ &t_2\geq\boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta}\\ &\mathbf{1}^T\boldsymbol{\beta}=c\\ &\boldsymbol{\beta},\gamma\geq\mathbf{0}\\ &\mathbf{K}=\text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta}). \quad (12)\end{aligned}$$

Now, we utilize two techniques on converting optimization problems to SOCP problems as discussed in [22], [27]. Let $\mathbf{y}\in\mathbb{R}^n$, $\mathbf{s}=[s_1,\ldots,s_k]^T\in\mathbb{R}^k_+$, and

$$\mathbf{M}(\mathbf{s})=\sum_{i=1}^k s_i\mathbf{M}_i \quad (13)$$

where each $\mathbf{M}_i\in\mathbb{R}^{n\times n}$ is psd. The first technique shows that inequality constraints on fractional quadratic functions of the form

$$t\geq\mathbf{y}^T\mathbf{M}(\mathbf{s})^{-1}\mathbf{y} \quad (14)$$

where $t\in\mathbb{R}_+$, can be replaced by the system

$$\sum_{i=1}^k\mathbf{D}_i\mathbf{w}_i=\mathbf{y},\quad t\geq\sum_{i=1}^k t_i,\quad s_it_i\geq\mathbf{w}_i^T\mathbf{w}_i$$

where, for $i=1,\ldots,k,t_i\in\mathbb{R}_+,\mathbf{w}_i\in\mathbb{R}^{r_i},r_i=\text{rank}(\mathbf{M}_i)$ and $\mathbf{D}_i\in\mathbb{R}^{n\times r_i}$ such that $\mathbf{M}_i=\mathbf{D}_i\mathbf{D}_i^T$. As will be seen in Section IV-B, $\mathbf{G}(\mathbf{K})$ can often be written in the form of (13) in many kernel methods, i.e.,

$$\mathbf{G}(\mathbf{K})=\sum_{i,j=1}^m\beta_{ij}\mathbf{G}_{ij} \quad (15)$$

where $\beta_{ij}\in\mathbb{R}_+$ and $\mathbf{G}_{ij}\succeq 0$. We also decompose $\mathbf{G}_{ij}$ in the same form as for $\mathbf{M}_i$, i.e.,

$$\mathbf{G}_{ij}=\tilde{\mathbf{G}}_{ij}\tilde{\mathbf{G}}_{ij}^T. \quad (16)$$

This is always possible as $\mathbf{G}_{ij}\succeq 0$. The constraint

$$t_1\geq(\mathbf{A}^T\gamma+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\gamma+\mathbf{v})$$

in (11) is, thus, of the same form as (14) and can be replaced by

$$\sum_{i,j=1}^m\tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij}=\mathbf{A}^T\gamma+\mathbf{v},t_1\geq\sum_{i,j=1}^m\tau_{ij},\beta_{ij}\tau_{ij}\geq\mathbf{c}_{ij}^T\mathbf{c}_{ij} \quad (17)$$

where $\tau_{ij}\in\mathbb{R}_+,r_{ij}=\text{rank}(\mathbf{G}_{ij})$ and $\mathbf{c}_{ij}\in\mathbb{R}^{r_{ij}}$ for $i,j=1,\ldots,m$.

The second technique converts hyperbolic constraints of the form $xy\geq\mathbf{w}^T\mathbf{w}$, where $x,y\in\mathbb{R}_+,\mathbf{w}\in\mathbb{R}^n$, to the equivalent second-order cone constraint $x+y\geq\|[\begin{smallmatrix}2\mathbf{w}\\x-y\end{smallmatrix}]\|$. Now, define $\underline{\mathbf{G}}$ by

$$\underline{\mathbf{K}}=\underline{\mathbf{G}}\underline{\mathbf{G}}^T. \quad (18)$$

Then, for the constraints $\beta_{ij}\tau_{ij}\geq\mathbf{c}_{ij}^T\mathbf{c}_{ij}$ in (17) and

$$t_2\geq\boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta}=(\underline{\mathbf{G}}^T\boldsymbol{\beta})^T(\underline{\mathbf{G}}^T\boldsymbol{\beta})$$

in (12), as $\beta_{ij},\tau_{ij},t_2\in\mathbb{R}_+$, they can be rewritten as $\beta_{ij}+\tau_{ij}\geq\|[\begin{smallmatrix}2\mathbf{c}_{ij}\\\beta_{ij}-\tau_{ij}\end{smallmatrix}]\|$ and $t_2+1\geq\|[\begin{smallmatrix}2\underline{\mathbf{G}}^Tmbi\beta\\t_2-1\end{smallmatrix}]\|$, respectively.

Finally, putting these and (17) back into (10), we obtain

$$\begin{aligned}\min_{\boldsymbol{\beta},\gamma,t_1,t_2,\boldsymbol{\tau},\mathbf{c}}\quad&\frac{1}{2}t_1+\gamma^T\mathbf{b}+\frac{\lambda_Q}{2}t_2\\ \text{s.t.}\quad&\sum_{i,j=1}^m\tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij}=\mathbf{A}^T\gamma+\mathbf{v}\\ &t_1\geq\sum_{i,j=1}^m\tau_{ij} \quad (19)\\ &\beta_{ij}+\tau_{ij}\geq\left\|\begin{bmatrix}2\mathbf{c}_{ij}\\\beta_{ij}-\tau_{ij}\end{bmatrix}\right\|\\ &t_2+1\geq\left\|\begin{bmatrix}2\underline{\mathbf{G}}^T\boldsymbol{\beta}\\t_2-1\end{bmatrix}\right\|\\ &\mathbf{1}^T\boldsymbol{\beta}=c\\ &\boldsymbol{\beta},\gamma,\boldsymbol{\tau}\geq\mathbf{0} \quad (20)\end{aligned}$$

where $\boldsymbol{\tau}=[\tau_{ij}]_{i,j=1}^m,\mathbf{c}=[\mathbf{c}_{ij}]_{i,j=1}^m$, and this is a SOCP problem[6].

[5]When $\mathbf{G}(\mathbf{K})$ is only positive semidefinite, we can use the pseudoinverse $\mathbf{G}(\mathbf{K})^+$ in place of $\mathbf{G}(\mathbf{K})^{-1}$.

[6]As in linear programming, the inequality constraint $t_1\geq\sum_{i,j=1}^m\tau_{ij}$ in (19) can be reduced to standard form in (7) without undue difficulty, by simply using a slack variable z to obtain $t_1+z=\sum_{i,j=1}^m\tau_{ij}$ and the bound $z\geq 0$. However, this is not necessary here, as the SOCP solver used in the experiments can handle these inequality constraints directly.

B. Kernel Learning Examples

The hyperkernel method has been applied to a variety of kernel methods, and their SDP formulations have been discussed in [18]. In this section, we derive the corresponding SOCP formulations, which can then be solved more efficiently than their SDP counterparts.

1) C-SVM: The most popular kernel classifier is the C-SVM [5]. Its primal is

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

while the dual is

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T(\mathbf{K}\odot\mathbf{y}\mathbf{y}^T)\boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{\alpha}^T\mathbf{y} = 0 \\ & \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1} \end{aligned}$$

where $\mathbf{1} = [1,\ldots,1]^T \in \mathbb{R}^m, \mathbf{y} = [y_1,\ldots,y_m]^T$ with each $y_i \in \{-1,1\}$ being the training labels, $\odot$ denotes the Hadamard product and $C \in \mathbb{R}_+$ is a user-defined parameter. On comparison with (2), we have $\mathbf{v} = \mathbf{1}$ and

$$\begin{aligned} \mathbf{G}(\mathbf{K}) = \mathbf{K}\odot\mathbf{y}\mathbf{y}^T &= \left(\sum_{i,j=1}^m \beta_{ij}\underline{\mathbf{K}}_{ij}\right)\odot\mathbf{y}\mathbf{y}^T \\ &= \sum_{i,j=1}^m \beta_{ij}(\underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}^T) \end{aligned}$$

on using (5) and the distributive property of the Hadamard product. This confirms that $\mathbf{G}(\mathbf{K})$ is of the form in (15), with $\mathbf{G}_{ij} = \underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}^T$. It is also easy to see[7] that $\mathbf{G}_{ij}\succeq 0$. Moreover, $\mathbf{A} = [\mathbf{I}_m, -\mathbf{I}_m, -\mathbf{y}, \mathbf{y}]^T$ and $\mathbf{b} = [\mathbf{0}^T, C\mathbf{1}^T, 0, 0]^T$, where $\mathbf{I}_m$ is the $m\times m$ identity matrix and $\mathbf{0} = [0,\ldots,0]^T \in \mathbb{R}^m$. Define $\tilde{\mathbf{G}}_{ij}$ and $\underline{\mathbf{G}}$ by (16) and (18), respectively, and $\gamma = [\boldsymbol{\eta}, \boldsymbol{\xi}, b_1, b_2]^T$. (20) then becomes

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\eta},\boldsymbol{\xi},b,t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \quad & \frac{1}{2}t_1 + C\boldsymbol{\xi}^T\mathbf{1} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \sum_{i,j=1}^m \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = -b\mathbf{y} + \boldsymbol{\eta} - \boldsymbol{\xi} + \mathbf{1} \\ & t_1 \geq \sum_{i,j=1}^m \tau_{ij} \\ & \beta_{ij} + \tau_{ij} \geq \left\|\begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix}\right\| \\ & t_2 + 1 \geq \left\|\begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix}\right\| \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \boldsymbol{\eta}, \boldsymbol{\xi}, \boldsymbol{\beta}, b_1, b_2, \boldsymbol{\tau} \geq \mathbf{0} \end{aligned}$$

where $b = b_1 - b_2$ and $\boldsymbol{\xi}$ can be shown to be equal to the bias and primal slack variables, respectively, in the C-SVM.

2) ν-SVM, Lagrangian SVM, One-Class SVM and ν-Support Vector Regression: Hyperkernel methods for ν-SVM [30], Lagrangian SVM [31], one-class SVM [31] and ν-support vector regression (ν-SVR) [30] can be analogously formulated as SOCP problems. For simplicity of presentation, here we only list out their corresponding SOCP formulations. ν-SVM

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\eta},\boldsymbol{\xi},\rho,b,t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \quad & \frac{1}{2}t_1 - \rho + \frac{1}{\nu m}\boldsymbol{\xi}^T\mathbf{1} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \sum_{i,j=1}^m \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = -b\mathbf{y} + \boldsymbol{\eta} - \boldsymbol{\xi} + \rho\mathbf{1} \\ & t_1 \geq \sum_{i,j=1}^m \tau_{ij} \\ & \beta_{ij} + \tau_{ij} \geq \left\|\begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix}\right\| \\ & t_2 + 1 \geq \left\|\begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix}\right\| \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \rho, \boldsymbol{\eta}, \boldsymbol{\xi}, \boldsymbol{\beta}, b_1, b_2, \boldsymbol{\tau} \geq \mathbf{0} \end{aligned}$$

where $\tilde{\mathbf{G}}_{ij}$ and $\underline{\mathbf{G}}$ are as defined for the C-SVM.

Lagrangian SVM

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\eta},\boldsymbol{\xi},b,t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \quad & \frac{1}{2}t_1 + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \sum_{i,j=1}^m \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} + b\mathbf{y} + \boldsymbol{\xi} = \boldsymbol{\eta} + \mathbf{1} \\ & t_1 \geq \sum_{i,j=1}^m \tau_{ij} + \tau_1 + \tau_2 \\ & \beta_{ij} + \tau_{ij} \geq \left\|\begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix}\right\| \\ & 1 + \tau_1 \geq \left\|\begin{bmatrix} 2\mathbf{c}_1 \\ 1 - \tau_1 \end{bmatrix}\right\| \\ & \frac{m}{C} + \tau_2 \geq \left\|\begin{bmatrix} 2\mathbf{c}_2 \\ \frac{m}{C} - \tau_2 \end{bmatrix}\right\| \\ & t_2 + 1 \geq \left\|\begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix}\right\| \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \boldsymbol{\eta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\tau} \geq \mathbf{0} \end{aligned}$$

where $\tilde{\mathbf{G}}_{ij}$ is given by $\tilde{\mathbf{G}}_{ij}\tilde{\mathbf{G}}_{ij}^T = \underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}^T, \underline{\mathbf{G}}$ by (18) and $\boldsymbol{\tau} = [\tau_{11},\ldots,\tau_{mm},\tau_1,\tau_2]^T$.

One-class SVM

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\eta},\boldsymbol{\xi},\rho,t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \quad & \frac{1}{2}t_1 - \rho + \frac{1}{\nu m}\boldsymbol{\xi}^T\mathbf{1} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \sum_{i,j=1}^m \mathbf{G}_{ij}\mathbf{c}_{ij} = \boldsymbol{\eta} - \boldsymbol{\xi} + \rho\mathbf{1} \\ & t_1 \geq \sum_{i,j=1}^m \tau_{ij} \\ & \beta_{ij} + \tau_{ij} \geq \left\|\begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix}\right\| \\ & t_2 + 1 \geq \left\|\begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix}\right\| \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \boldsymbol{\eta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \rho_1, \rho_2, \quad \boldsymbol{\tau} \geq \mathbf{0} \end{aligned}$$

[7]For an arbitrary $\mathbf{z} = [z_1,\ldots,z_m]^T \in \mathbb{R}^m, \mathbf{z}^T(\underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}^T)\mathbf{z} = \sum_{k,l=1}^m z_k z_l[\underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}^T]_{kl} = \sum_{k,l=1}^m (y_k z_k)(y_l z_l)\underline{k}((\mathbf{x}_i,\mathbf{x}_j),(\mathbf{x}_k,\mathbf{x}_l)) \geq 0$, as $\underline{k}((\mathbf{x}_i,\mathbf{x}_j),(\cdot,\cdot))$ is a kernel. Thus, $\underline{\mathbf{K}}_{ij}\odot\mathbf{y}\mathbf{y}'\succeq 0$.

where $\tilde{\mathbf{G}}_{ij}$ and $\underline{\mathbf{G}}$ are defined by (16) and (18), respectively, and $\rho = \rho_1 - \rho_2$ is the margin.

ν-SVR

$$\begin{aligned}\min \quad & \frac{1}{2}t_1 + C\nu\epsilon + \frac{C}{m}(\boldsymbol{\xi}+\boldsymbol{\xi}^*)^T\mathbf{1} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \sum_{i,j=1}^{m} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = \mathbf{y} - b\mathbf{1} - \epsilon\mathbf{1} - \boldsymbol{\xi} + \boldsymbol{\eta} \\ & 0 = -2\epsilon\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{\xi}^* + \boldsymbol{\eta} + \boldsymbol{\eta}^* \\ & t_1 \geq \sum_{i,j=1}^{m} \tau_{ij} \\ & \beta_{ij} + \tau_{ij} \geq \left\| \begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix} \right\| \\ & t_2 + 1 \geq \left\| \begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix} \right\| \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \epsilon, \boldsymbol{\eta}, \boldsymbol{\eta}^*, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\beta}, b_1, b_2, \quad \boldsymbol{\tau} \geq \mathbf{0}\end{aligned}$$

w.r.t. $\boldsymbol{\beta}, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*, \epsilon, b, t_1, t_2, \boldsymbol{\tau}, \mathbf{c}$, where $\underline{\mathbf{K}}_{ij} = \tilde{\mathbf{G}}_{ij}\tilde{\mathbf{G}}_{ij}^T$, $\underline{\mathbf{G}}$ is defined by (18) and $b = b_2 - b_1$.

3) Kernel Target Alignment: The kernel target alignment [6] measures the similarity between the kernel matrix and the training labels. It is defined as

$$A(\mathbf{K}, \mathbf{y}\mathbf{y}^T) = \frac{\langle \mathbf{K}, \mathbf{y}\mathbf{y}^T \rangle}{m\sqrt{\langle \mathbf{K}, \mathbf{K} \rangle}} \tag{21}$$

where $\langle \mathbf{A}, \mathbf{B} \rangle = \operatorname{tr}(\mathbf{A}^T\mathbf{B})$ denotes the Frobenius product of matrices $\mathbf{A}$ and $\mathbf{B}$. The alignment is not directly based on the generalized risk functional and so the approach in Section IV does not apply. However, we can still maximize (21) by maximizing $\langle \mathbf{K}, \mathbf{y}\mathbf{y}^T \rangle$ and minimizing $\langle \mathbf{K}, \mathbf{K} \rangle$ together, yielding the optimization problem

$$\min_{\mathbf{K} \succeq 0} -\langle \mathbf{K}, \mathbf{y}\mathbf{y}^T \rangle + C\langle \mathbf{K}, \mathbf{K} \rangle$$

where C is a user-defined parameter. Substituting in (5) and adding the hyperkernel regularizer $(\lambda_Q/2)\boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta}$, we obtain

$$\begin{aligned}\min_{\boldsymbol{\beta}, t_1, t_2} \quad & -\sum_{i,j=1}^{m} \beta_{ij}\operatorname{tr}(\underline{\mathbf{K}}_{ij}\mathbf{y}\mathbf{y}^T) + Ct_1 + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & t_1 \geq \sum_{i,j,k,l=1}^{m} \beta_{ij}\beta_{kl}\operatorname{tr}(\underline{\mathbf{K}}_{ij}\underline{\mathbf{K}}_{kl}) \\ & t_2 \geq \boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta} \\ & \mathbf{1}^T\boldsymbol{\beta} = c \\ & \boldsymbol{\beta} \geq \mathbf{0}\end{aligned}$$

which is a QCQP. As mentioned in Section III, QCQP can also be solved as SOCP, though it is often more convenient to solve it directly.

TABLE I
DATA SETS USED IN THE EXPERIMENTS. THE FIRST EIGHT ARE CLASSIFICATION PROBLEMS WHILE THE LAST FOUR ARE REGRESSION PROBLEMS

date set	# training patterns	# testing patterns	# attributes
toy	100	1000	2
colon	38	24	2000
heart	162	108	13
ionosphere	211	140	33
liver	207	138	6
lymphoma	58	38	4096
pima	461	307	8
sonar	125	83	60
boston	304	202	13
imports	96	63	15
computer	126	83	6
mpg	236	156	7

C. Worst-Case Time Complexities

In this section, we compare the worst-case time complexities of the SDP and SOCP formulations. In the sequel, the following notations are used: $\mathbf{x} \in \mathbb{R}^N, \mathbf{f} \in \mathbb{R}^N, \mathbf{A}_i \in \mathbb{R}^{(N_i-1)\times N}, \mathbf{b}_i \in \mathbb{R}^{N_i-1}, \mathbf{c}_i \in \mathbb{R}^N$ and $\mathbf{d}_i \in \mathbb{R}$.

From [26], it is known that for a SDP problem of the form

$$\begin{aligned}\min_{\mathbf{x}} \quad & \mathbf{f}^T\mathbf{x} \\ \text{s.t.} \quad & \begin{bmatrix} (\mathbf{c}_i^T\mathbf{x} + \mathbf{d}_i)\mathbf{I} & \mathbf{A}_i\mathbf{x} + \mathbf{b}_i \\ (\mathbf{A}_i\mathbf{x} + \mathbf{b}_i)^T & \mathbf{c}_i^T\mathbf{x} + \mathbf{d}_i \end{bmatrix} \succeq 0, \\ & i = 1, \ldots, N\end{aligned} \tag{22}$$

its time complexity for each iteration is $O(N^2\sum_i N_i^2)$. Now, as mentioned in Section II-B, the kernel learning optimization problem in (6) is a SDP [17], which can be written as

$$\begin{aligned}\min_{\boldsymbol{\beta}, \boldsymbol{\gamma}, t_1, t_2} \quad & \frac{1}{2}t_1 + \gamma^T\mathbf{b} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{G}(\mathbf{K}) & (\mathbf{A}^T\gamma + \mathbf{v}) \\ (\mathbf{A}^T\gamma + \mathbf{v})^T & t_1 \end{bmatrix} \succeq 0 \\ & \begin{bmatrix} \mathbf{I} & \underline{\mathbf{G}}^T\boldsymbol{\beta} \\ (\underline{\mathbf{G}}^T\boldsymbol{\beta})^T & t_2 \end{bmatrix} \succeq 0 \\ & \mathbf{1}^T\boldsymbol{\beta} - c \geq 0 \\ & c - \mathbf{1}^T\boldsymbol{\beta} \geq 0 \\ & \boldsymbol{\beta}, \gamma \geq \mathbf{0}.\end{aligned}$$

Comparing this with the form in (22), we have $N = O(m^2)$, $\sum_i N_i^2 = O(m^4)$ and the time complexity per iteration is, thus, $O(m^8)$. Using the primal-dual method for solving this SDP, the accuracy of a given solution can be improved by an absolute constant factor in $O((m^2)^{1/2}) = O(m)$ iterations [27]. Hence, the total complexity is $O(m^8 \times m) = O(m^9)$.

On the other hand, for a SOCP of the form

$$\begin{aligned}\min_{\mathbf{x}} \quad & \mathbf{f}^T\mathbf{x} \\ \text{s.t.} \quad & \|\mathbf{A}_i\mathbf{x} + \mathbf{b}_i\| \leq \mathbf{c}_i^T\mathbf{x} + \mathbf{d}_i, \quad i = 1, \ldots, N\end{aligned} \tag{23}$$

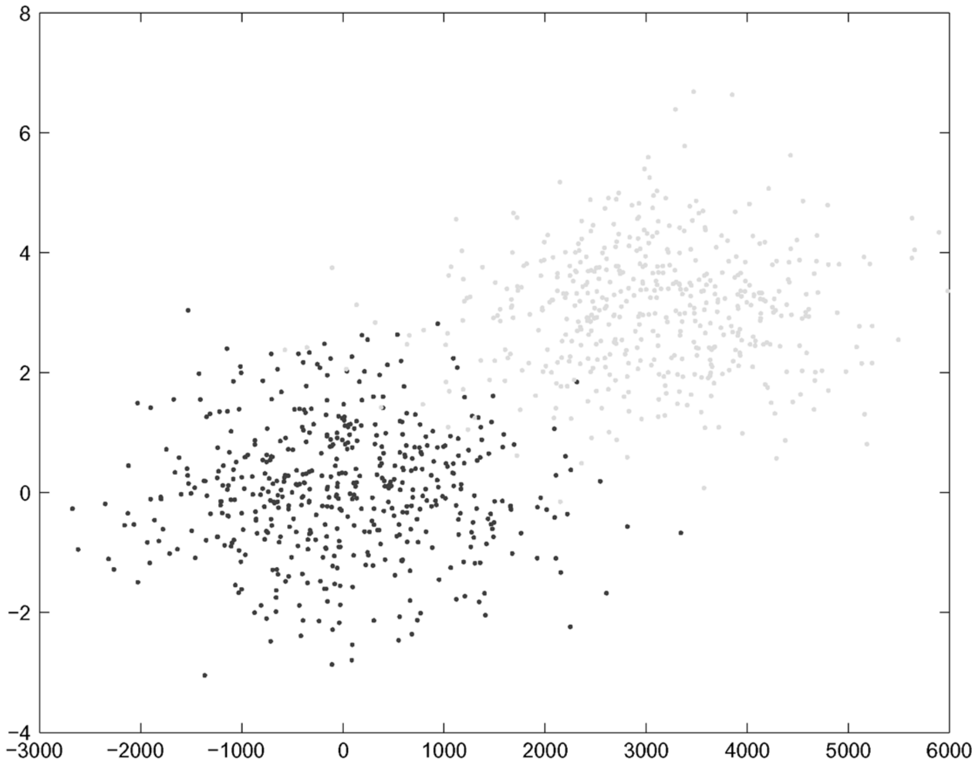

Fig. 1. Data distribution of the toy data set.

its time complexity is only $O(N^2 \sum_i N_i)$ [26]. Now, the SOCP formulation in (20) can be written as

$$\begin{aligned}
\min \quad & \frac{1}{2}t_1 + \gamma^T\mathbf{b} + \frac{\lambda_Q}{2}t_2 + \tilde{C}\mathbf{u}'\mathbf{1} \\
\text{s.t.} \quad & \sum_{i,j=1}^{m} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} - \mathbf{A}^T\gamma - \mathbf{v} + \mathbf{u} \geq \mathbf{0} \\
& -\sum_{i,j=1}^{m} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} + \mathbf{A}^T\gamma + \mathbf{v} \geq \mathbf{0} \\
& \mathbf{u} \geq \mathbf{0} \\
& t_1 \geq \sum_{i,j=1}^{m} \tau_{ij} \\
& \beta_{ij} + \tau_{ij} \geq \left\| \begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix} \right\|, \quad i,j = 1,\ldots,m \\
& t_2 + 1 \geq \left\| \begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix} \right\| \\
& \mathbf{1}^T\boldsymbol{\beta} = c \\
& \boldsymbol{\beta}, \gamma, \boldsymbol{\tau} \geq \mathbf{0}
\end{aligned}$$

w.r.t. $\boldsymbol{\beta}, \gamma, t_1, t_2, \boldsymbol{\tau}, \mathbf{c}, \mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^m$ is a slack variable, and $\tilde{C}$ is a very large constant used to guarantee that the value of $\mathbf{u}$ goes to zero. Comparing with (23), we have $N = O(m^2)$ and $\sum_k N_k = \sum_{i,j=1}^m r_{ij} = O(m^3)$, where $r_{ij} = \mathrm{rank}(\mathbf{G}_{ij}) \leq m$. Thus, the time complexity per iteration is $O(m^7)$. Using the primal-dual method for solving this SOCP, the accuracy of a given solution can be improved by an absolute constant factor in $O(m^{3/2})$ iterations [27]. Hence, the total complexity is $O(m^7 \times m^{1.5}) = O(m^{8.5})$, which is smaller than that of the SDP formulation. Note that while these worst-case time complexities may appear huge, their empirical time complexities are usually much smaller, as will be experimentally demonstrated in Section V.

TABLE II
TEST SET ACCURACIES (%) ON THE TOY DATA SET

# training samples	SDP	SOCP	QCQP
10	84.60 ± 10.60	84.60 ± 10.60	84.55 ± 12.41
20	95.83 ± 1.70	95.83 ± 1.70	94.99 ± 3.50
30	96.15 ± 1.45	96.15 ± 1.45	95.88 ± 3.93
40	97.02 ± 0.71	97.02 ± 0.71	97.21 ± 0.63
50	97.39 ± 0.51	97.39 ± 0.51	97.34 ± 0.71
60	-	97.35 ± 0.49	97.33 ± 0.61
70	-	97.49 ± 0.36	97.47 ± 0.38
80	-	97.34 ± 0.49	-
90	-	97.51 ± 0.36	-
100	-	97.36 ± 0.41	-

V. EXPERIMENTS

In this section, we demonstrate the speed-up that can be obtained by replacing the SDP formulation in [17] with our SOCP formulation. For illustration purposes, classification experiments using the C-SVM (Section IV-B.1) and regression experiments using the ν-SVR (Section IV-B.2) are discussed in Sections V-A and V-B, respectively. Table I summarizes the characteristics of the data sets used. Note that the induction, rather than transduction, setting will be employed here. Moreover, as in [17], we use the automatic relevance determination (ARD) hyperkernel

$$\underline{k}((\mathbf{x}, \mathbf{x}'), (\mathbf{x}'', \mathbf{x}''')) = \prod_{j=1}^{d} \frac{1 - \lambda_h}{1 - \lambda_h \exp\left(-\sigma_j^{-1}((x_j - x_j')^2 + (x_j'' - x_j''')^2)\right)}$$

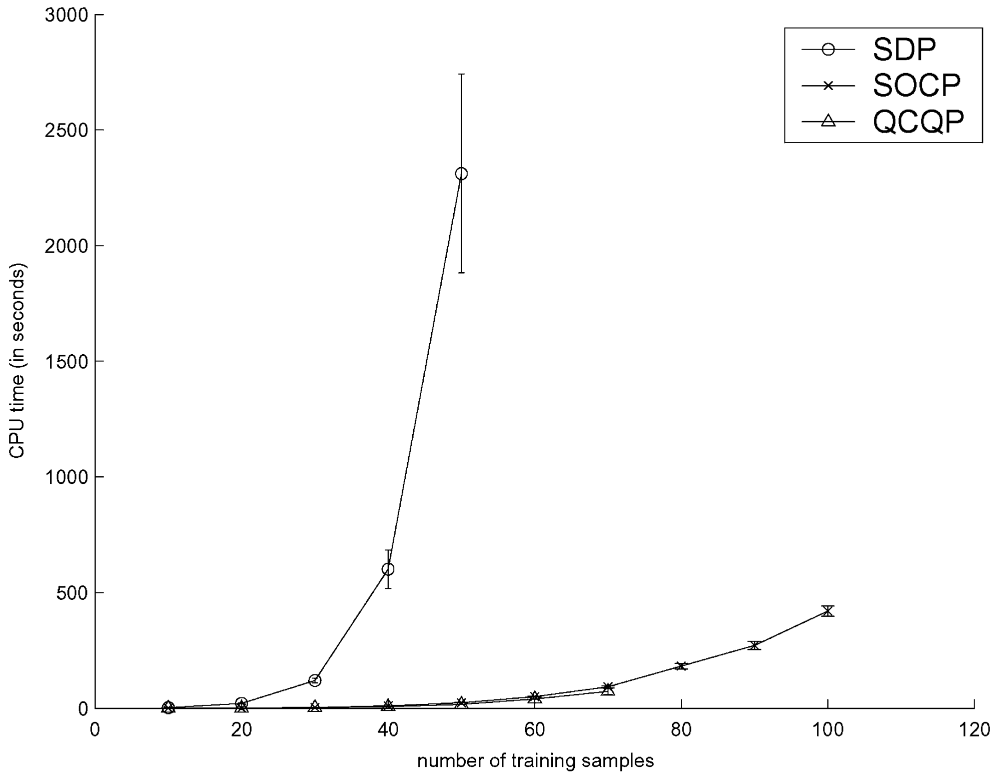

Fig. 2. CPU time required on the toy data set (note that the two curves for SOCP and QCQP have almost overlapped). The QCQP implementation takes more storage and has to stop at a training set size of 70. This may be alleviated by reformulating the QCQP as a SOCP.

where x_js are the components of $\mathbf{x} \in \mathbb{R}^d$ (and similarly for $x'_j, x_j{}''$ and x_j'''), and $\lambda_h, \sigma_1, \ldots, \sigma_d$ are user-defined parameters with the same setting as in [17]. For comparison, we also perform [2] under this setting, by *assuming* that the candidate kernel matrix is of the form in (5). Also, as in [2], we fix its trace at $\sum_{i,j=1}^m \mathrm{trace}(\underline{\mathbf{K}}_{ij})$. The mixing coefficients β_{ij}s, together with the SVM (or SVR), are then to be learned. It is shown in [2] that the resultant optimization problem is a QCQP. We use SDPT3 (version 3.02)[8] [34] as the SDP solver, and MOSEK (version 3)[9] for solving SOCP and QCQP. All implementations are in MATLAB, and the experiments are performed on a 2.4-GHz Pentium-4 machine, with 1-GB memory, and running Windows XP.

A. Classification

1) Toy Data Set: The first experiment is performed on the two-class data used in [17]. It is generated from two Gaussian distributions, one centered at (0,0) and the other at (3 3000), with highly nonisotropic variance (the standard deviation is 1 in one dimension and 1000 in the other) (Fig. 1). In order to better demonstrate the computational requirements of the different formulations, low rank approximation on the hyperkernel matrix is not used here. The number of training samples is varied from 10 to 100, and an independent set of 1000 samples are used for testing. To reduce statistical variability, results here are based on averages over 30 random repetitions.

Table II compares the test set accuracies obtained. As both the SDP and SOCP formulations are derived from the same optimization problem, they yield identical kernel functions and identical C-SVMs. On the other hand, their speeds are vastly different. As can be seen from Fig. 2, our SOCP formulation is about 100 times faster than that of SDP. Indeed, the SDP formulation is so slow that we have to stop when the training set size reaches 50. Notice that the QCQP-based method yields comparable generalization performance on this simple toy problem. Moreover, it is also as fast as our SOCP formulation. Empirically, the time complexities we obtained for SDP, SOCP, and QCQP are $O(m^{4.27})$, $O(m^{3.38})$ and $O(m^{3.11})$, respectively.

2) Real-World Data Sets: The second set of experiments are performed on seven real-world classification data sets:[10] colon cancer (`colon`), heart disease (`heart`), ionosphere (`ionosphere`), liver disorders (`liver`), lymphoma (`lymphoma`), pima indians diabetes (`pima`) and sonar (`sonar`). We use 60% of the data for training and the remaining 40% for testing. Results here are based on averages over 100 random repetitions. Moreover, we perform low rank approximation on the hyperkernel matrix as mentioned in [1] and [18]. We observe that the resultant matrix ranks obtained are in the range of 10–20.

Results are shown in Table III. As can be seen, the test set accuracies obtained by the SDP and SOCP formulations are identical, but significant speedup can be achieved with the use of SOCP. Moreover, in terms of the test set accuracy, both the SDP and SOCP formulations are better than the QCQP-based method on all data sets tested. Using the one-tailed paired t test, this dif-

[8]SDPT3 can be downloaded from http://www.math.nus.edu.sg/~mattohkc/sdpt3.html. A recent benchmark study [33] shows that SDPT3 is comparable with SeDuMi (used in [17]) on small SDP problems, but is more capable of solving larger SDP problems.

[9]MOSEK can be downloaded from http://www.mosek.com.

[10]Data sets `heart`, `ionosphere`, `liver`, `pima`, and `sonar` are from the UCI machine learning repository [35], while `colon` and `lymphoma` can be downloaded from http://www.kyb.tuebingen.mpg.de/bs/people/weston/l0.

TABLE III
TEST SET ACCURACIES AND CPU TIME ON THE REAL-WORLD CLASSIFICATION DATA SETS

	data set	SDP	SOCP	QCQP
accuracy (in %)	colon	84.08 ± 5.81	84.08 ± 5.81	83.04 ± 6.39
	heart	83.80 ± 2.80	83.80 ± 2.80	82.43 ± 2.82
	ionosphere	93.24 ± 1.55	93.24 ± 1.55	84.56 ± 3.76
	liver	64.80 ± 4.12	64.80 ± 4.12	64.55 ± 4.22
	lymphoma	90.24 ± 4.90	90.24 ± 4.90	82.30 ± 6.39
	pima	77.25 ± 1.71	77.25 ± 1.71	75.44 ± 1.69
	sonar	79.18 ± 4.41	79.18 ± 4.41	74.20 ± 3.94
CPU time (in sec)	colon	1.15 ± 0.11	0.28 ± 0.17	0.22 ± 0.13
	heart	17.25 ± 0.99	2.39 ± 0.18	0.71 ± 0.11
	ionosphere	27.50 ± 2.94	4.75 ± 0.26	1.41 ± 0.10
	liver	21.05 ± 0.82	3.17 ± 0.25	1.42 ± 0.17
	lymphoma	2.13 ± 0.20	0.39 ± 0.02	0.22 ± 0.04
	pima	224.54 ± 9.04	45.99 ± 2.21	9.58 ± 0.30
	sonar	8.91 ± 0.39	1.60 ± 0.15	0.53 ± 0.14

ference is statistically significant at a 0.025 level of significance. In terms of speed, our SOCP formulation is very competitive with the QCQP-based method, with a running time of about 1.3 to 4.8 times that of the QCQP-based method on average.

B. Regression

While Lanckriet *et al.*'s method [2] only addresses the kernel learning problem in classification problems, the hyperkernel method can be naturally applied to a variety of scenarios (Section IV-B). In this section, we demonstrate one such example, namely, the ν-SVR in Section IV-B.2. For comparison, we also extend the classification formulation in [2] by replacing the margin criterion (and the corresponding constraints) in the SVM by the objective function of ν-SVR, i.e.,

$$\begin{aligned}\min_{\mathbf{K}}\max_{\boldsymbol{\lambda},\boldsymbol{\lambda}^*} \quad & (\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{y} - \frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{K}(\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)\\ \text{s.t.} \quad & (\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{1} = 0\\ & (\boldsymbol{\lambda}+\boldsymbol{\lambda}^*)^T\mathbf{1} \le C\nu\\ & \mathbf{0} \le \boldsymbol{\lambda}, \boldsymbol{\lambda}^* \le \frac{C}{m}\mathbf{1}.\\ & \text{trace}(\mathbf{K}) = \kappa\\ & \mathbf{K} = \sum_{i,j=1}^{m} \beta_{ij}\mathbf{K}_{ij}\\ & \beta_{ij} \ge 0\end{aligned}$$

where κ is a user-defined constant[11]. Following a similar derivation as in [2], it can be shown that the resultant optimization problem is still a QCQP

$$\begin{aligned}\max_{\boldsymbol{\lambda},\boldsymbol{\lambda}^*,t} \quad & (\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{y} - t\\ \text{s.t.} \quad & \frac{\text{tr}(\mathbf{K}_{ij})}{\kappa}t \ge \frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{K}_{ij}(\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)\\ & (\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)^T\mathbf{1} = 0\\ & (\boldsymbol{\lambda}+\boldsymbol{\lambda}^*)^T\mathbf{1} \le C\nu\\ & \mathbf{0} \le \boldsymbol{\lambda}, \boldsymbol{\lambda}^* \le \frac{C}{m}\mathbf{1}.\end{aligned}$$

[11]Following [2], we set $\kappa = \sum_{i,j=1}^{m} \mathrm{trace}(\mathbf{K}_{ij})$ in the experiment.

TABLE IV
CPU TIME (IN SECONDS) ON THE REGRESSION DATA SETS

data set	SDP	SOCP	QCQP
boston	363.09 ± 37.23	13.77 ± 1.15	22.41 ± 4.42
imports	30.96 ± 2.25	0.75 ± 0.12	1.12 ± 0.07
computer	39.54 ± 3.27	0.86 ± 0.14	3.66 ± 1.35
mpg	181.16 ± 12.87	4.60 ± 0.25	11.26 ± 0.91

Experiments are performed on four real-world regression data sets from the UCI machine learning repository [35]: Boston housing (`boston`), auto imports (`imports`), computer hardware (`computer`) and auto mpg (`mpg`). The experimental setup is the same as that for classification problems, with low rank approximation performed on the hyperkernel matrix. Denote the n test patterns by $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and the resultant function obtained by ν-SVR (with kernel learning) by f. The following three error criteria are used for performance comparison:

1) root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2}$$

2) mean absolute error (MAE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - f(\mathbf{x}_i)|$$

3) mean relative error (MRE):

$$\text{MRE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - f(\mathbf{x}_i)}{y_i}\right|.$$

Results based on averages over 100 random repetitions are shown in Tables IV and V. As can be seen, the SDP and SOCP formulations yield almost identical test errors, with minor differences due to the use of different optimization solvers. In terms of all three error measures, both the SDP and SOCP formulations are better than the QCQP-based method, and the differences are statistically significant at a 0.025 level of significance. Moreover, as can be seen from Table V, our SOCP

TABLE V
PREDICTION ERRORS ON THE REGRESSION DATA SETS

data set	error	SDP	SOCP	QCQP
boston	RMSE	3.45 ± 0.50	3.45 ± 0.50	4.36 ± 0.56
	MAE	2.29 ± 0.19	2.29 ± 0.19	2.73 ± 0.22
	MRE	0.12 ± 0.01	0.12 ± 0.01	0.13 ± 0.01
imports	RMSE	2571.84 ± 525.37	2544.08 ± 524.63	5890.30 ± 520.80
	MAE	1623.35 ± 219.02	1614.48 ± 217.50	4850.05 ± 307.14
	MRE	0.13 ± 0.01	0.13 ± 0.01	0.49 ± 0.04
computer	RMSE	57.39 ± 39.36	56.22 ± 39.30	126.72 ± 32.15
	MAE	12.30 ± 7.27	11.92 ± 7.21	62.06 ± 7.80
	MRE	0.09 ± 0.07	0.08 ± 0.06	1.06 ± 0.27
mpg	RMSE	2.78 ± 0.22	2.73 ± 0.21	2.88 ± 0.21
	MAE	2.02 ± 0.09	1.98 ± 0.09	2.04 ± 0.13
	MRE	0.09 ± 0.00	0.09 ± 0.00	0.09 ± 0.01

formulation is again much faster than the traditional SDP formulation for hyperkernels, and is even faster than the QCQP-based method under this regression setting.

VI. CONCLUSION

In this paper, we show that the kernel (function) learning method using hyperkernels can be equivalently formulated as a SOCP problem, which can be solved more efficiently than the traditional SDP formulation. Experimental results on both classification and regression problems, with toy and real-world data sets, exhibit significant speedups. We also demonstrate that the hyperkernel method yields better generalization performance than the kernel matrix learning method of [2]. The combination of the proposed SOCP formulation with low rank approximation on the hyperkernel matrix will, thus, enable the method of hyperkernels to be efficiently applied even on large data sets, with good generalization performance.

In the experiments, we only used a straightforward implementation for the SOCP formulations. In the future, we will further exploit potential structure and sparsity in our SOCP formulation, which have often provided substantial speedups in many convex optimization problems. Moreover, Bach *et al.* [36] recently proposed a speedup technique for their kernel learning method in [2] with the use of sequential minimal optimization (SMO) [37]. Although this kernel learning method yielded inferior performance in our experiments, the possible integration of this SMO technique into our SOCP formulation will also be studied in the future. Moreover, the two techniques that we have used for reducing optimization problems to SOCP problems are very general and we will explore similar reductions for some recently introduced SDP-based methods in the kernel literature (such as [38]). Finally, note that the hyperkernel method can also be easily extended for transduction problems, and its comparison with the transductive method in [2] will be further explored.

REFERENCES

[1] C. S. Ong, A. J. Smola, and R. C. Williamson, "Hyperkernels," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003.

[2] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *J. Mach. Learn. Res.*, vol. 5, pp. 27–72, 2004.

[3] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

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

[5] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[6] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola, "On kernel-target alignment," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002.

[7] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Mach. Learn.*, vol. 46, no. 1–3, pp. 131–159, 2002.

[8] J. T. Kwok, "The evidence framework applied to support vector machines," *IEEE Trans. Neural Netw.*, vol. 11, no. 5, pp. 1162–1173, Oct. 2000.

[9] P. Sollich, "Bayesian methods for support vector machines: Evidence and predictive class probabilities," *Mach. Learn.*, vol. 46, no. 1/3, pp. 21–52, 2002.

[10] M. M. S. Lee, S. S. Keerthi, C. J. Ong, and D. DeCoste, "An efficient method for computing leave-one-out error in support vector machines with Gaussian kernels," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 750–757, May 2004.

[11] H. Xiong, M. N. S. Swamy, and M. O. Ahmad, "Optimizing the kernel in the empirical feature space," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 460–474, Mar. 2005.

[12] K. P. Bennett, M. Momma, and M. J. Embrechts, "MARK: A boosting algorithm for heterogeneous kernel models," in *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Edmonton, AB, Canada, Jul. 2002, pp. 24–31.

[13] K. Crammer, J. Keshet, and Y. Singer, "Kernel design using boosting," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003.

[14] G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semi-definite programming," in *Proc. 19th Int. Conf. Machine Learning*, Sydney, Australia, 2002, pp. 323–330.

[15] K. Tsuda, S. Uda, T. Kin, and K. Asai, "Minimizing the cross validation error to mix kernel matrices of heterogeneous biological data," *Neural Process. Lett.*, vol. 19, pp. 63–72, 2004.

[16] Z. Zhang, D.-Y. Yeung, and J. T. Kwok, "Bayesian inference for transductive learning of kernel matrix using the Tanner-Wong data augmentation algorithm," in *Proc. 21st Int. Conf. Machine Learning*, Banff, AB, Canada, Jul. 2004, pp. 935–942.

[17] C. S. Ong and A. J. Smola, "Machine learning with hyperkernels," in *Proc. 20th Int. Conf. Machine Learning*, Washington, DC, 2003, pp. 568–575.

[18] C. S. Ong, A. J. Smola, and R. C. Williamson, "Learning the kernel with hyperkernels," *J. Mach. Learn. Res.*, to be published.

[19] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," *J. Mach. Learn. Res.*, vol. 2, pp. 243–264, Dec. 2001.

[20] T. Zhang, "Some sparse approximation bounds for regression problems," in *Proc. 18th Int. Conf. Machine Learning*, 2001, pp. 624–631.

[21] O. Bousquet and D. J. L. Herrmann, "On the complexity of learning the kernel matrix," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003.

[22] F. Alizadeh and D. Goldfarb, "Second-order cone programming," *Math. Programm. Ser. B*, vol. 95, pp. 3–51, 2003.

[23] I. W. Tsang and J. T. Kwok, "Efficient hyperkernel learning using second-order cone programming," in *Proc. 15th Eur. Conf. Machine Learning*, Pisa, Italy, Sep. 2004, pp. 453–464.

[24] G. Wahba, "Spline models for observational data," presented at the Number 59 in CBMS—NSF Regional Conf. Ser. Applied Mathematics, Philadelphia, PA, 1990.

[25] ——, "Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV," in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 69–88.

[26] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra Appl.*, vol. 284, pp. 193–228, 1998.

[27] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming* Philadelphia, PA, 1994. SIAM.

[28] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Rev.*, vol. 38, no. 1, pp. 49–95, 1996.

[29] E. D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Math. Programm.*, vol. 95, no. 2, pp. 249–277, 2003.

[30] B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Comput.*, vol. 12, no. 4, pp. 1207–1245, 2000.

[31] O. L. Mangasarian and D. R. Musicant, "Lagrangian support vector machines," *J. Mach. Learn. Res.*, vol. 1, pp. 161–177, 2001.

[32] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001.

[33] H. D. Mittelmann, "An independent benchmarking of SDP and SOCP solvers," *Math. Programm. Ser. B*, vol. 95, pp. 407–430, 2003.

[34] K. C. Toh, M. J. Todd, and R. H. Tutuncu, "SDPT3—A matlab software package for semidefinite programming," *Optim. Meth. Software*, vol. 11, pp. 545–581, 1999.

[35] C. Blake, E. Keogh, and C. J. Merz. (1998) *UCI Repository of Machine Learning Databases* [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[36] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the SMO algorithm," presented at the 21st Int. Conf. Machine Learning, Banff, AB, Canada, Jul. 2004.

[37] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.

[38] T. D. Bie and N. Cristianini, "Convex methods for transduction," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

Ivor Wai-Hung Tsang received the B.Eng. and M.Phil. degrees in the computer science from the Hong Kong University of Science and Technology (HKUST), Clear Water Bay, in 2001 and 2003, respectively. He is currently pursuing the Ph.D. degree at HKUST.

His scientific interests includes machine learning and kernel methods.

Mr. Tsang was the Honor Outstanding Student at HKUST in 2001.

James Tin-Yau Kwok received the Ph.D. degree in computer science from the Hong Kong University of Science and Technology (HKUST), Clear Water Bay, in 1996.

He was with the Department of Computer Science, Hong Kong Baptist University, as an Assistant Professor. He returned to the HKUST in 2000, where he is currently an Assistant Professor with the Department of Computer Science. His research interests include kernel methods, machine learning, pattern recognition, and artificial neural networks.