Efficient Hyperkernel Learning Using Second-Order Cone Programming

Ivor W. Tsang and James T. Kwok

Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{ivor,jamesk}@cs.ust.hk

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 [12]. 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. Experimental results on both toy and real-world data sets show significant speedup. Moreover, in comparison with the kernel matrix learning method proposed by Lanckriet *et al.* [7], our proposed SOCP-based hyperkernel method yields better generalization performance, with a speed that is comparable to their formulation based on quadratically constrained quadratic programming (QCQP).

1 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 [14]. 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. The associated kernel parameters (such as the order in a polynomial kernel) can then be determined by optimizing a quality functional of the kernel [12], such as kernel target alignment, generalization error bounds, Bayesian probabilities and cross-validation error.

Instead of adapting only the kernel parameters, a recent development is on adapting also the form of the kernel itself. 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 the kernel function by just learning the kernel matrix instead. As the kernel matrix must be positive semidefinite (psd), Lanckriet *et al.* [7] used semidefinite programming (SDP) to optimize a cost function (such as the hard/soft margin[1]) on the train-

[1] When the kernel-target alignment [5] is used instead, [7] can be shown to be a generalization of the kernel learning method in [5].

ing 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.* [7] constrained the kernel matrix to be in a convex subset of psd matrices with a fixed trace. Other kernel matrix learning methods, such as using boosting to optimize a weighted combination of base kernels [4], and the use of a Bayesian hierarchical model together with the Tanner-Wong data augmentation algorithm [16], have also been recently proposed.

However, transduction requires knowing the test patterns in advance and thus may not be always appropriate. In an inductive setting, a novel approach that learns the kernel *function* directly is the method of hyperkernels [12]. By introducing the notion of a hyper reproducing kernel Hilbert space, the desired kernel function can be obtained by minimizing a regularized quality functional, in which the capacity of the search space is explicitly controlled by a regularization term. It can be shown that the kernel function obtained is always a linear combination of a finite number of pre-specified hyperkernel evaluations. Often, this is further constrained to be a positive linear combination. As will be discussed in Section 2, 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 [12]. Recently, simultaneous optimization of both sets of variables is made possible by formulating this as a SDP problem [11].

However, even with the recent advances in interior point methods, solving SDPs is still very computationally 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 [11, 12], this problem is partially alleviated by reducing the number of variables using a low rank approximation (e.g., [6]) on the hyperkernel matrix. While this often makes the SDP problem more tractable, an alternative formulation faster than SDP is still very desirable.

Moreover, as mentioned earlier, the kernel function obtained can be expressed as a positive linear combination of hyperkernels. Consequently, the training kernel matrix becomes a positive linear combination of some pre-specified matrices. By regarding this derived property rather as a constraint in the kernel learning process, the approach in [7] can also be applied in this inductive setting, and the computational problem reduces to solving a quadratically constrained quadratic programming (QCQP) problem, which is much faster than solving SDPs. Recently, Bousquet and Herrmann [3] proposed an even faster method based on gradient descent. Note that the hyperkernel method cannot be similarly reduced to a QCQP because it uses the hyperkernel prior, while [3, 7] use the trace of the kernel matrix for complexity control. However, although these QCQP-based and descent-based methods have significant speed advantages over the SDP-based hyperkernel method, our experiments in Section 5 show that the hyperkernel method has better generalization performance over 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 [7]. In particular, we will show that the hyperkernel method can be equivalently formulated as a second-order cone program (SOCP) [1]. 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 solving SOCPs have a much better worst-case complexity and run far more efficiently in practice than those for SDPs [1].

The rest of this paper is organized as follows. Section 2 reviews the method of hyperkernels and Section 3 gives a short introduction on SOCP. Section 4 describes the proposed general procedure for obtaining a SOCP formulation for the learning problem, which is then followed by some specific kernel method examples. Experimental results on both toy and real-world data sets are presented in Section 5, and the last section gives some concluding remarks.

2 Learning with Hyperkernels

In this section, we review the method of hyperkernels as introduced in [11–13]. Section 2.1 first introduces the concepts of reproducing kernel Hilbert space (RKHS) and regularized risk functional minimization, which are then extended to hyper-RKHS and regularized quality functional minimization in Section 2.2.

In the sequel, $\mathbf{A} \succeq 0$ means that the matrix $\mathbf{A}$ is symmetric and positive semidefinite (psd), $\mathbb{R}^k_+, \mathbb{R}^k_{++}$ denote the sets of non-negative and positive vectors in $\mathbb{R}^k$ respectively, and $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ is the training set. Moreover, vector/matrix transpose is denoted by the superscript ^{T}.

2.1 Reproducing Kernel Hilbert space

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 *reproducing kernel Hilbert space* (RKHS) with kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ if

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

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

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

where $\|f\|_{\mathcal{H}_k}$ is the RKHS norm of f and $\lambda \in \mathbb{R}_{++}$ is a user-defined constant. 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 $1 \le i \le m$.

Instead of solving the primal in (1) directly, most kernel methods solve its dual instead, which is usually a quadratic programming (QP) problem of the form

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

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.

2.2 Hyper Reproducing Kernel Hilbert space

Denote $\underline{\mathcal{X}} = \mathcal{X} \times \mathcal{X}$. Consider a Hilbert space of functions $k : \underline{\mathcal{X}} \to \mathbb{R}$. Analogous to a RKHS, the Hilbert space $\underline{\mathcal{H}}$ of functions $k : \underline{\mathcal{X}} \to \mathbb{R}$ is a *hyper reproducing kernel Hilbert space* (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: $\langle k, \underline{k}(\underline{\mathbf{x}}, \cdot)\rangle_{\underline{\mathcal{H}}} = k(\underline{\mathbf{x}}), \quad \forall k \in \underline{\mathcal{H}}, \forall \underline{\mathbf{x}} \in \underline{\mathcal{X}}$, where $\langle \cdot, \cdot \rangle_{\underline{\mathcal{H}}}$ denotes the dot product in $\underline{\mathcal{H}}$;
2. $\underline{k}$ spans $\underline{\mathcal{H}}$; and
3. For any fixed $\underline{\mathbf{x}} \in \underline{\mathcal{X}}$, the hyperkernel $\underline{k}$ is a valid kernel in its second argument.

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 regularized quality functional over the entire space $\underline{\mathcal{H}}$ of kernels, which leads to the primal:

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

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

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

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 often constrained to be non-negative. 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:

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

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:

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

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 $\mathbf{1}^T\boldsymbol{\beta} = c$, where $\mathbf{1} = [1, \ldots, 1]^T \in \mathbb{R}^{m^2}$ 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 [11].

3 Second-Order Cone Programming

In recent years, it has been found that many optimization problems can be formulated as second-order cone programs (SOCP) [1, 8, 10]. SOCP is 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[2]. Its standard primal form is:

$$\min_{\mathbf{x}_1, \ldots, \mathbf{x}_r} \mathbf{c}_1^T \mathbf{x}_1 + \ldots + \mathbf{c}_r^T \mathbf{x}_r \ \text{ s.t. } \ \mathbf{A}_1 \mathbf{x}_1 + \ldots + \mathbf{A}_r \mathbf{x}_r = \mathbf{b}, \ \ \mathbf{x}_i \succeq_{\mathcal{Q}} \mathbf{0},$$

where $\mathbf{x}_i \in \mathbb{R}^{n_i}, \mathbf{c}_i \in \mathbb{R}^{n_i}, \mathbf{b} \in \mathbb{R}^m, \mathbf{A}_i \in \mathbb{R}^{m \times n_i}$ and inequalities of the form $\mathbf{x} \succeq_{\mathcal{Q}} \mathbf{y}$ means $\mathbf{x} - \mathbf{y} \in \mathcal{Q}$. The corresponding dual is

$$\max_{\mathbf{y}, \mathbf{z}_1, \ldots, \mathbf{z}_r} \mathbf{b}^T \mathbf{y} \ \text{ s.t. } \ \mathbf{A}_i^T \mathbf{y} + \mathbf{z}_i = \mathbf{c}_i, \ \ \mathbf{z}_i \succeq_{\mathcal{Q}} \mathbf{0},$$

where $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{z}_i \in \mathbb{R}^{n_i}$. 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 (LP), has been developed for SOCPs. In particular, the strong duality theorem [1] guarantees that if the primal and dual problems have strictly feasible solutions, then both have optimal solutions $\mathbf{x}^*, (\mathbf{y}^*, \mathbf{z}^*)$ with zero duality gap (i.e., $\mathbf{c}^T \mathbf{x}^* = \mathbf{b}^T \mathbf{y}^*$).

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 [15]. Thus, the complexity of SOCP falls between those of SDP and QCQP. While SOCPs can be solved as SDPs, it is however not recommended to do so [1, 8]. Interior point methods that solve SOCP directly have a much better worst-case complexity and run far more efficiently in practice than solving the corresponding SDP.

[2] The *second-order cone* $\mathcal{Q}$ is the norm cone for the Euclidean norm: $\mathcal{Q} = \{\mathbf{x} = [x_0, \tilde{\mathbf{x}}^T]^T \ : \ x_0 \geq \|\tilde{\mathbf{x}}\|\}$, where $\mathbf{x} = [x_0, x_1, \ldots, x_n]^T \in \mathbb{R}^n$, $\tilde{\mathbf{x}} = [x_1, \ldots, x_n]^T \in \mathbb{R}^{n-1}$ and $\| \cdot \|$ denotes the standard Euclidean norm.

4 SOCP Formulation

In Section 4.1, we will show that the general optimization problem in (6) can be equivalently formulated as a SOCP, which can then be solved more efficiently than a SDP. The specific SOCP formulations for two popular kernel classifiers, namely, C-SVM and ν-SVM, are then discussed in Section 4.2. Because of the lack of space, SOCP formulations for the other kernel methods, will be reported in a longer version of this paper.

4.1 General Formulation

First, the Lagrangian for (2) is $\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\gamma}) = \boldsymbol{\alpha}^T\mathbf{v} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{G}(\mathbf{K})\boldsymbol{\alpha} + \boldsymbol{\gamma}^T(\mathbf{A}\boldsymbol{\alpha}+\mathbf{b})$, where $\boldsymbol{\gamma} \in \mathbb{R}^n_+$. Setting its derivative w.r.t. $\boldsymbol{\alpha}$ to zero, we have[3]

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

Substituting this back into (6), we then obtain:

$$\begin{aligned}
&\min_{\boldsymbol{\beta},\boldsymbol{\gamma}} && \frac{1}{2}(\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v}) + \boldsymbol{\gamma}^T\mathbf{b} + \frac{\lambda_Q}{2}\boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta} \\
&\text{s.t.} && \mathbf{1}^T\boldsymbol{\beta} = c,\ \ \boldsymbol{\beta},\boldsymbol{\gamma} \geq \mathbf{0},\ \ \mathbf{K} = \text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta}) \\
= &\min_{\boldsymbol{\beta},\boldsymbol{\gamma},t_1,t_2} && \frac{1}{2}t_1 + \boldsymbol{\gamma}^T\mathbf{b} + \frac{\lambda_Q}{2}t_2 \\
&\text{s.t.} && t_1 \geq (\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v}),\ \ t_2 \geq \boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta}, \\
& && \mathbf{1}^T\boldsymbol{\beta} = c,\ \ \boldsymbol{\beta},\boldsymbol{\gamma} \geq \mathbf{0},\ \ \mathbf{K} = \text{reshape}(\underline{\mathbf{K}}\boldsymbol{\beta}).
\end{aligned} \tag{7}$$

Now, we will utilize two techniques on converting problems to SOCPs as discussed in [1, 10]. Let $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{s} = [s_1,\ldots,s_k]^T \in \mathbb{R}^k_+$, and

$$\mathbf{M}(\mathbf{s}) = \sum_{i=1}^{k} s_i\mathbf{M}_i, \tag{8}$$

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 $t \geq \mathbf{y}^T\mathbf{M}(\mathbf{s})^{-1}\mathbf{y}$, where $t \in \mathbb{R}_+$, can be replaced by the system:

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

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 4.2, $\mathbf{G}(\mathbf{K})$ can be written in the form of (8) in many kernel methods, i.e.,

$$\mathbf{G}(\mathbf{K}) = \sum_{i,j=1}^{m}\beta_{ij}\mathbf{G}_{ij}, \tag{9}$$

[3] When $\mathbf{G}(\mathbf{K})$ is only positive semidefinite, we can use the pseudo-inverse $\mathbf{G}(\mathbf{K})^+$ instead of $\mathbf{G}(\mathbf{K})^{-1}$.

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.,

$$\mathbf{G}_{ij} = \tilde{\mathbf{G}}_{ij}\tilde{\mathbf{G}}_{ij}^T, \tag{10}$$

which is always possible as $\mathbf{G}_{ij} \succeq 0$. The constraint $t_1 \geq (\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v})^T\mathbf{G}(\mathbf{K})^{-1}(\mathbf{A}^T\boldsymbol{\gamma}+\mathbf{v})$ in (7) can thus be replaced by

$$\sum_{i,j=1}^{m} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = \mathbf{A}^T\boldsymbol{\gamma} + \mathbf{v}, \;\; t_1 \geq \sum_{i,j=1}^{m} \tau_{ij}, \;\; \beta_{ij}\tau_{ij} \geq \mathbf{c}_{ij}^T\mathbf{c}_{ij}, \tag{11}$$

where $\tau_{ij} \in \mathbb{R}_+, r_{ij} = \operatorname{rank}(\mathbf{G}_{ij})$ 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 $x+y \geq \left\| \begin{bmatrix} 2\mathbf{w} \\ x-y \end{bmatrix} \right\|$. Define $\underline{\mathbf{G}}$ by

$$\underline{\mathbf{K}} = \underline{\mathbf{G}}\underline{\mathbf{G}}^T. \tag{12}$$

Then, for the constraints $\beta_{ij}\tau_{ij} \geq \mathbf{c}_{ij}^T\mathbf{c}_{ij}$ in (11) and

$$t_2 \geq \boldsymbol{\beta}^T\underline{\mathbf{K}}\boldsymbol{\beta} = (\underline{\mathbf{G}}^T\boldsymbol{\beta})^T(\underline{\mathbf{G}}^T\boldsymbol{\beta}) \tag{13}$$

in (7), as $\beta_{ij}, \tau_{ij}, t_2 \in \mathbb{R}_+$, they can be rewritten as $\beta_{ij} + \tau_{ij} \geq \left\| \begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix} \right\|$ and $t_2 + 1 \geq \left\| \begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix} \right\|$, respectively.

Finally, putting these and (11) back into (7), we obtain

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\gamma},t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \;\; & \frac{1}{2}t_1 + \boldsymbol{\gamma}^T\mathbf{b} + \frac{\lambda_Q}{2}t_2 \\ \text{s.t.} \;\; & \sum_{i,j=1}^{m} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = \mathbf{A}^T\boldsymbol{\gamma} + \mathbf{v}, t_1 \geq \sum_{i,j=1}^{m} \tau_{ij}, \\ & \beta_{ij} + \tau_{ij} \geq \left\| \begin{bmatrix} 2\mathbf{c}_{ij} \\ \beta_{ij} - \tau_{ij} \end{bmatrix} \right\|, \;\; t_2 + 1 \geq \left\| \begin{bmatrix} 2\underline{\mathbf{G}}^T\boldsymbol{\beta} \\ t_2 - 1 \end{bmatrix} \right\|, \\ & \mathbf{1}^T\boldsymbol{\beta} = c, \;\; \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\tau} \geq \mathbf{0}, \end{aligned} \tag{14}$$

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

In this kernel learning setting, it can be shown that the solving of SDP, SOCP and QCQP by interior point methods yield worst-case complexities of $O(n^{6.5})$, $O(n^5)$ and $O(n^5)$, respectively [7].

4.2 Examples

We first consider the C-SVM, whose dual is:

$$\max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T(\mathbf{K} \odot \mathbf{y}\mathbf{y}^T)\boldsymbol{\alpha} \;\; \text{s.t.} \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0, \;\; \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1},$$

where $\mathbf{1} = [1, \ldots, 1]^T \in \mathbb{R}^m, \mathbf{y} = [y_1, \ldots, y_m]^T \in \mathbb{R}^m$, $\odot$ denotes the Hadamard product and $C \in \mathbb{R}_+$ is a user-defined parameter. Comparing with (2), we have $\mathbf{v} = \mathbf{1}$ and $\mathbf{G}(\mathbf{K}) = \mathbf{K} \odot \mathbf{y}\mathbf{y}^T = (\sum_{i,j=1}^m \beta_{ij}\underline{\mathbf{K}}_{ij}) \odot \mathbf{y}\mathbf{y}^T = \sum_{i,j=1}^m \beta_{ij}(\underline{\mathbf{K}}_{ij} \odot \mathbf{y}\mathbf{y}^T)$, on using (5). Thus, $\mathbf{G}(\mathbf{K})$ is of the form in (9), with $\mathbf{G}_{ij} = \underline{\mathbf{K}}_{ij} \odot \mathbf{y}\mathbf{y}^T \succeq 0$. 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 (10) and (12) respectively, and $\boldsymbol{\gamma} = [\boldsymbol{\eta}, \boldsymbol{\xi}, b_1, b_2]^T$, (14) then becomes

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

where $b = b_1 - b_2$ can be shown to be the bias in C-SVM.

Another popular kernel classifier is the ν-SVM, in which $\nu \in (0, 1)$ controls the fractions of errors and support vectors. Its dual is

$$
\max_{\boldsymbol{\alpha}} -\frac{1}{2}\boldsymbol{\alpha}^T(\mathbf{K} \odot \mathbf{y}\mathbf{y}^T)\boldsymbol{\alpha} \;\; \text{s.t.} \;\; \boldsymbol{\alpha}^T\mathbf{y} = 0, \;\; \boldsymbol{\alpha}^T\mathbf{1} \geq 1, \;\; \mathbf{0} \leq \boldsymbol{\alpha} \leq \frac{1}{\nu m}\mathbf{1}.
$$

Again, on comparison with (2), we have $\mathbf{v} = \mathbf{0}$, $\mathbf{G}(\mathbf{K}) = \mathbf{K} \odot \mathbf{y}\mathbf{y}^T$ (as for C-SVM), $\mathbf{A} = [\mathbf{I}_m, -\mathbf{I}_m, -\mathbf{y}, \mathbf{y}, \mathbf{1}]^T$ and $\mathbf{b} = [\mathbf{0}^T, \frac{1}{\nu m}\mathbf{1}^T, 0, 0, -1]^T$. Define $\tilde{\mathbf{G}}_{ij}$ and $\underline{\mathbf{G}}$ as for C-SVM, $\boldsymbol{\gamma} = [\boldsymbol{\eta}, \boldsymbol{\xi}, b_1, b_2, \rho]^T$ and $b = b_1 - b_2$, (14) then becomes

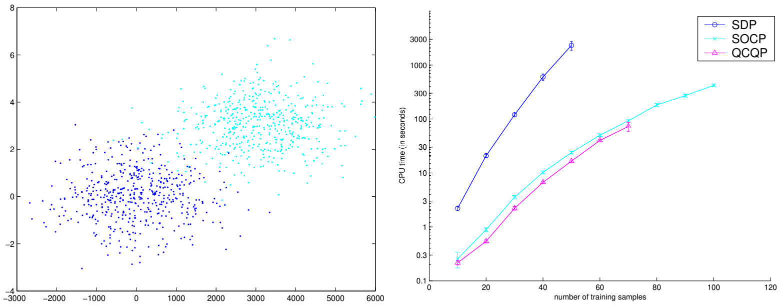
$$
\begin{aligned}
\min_{\boldsymbol{\beta},\boldsymbol{\eta},\boldsymbol{\xi},\rho,b,t_1,t_2,\boldsymbol{\tau},\mathbf{c}} \quad & \frac{1}{2}t_1 - \rho + \frac{1}{\nu m}\boldsymbol{\xi}^T\mathbf{1} + \frac{\lambda_Q}{2}t_2 \\
\text{s.t.} \quad & \sum_{ij} \tilde{\mathbf{G}}_{ij}\mathbf{c}_{ij} = -b\mathbf{y}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

5 Experiments

The use of hyperkernels has been shown to produce SVMs that are always equally good, or better than, traditional SVMs tuned by cross-validation [11]. In this section, we demonstrate the speed-up that can be obtained by replacing the SDP formulation in [11] with our SOCP formulation. Because of the lack of space, only classification experiments using the C-SVM (Section 4.2) in an inductive setting are performed. We use the ARD (automatic relevance determination) hyperkernel, with the same setting as in [11]. For comparison, we also perform [7] under this setting. In particular, the candidate kernel matrix is *assumed* to be of the form in (5), with its trace fixed at $\sum_{ij} \text{trace}(\underline{\mathbf{K}}_{ij})$. The mixing coefficients β_{ij}'s are to be learned, resulting in a QCQP. We use SDPT3 as the SDP solver[4], and MOSEK for solving SOCP and QCQP. All implementations are in Matlab, and the experiments are performed on a 2.4GHz Pentium 4 machine, with 1GB memory.

Comparison has also been made with the method proposed in [3] (which is adapted for the soft-margin C-SVM here). Analogous to the QCQP-based method, gradient descent is performed w.r.t. the β_{ij}'s in (5). However, because of the nonlinear nature of the optimization problem and also the large number (m^2) of β_{ij}'s to be learned[5], its performance turns out to be highly sensitive to the optimizer used. In particular, if a simple gradient descent scheme with a fixed learning rate is adopted, it can be faster than the QCQP-based method but the kernel solution has poor generalization performance, especially on the real-world data sets in Section 5.2. On the other hand, if a more sophisticated solver is used[6], its generalization performance becomes almost comparable to that of the hyperkernel method, but the learning is even much slower than hyperkernel's SDP formulation. Thus, because of this large variability, results on this descent-based method will not be reported in the sequel.

5.1 Toy Data Set



(a) Data distribution. (b) CPU time required.

Fig. 1. The toy data set (Note that the y-axis in Figure 1(b) is in log scale).

The first experiment is performed on the two-class data used in [11] (Figure 1(a)). It is generated by two Gaussian distributions with highly non-isotropic variance (the standard deviation is 1 in one dimension and 1000 in the other). In order to better demonstrate the computational requirements of the different

[4] A recent benchmark study in [9] shows that SDPT3 is able to solve larger SDP problems than SeDuMi (used in [11]).

[5] Note that linear combinations of only three kernel matrices have been conducted in the experiments of [3].

[6] To be more specific, we used the fmincon function in Matlab's Optimization Toolbox.

formulations, low rank approximation on the hyperkernel matrix is not used in this toy problem. 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 1 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. In practice, there may be minor differences due to the use of different optimization solvers. Notice that the QCQP-based method also yields comparable generalization performance on this simple toy problem. However, its implementation takes more storage and so has to stop at a training set size of 70. Moreover, we will see in Section 5.2 that there is also a significant difference in generalization performance on the real-world data sets. On the other hand, the speeds of the SDP and SOCP formulations are vastly different. As can be seen from Figure 1(b), our SOCP formulation, which is as fast as the QCQP-based method, is about 100 times faster than that of SDP. In fact, the SDP formulation is so slow that we have to stop when the training set size reaches 50. 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.

Table 1. 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	-

5.2 Real-world Data Sets

The second set of experiments are performed on seven real-world data sets[7]. 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 [12, 13]. We observe that the resultant matrix ranks obtained are in the range 10-20.

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

Results on the various methods are shown in Table 2. As can be seen, the test set accuracies obtained by the SDP and SOCP formulations are almost identical, while significant speedup can be achieved by using SOCP. Moreover, in terms of 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 difference 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.

Table 2. Test set accuracies and CPU time on the real-world 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

6 Conclusion

In this paper, we show that the method of hyperkernels can be equivalently formulated as a SOCP, which can be solved more efficiently than the traditional SDP formulation. Experimental results on using the C-SVM on both toy and real-world data sets demonstrate significant speedups. We also demonstrate that the hyperkernel method yields better generalization performance than the QCQP-based method in [7]. The combination of this 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. 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, recently, Bach *et al.* [2] proposed a fast kernel learning method based on the sequential minimal optimization algorithm, and this will be also be studied in the future. Finally, note that the hyperkernel method can also be easily extended for transduction problems, and its comparison with the transductive method in [7] will be further explored.

References

1. F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming, Series B*, 95:3–51, 2003.
2. F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the Twentieth-First International Conference on Machine*, Banff, Alberta, Canada, July 2004.
3. O. Bousquet and D.J.L. Herrmann. On the complexity of learning the kernel matrix. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.
4. K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.
5. N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.
6. S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representation. *Journal of Machine Learning Research*, 2:243–264, December 2001.
7. 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.
8. M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra Applications*, 284:193–228, 1998.
9. H.D. Mittelmann. An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming Series B*, 95:407–430, 2003.
10. Y. Nesterov and A. Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994.
11. C.S. Ong and A.J. Smola. Machine learning with hyperkernels. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 568–575, Washington DC, USA, 2003.
12. C.S. Ong, A.J. Smola, and R.C. Williamson. Hyperkernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.
13. C.S. Ong, A.J. Smola, and R.C. Williamson. Learning the kernel with hyperkernels, 2003. Submitted to Journal of Machine Learning Research.
14. B. Schölkopf and A.J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.
15. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.
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 *Proceedings of the Twentieth-First International Conference on Machine Learning*, Banff, Alberta, Canada, July 2004.