Neural Comput & Applic
DOI 10.1007/s00521-012-1022-2

ORIGINAL ARTICLE

A primal method for multiple kernel learning

Zhifeng Hao · Ganzhao Yuan · Xiaowei Yang · Zijie Chen

Received: 12 February 2012 / Accepted: 21 June 2012

Abstract The canonical support vector machines (SVMs) are based on a single kernel, recent publications have shown that using multiple kernels instead of a single one can enhance interpretability of the decision function and promote classification accuracy. However, most of existing approaches mainly reformulate the multiple kernel learning as a saddle point optimization problem which concentrates on solving the dual. In this paper, we show that the multiple kernel learning (MKL) problem can be reformulated as a BiConvex optimization and can also be solved in the primal. While the saddle point method still lacks convergence results, our proposed method exhibits strong optimization convergence properties. To solve the MKL problem, a two-stage algorithm that optimizes canonical SVMs and kernel weights alternately is proposed. Since standard Newton and gradient methods are too time-consuming, we employ the truncated-Newton method to optimize the canonical SVMs. The Hessian matrix need not be stored explicitly, and the Newton direction can be computed using several Preconditioned Conjugate Gradient steps on the Hessian operator equation, the algorithm is shown more efficient than the current primal approaches in this MKL setting. Furthermore, we use the Nesterov's optimal gradient method to optimize the kernel weights. One remarkable advantage of solving in the primal is that it achieves much faster convergence rate than solving in the dual and does not require a two-stage algorithm even for the single kernel LapSVM. Introducing the Laplacian regularizer, we also extend our primal method to semi-supervised scenario. Extensive experiments on some UCI benchmarks have shown that the proposed algorithm converges rapidly and achieves competitive accuracy.

Keywords Multiple kernel learning · Support vector machines · Laplacian semi-supervised learning · Truncated-Newton method · Nesterov's optimal gradient method

Z. Hao · G. Yuan (✉) · Z. Chen
School of Computer Science and Engineering,
South China University of Technology,
Guangzhou 510006, People's Republic of China
e-mail: y.ganzhao@scut.edu.cn

Z. Hao
e-mail: mazfhao@scut.edu.cn

Z. Chen
e-mail: c.zijie@scut.edu.cn

Z. Hao
Faculty of Computer Science, Guangdong University
of Technology, Guangzhou, People's Republic of China

X. Yang
Department of Mathematics, South China University
of Technology, Guangzhou 510641, People's Republic of China
e-mail: xwyang@scut.edu.cn

1 Introduction

Kernel methods have been extensively used in a variety of learning tasks with the best known example of support vector machines [20]. They work through mapping the input data into a high-dimensional (possibly infinite-dimensional) feature space, where the mapping is represented by introducing a kernel. The kernel can intuitively compute the similarity between two examples. Let $\mathcal{L} = \{x_i, y_i\}_i^n$ denote the dataset where x_i belongs to some input space and $y_i = \{+1, -1\}$ is the class label of x_i. The result of support vector machines (SVMs) learning is of the form given in Eq. (1).

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i k(x_i, x) + b\right) \tag{1}$$

Published online: 12 July 2012

where $k(\cdot,\cdot)$ is a given kernel associated with a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$. So long as the kernel matrix $\mathbf{K}$ is positive semidefinite, the optimization function is convex and convergence of the training algorithm is guaranteed. However, in the typical use of the kernel method algorithm, the choice of the kernel, which is vital to improved performance, is left to the user. One has to hand-craft the parameters in advance to select a good kernel. This problem can in principle be solved by cross-validation; however, the machine learning community may be interested in more flexible models. Recently, a so-called multiple kernel learning (MKL) [15] technique has shown the necessity to consider multiple kernels instead of a single fixed one. In general, MKL tries to form an ensemble kernel to enhance interpretability of the decision function and promote classification accuracy. In such case, a common approach is to consider that the kernel $k(x_i, x_j)$ is a convex linear combination of some basis kernels: $k(x_i, x_j) = \sum_{m=1}^{M} d_m\ k^m\ (x_i, x_j)$ with $\sum_{m=1}^{M} d_m = 1$, $d_m \geq 0$ where M is the total number of kernels.

There are two active research directions in multiple kernel learning. The first one is to promote the training speeds of MKL algorithms by transforming the block regularization [4, 15, 21] (or mixed-norm regularization) to the (un)weighted sum l_2-norm regularization [14, 18, 23]. The MKL problem has been introduced by Lanckriet et al. [15], leading to a Quadratically Constrained Quadratic Programming (QCQP) problem that becomes rapidly intractable as the number of training instances or kernels become large. The MKL problem can in principle be solved by some off-the-shelf optimization software, but the uses of such general purpose algorithms will only give poor efficiency. Moreover, the kernel learning problem is non-smooth, making the direct application of gradient descent methods infeasible, Bach et al. [4] therefore added a second weighted sum l_2-norm regularization to the objective function of the MKL, making the problem smooth where gradient method such as SMO can be applied. Sonnenburg et al. [21] reformulated the MKL problem of [15] as a Semi-Infinite Linear Program (SILP). The advantage of the proposed method is that the algorithm can be solved by iteratively using existing single kernel SVMs code. However, one disadvantage of the above approaches is that they employ mixed-norm regularization that only results in slow convergence. Recently, based on the MKL framework in Bach et al. [4], Rakotomamonjy et al. [18] proposed a new formulation of the MKL problem and successfully applied it into the SimpleMKL that makes MKL more practical for large-scale learning. They replace the mixed-norm regularization with a weighted l_2-norm regularization which leads to a smooth and saddle point optimization problem. By using a variational formulation of the mixed-norm regularization, they proved that the proposed formulation was equivalent to the ones of Bach et al. [4], Lanckriet et al. [15] and Sonnenburg et al. [21].

The second one is to promote the predictive accuracy of the MKL. Generally speaking, this can be achieved in two ways. (1) Multiple kernel learning looks for a decision function of the form in Eq. (2).

$$f(x) = \operatorname{sign}\left(\sum_{m=1}^{M} d_m f_m(x) + b\right) \tag{2}$$

where $f_m(x)$ is associated with different kernels or different kernel parameters. The simplest way is to use an unweighted sum of kernel functions. Using an unweighted sum gives equal preference to all kernels, but this may not be optimal. A better approach is to learn a weighted sum kernels, and this also allows extracting information from the weighted kernels and leading to a more adaptive discriminant function. (2) In order to avoid overfitting, some regularization techniques on the weights are needed. l_1-norm of the kernel weights, also known as the simplex constraint, is mostly used in MKL methods. The advantage of the simplex constraint is that it leads to a sparse solution, that is, only a few base kernels carry significant weights. However, it is reported that [8] kernel selection with l_1-regularization can lead to modest improvement but to performance degradation in larger-scale classification task. In contrast, l_p regularization ($p \geq 2$) [14, 23] regularization seldom degrades performance. Varma et al. [22] extended the existing MKL formulations to learn general kernel combinations subject to general regularization. It is demonstrated that the proposed formulation can lead to better results not only as compared to canonical MKL but also as compared to state-of-the-art wrapper and filter methods for feature selection.

In this paper, we discuss a primal method for multiple kernel learning. Solving in the primal enjoys a lot of advantages. It achieves much faster convergence rate than solving in the dual for the single kernel LapSVM because it does not require a two-stage algorithm to solve the optimization [16]. It is also shown that it is more suitable for large-scale optimization [12]. Keerthi et al. greedily selected a set of kernel basis functions of a specified maximum size to approximate the SVMs primal cost function and shown that it achieved similar accuracy with standard SVMs. Last but not least, most of existing approaches mainly reformulate the multiple kernel learning as a saddle point problem that concentrates on solving the dual optimization problem. In this paper, we show that the multiple kernel learning problem can be reformulated as a BiConvex optimization and can also be solved in the primal. While the saddle point method still lacks convergence results, our proposed methods exhibit strong optimization convergence properties. Motivated by the recent interest in

solving SVMs in the primal [7, 12, 16], we present a primal approach to the MKL problem that can also achieve comparable result to existing MKL methods. Specifically, our contributions are: (a) We extend the regularization techniques described in [24] to our primal MKL formulations and generalize to more multiple kernel learning algorithms. From the primal point of view, Sparse SVMs [12], Primal Regression [6], Laplacian SVMs [16], one-class SVMs [19], kernel ridge regression [8] can be trained in a multiple kernel way. (b) We proposed a new two-stage algorithm to solve the primal MKL problem. We employ a second-order method to optimize canonical SVMs and Nesterov's optimal gradient method to optimize the kernel weights alternately. Since our implementation uses a special kernel cache technique, the update of kernel weights is very cheap. (c) We apply the truncated-Newton method to minimize the MKL problem. Since the Hessian matrix need not be stored and the Newton direction can be computed using several Preconditioned Conjugate Gradient steps on the Hessian operator equation, the algorithm is more efficient than current quadratic hinge loss SVMs. (d) Introducing the Laplacian regularizer, we also extend our multiple kernel learning algorithm to semi-supervised scenario.

The rest of this paper is organized as follows. Our primal multiple kernel learning problem is introduced in Sect. 2. In Sect. 3, we present an efficient two-stage algorithm for solving the primal MKL problem. Section 4 provides some extensions of our work. Extensive experiments dealing with efficiency and comparison with other MKL methods are presented in Sect. 5. Finally, we conclude this paper in Sect. 6.

2 Multiple kernel learning framework

2.1 Preliminaries

In the standard supervised learning setup, we find a hypothesis $f \in \mathcal{H}$, that is generalized well on new and unseen data. Applying regularized risk minimization returns the minimize f^*:

$$f^* = \arg\min_f \lambda\Omega(f) + R_{\text{emp}}(f).$$

where $R_{\text{emp}}(f) = \sum_{i=1}^n \ell(y_i, f(x_i))$ is the empirical risk of hypothesis f, Ω is the regularizer. This formula can be also viewed as a multi-objective optimization, where λ ($\lambda \geq 0$) is a tuning parameter used to balance the effects of the two items. An SVM builds the decision rule $sign(f^*(x) + b^*)$, where f^* and b^* are defined as the solution of Eq. (3).

$$Q(f,b) = \min_{f,b} \lambda\|f\|_{\mathcal{H}}^2 + \sum_{i=1}^n \ell(y_i, f(x_i) + b) \tag{3}$$

2.2 Learning with multiple kernels

Considering the linear convex combination of the decision functions, we can express Eq. (3) in terms of d_m and $f_m(x)$ in Eq. (2), thus obtain Eq. (4):

$$\min_{d,f,b} \lambda \left\| \sum_{m=1}^M d_m f_m \right\|_{\mathcal{H}}^2 + \sum_{i=1}^n \ell\left(y_i, \sum_{m=1}^M d_m f_m(x_i) + b\right) \tag{4}$$

Equation (4) is a generalization of Eq. (3). While (4) is intuitive, it is hard to introduce the representation theorem. Here we follow the regularization techniques proposed in Ref. [24] and solve a relaxed optimization problem of Eq. (4) and we reach Eq. (5).

$$\min_{f,b} \min_{d} \lambda \sum_{m=1}^M d_m \|f_m\|_{\mathcal{H}}^2 + \sum_{i=1}^n \ell\left(y_i, \sum_{m=1}^M d_m f_m(x_i) + b\right) \tag{5}$$

Equation (5) is not necessarily convex due to the introduced variables d_m. On the other hand, in order to avoid overfitting, we use some regularization techniques on the weights d_m. We restrict the weights in a norm ball as $\sum_{m=1}^M d_m^p = 1$, $d_m \geq 0$. Since the convex combination of convex sets is also convex, Eq. (5) is transformed into a BiConvex optimization [5, 10, 11]. To encourage sparse kernel combinations, we mainly focus on the l_1-norm kernel basis selection formulation, that is $\mathcal{S} : \sum_{m=1}^M d_m = 1, d_m \geq 0$. This leads to the following optimization:

$$\min_{f,b} \min_{d} \left\{ \lambda \sum_{m=1}^M d_m \|f_m\|_{\mathcal{H}}^2 + \sum_{i=1}^n \ell\left(y_i, \sum_{m=1}^M d_m f_m(x_i) + b\right) \right\}$$
$$s.t. d \in \mathcal{S} \tag{6}$$

Proposition 1 *The objective function of* Eq. (6) *is the upper bound of* Eq. (4).

Proof Since $\| \sum_i^M d_m f_m \|_{\mathcal{H}}^2 = \| \sum_i^M \frac{d_m f_m}{d_m^{1/2}} d_m^{1/2} \|_{\mathcal{H}}^2$, using Cauchy-Schwarz inequality:

$$\left(\sum_i^M x_i y_i \right)^2 \leq \left(\sum_{j=1}^M x_j^2 \right) \left(\sum_{k=1}^M y_k^2 \right)$$

we obtain: $\| \sum_i^M d_m f_m \|_{\mathcal{H}}^2 \leq \sum_i^M \| \frac{d_m f_m}{d_m^{1/2}} \|_{\mathcal{H}}^2 \sum_i^M \| d_m^{1/2} \|_2^2 = \sum_i^M d_m \|f_m\|_{\mathcal{H}}^2 \sum_i^M d_m$. Noticing $\sum_i^M d_m = 1$, $d_m \geq 0$, we reach: $\| \sum_i^M d_m f_m \|_{\mathcal{H}}^2 \leq \sum_i^M d_m \|f_m\|_{\mathcal{H}}^2$. Therefore, Proposition 1 holds. □

Hence, minimizing the l_1-norm regularizer in Eq. (6), we are actually minimizing an upper bound of the true regularizer $\sum_{m=1}^M \|d_m f_m\|_{\mathcal{H}}^2$ in Eq. (4). Equation (6) can be solved by deriving its dual formulation, which is identical

to simpleMKL [18]. However, we solve this problem directly in the primal. In the following, we will show how to kernelize the optimization problem Eq. (6).

2.3 Kernelization

Let us now consider an SVM problem with an associated Reproducing Kernel Hilbert Space $\mathcal{H}$. Following [7], we use the representation theorem: $w = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ and seek a solution of the form:

$$f(x) = \sum_{i=1}^{n} \alpha_i \langle \phi(x_i), \phi(x) \rangle + b = \sum_{i=1}^{n} \alpha_i K(x_i, x) + b$$

Let us express Eq. (3) in terms of α:

$$Q(f, b) = \lambda \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) + \sum_{i=1}^{n} \ell \left(y_i, \sum_{j=1}^{n} \alpha_j k(x_i, x_j) + b \right) \tag{7}$$

where we use the kernel reproducing property in

$$\|f\|_{\mathcal{H}}^2 = \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle k(x_i, .), k(x_j, .) \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

We use the same denotations as in Ref. [7] and introduce the kernel matrix $\mathbf{K}$ with $K_{ij} = k(x_i, x_j)$ and k_i is the *ith* row of $\mathbf{K}$, then Eq. (7) can be rewritten as Eq. (8):

$$\lambda \alpha^T K \alpha + \sum_{i=1}^{n} \ell(y_i, K_i \alpha + b) \tag{8}$$

Since we will solve Eq. (8) in the primal, the smoothness of the loss function is important for the algorithm design. When the loss function is smooth, it makes the optimization problem continuous and differentiable in f. In such case, the smooth optimization can be directly applied. In this paper, we only discuss the smooth quadratic hinge loss function [7, 12, 16]:

$$\ell(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))^2$$

After introducing the quadratic hinge loss function and adding the scaling constant 1/2 to Eq. (5), we obtain the following optimization problem:

$$\min_{f \in \mathcal{H}, d} \frac{1}{2} \left\{ \lambda \sum_{m=1}^{M} d_m \|f_m\|_{\mathcal{H}}^2 + \sum_{i=1}^{n} \max \left(0, 1 - y_i \sum_{m=1}^{M} d_m f_m(x_i) + b \right)^2 \right\} \quad s.t. d \in \mathcal{S} \tag{9}$$

Again, using the representation theorem, we can rewrite Eqs. (9) as (10):

$$F(\alpha, b, d) = \min_{\alpha, b, d} \frac{1}{2} \lambda \alpha^T \mathbf{K} \alpha + \sum_{i=1}^{n} \max(0, 1 - y_i(k_i \alpha + b))^2 \quad s.t. \ \mathbf{K} = \sum_{m=1}^{M} d_m \mathbf{K}^m, \ k_i = \sum_{m=1}^{M} d_m k_i^m, \ d \in \mathcal{S} \tag{10}$$

where k_i^m is the ith row of the mth kernel.

3 Algorithm for the primal MKL

Let us focus on the optimization problem Eq. (10). For simplicity, we denote the canonical SVMs parameters α and b as $z = [\alpha^T \ b]^T$ where $\alpha \in \mathbb{R}^{n \times 1}, b \in \mathbb{R}^{1 \times 1}, F(z)$ as the objective function in Eq. (10) with respect to z while fixing d and $F(d)$ with respect to d while fixing z. The optimization problem is a BiConvex Optimization problem (separately but not jointly convex with respect to z and d). Instead of trying to solve the problem directly, we can use a two-step alternate optimization algorithm to find the discriminant function. The first step is to find the optimal cannonial SVMs z while fixing the kernel weights d, the second step is to update d while fixing z. The optimization of kernel weights d is an inner loop update in our algorithm. The complete algorithm of PrimalMKL is summarized in Algorithm 1.

Algorithm 1 Primal multiple kernel learning

1: $z^{(0)} = [\alpha^{(0)T} b^{(0)}]^T = \mathbf{0} \in \mathbb{R}^{(n+1) \times 1}$, set $t = 0$
2: initialize $d \in \mathbb{R}^{M \times 1}$ randomly
3: **while** not converge **do**
4: update K: $K = \sum_m d_m K^m$
5: minimize (11) to find the search direction $q^{(t)}$ //Algorithm 2
6: **if** $\|\nabla_z\|_2 < \chi_2$ **then**
7: return;
8: **end if**
9: find s^* by exact Newton line search: $s^* = \min_s F(z^{(t)} + s \times q^{(t)})$
10: $z^{(t)} = z^{(t)} + s^* \times q^{(t)}$
11: update kernel cache: $K^m \alpha^{(t)}$, $\alpha^{(t)T} K^m \alpha^{(t)}$, $m = [1{:}M]$
12: optimize the kernel weights d //Algorithm 4
13: $t = t + 1$
14: **end while**

3.1 Optimize canonical SVMs z

Fixing d, the optimization problem Eq. (10) becomes a canonical SVM optimization. Note that this is an unconstrained optimization problem. In Ref. [7, 12], the authors

investigated efficient solutions for the nonlinear SVMs. They mainly used Newton and Preconditioned Conjugate Gradient (PCG) method to optimize z. In Ref. [16], the authors have shown that the Preconditioned Conjugate Gradient method is more suited for Laplacian SVM than Newton method due to the nature of the intrinsic regularizer. Generally, it is believed that for small scale problem, Newton method will converge faster; for large-scale problem, however, due to the expensive Hessian matrix generation and inversion in the update rule, Newton method may give modest performance. In such case, PCG method is preferred. On the other hand, the gradient-based method such as PCG is known to achieve slow convergence near optimal. Although every update of z is very cheap, it takes more iterations to obtain a reasonable solution. Since we have to optimize the kernel weights, more iterations will cause more evaluation of the gradient of the objective function with respect to d. Once the kernel number is large, it is expensive to re-evaluate the gradient. We need a model that takes less iteration to converge to the minima like Newton method.

In order to solve this optimization problem more efficiently, we use the truncated-Newton [13]. Truncated-Newton builds a quadratic model to find a search direction that gives a good trade-off of the progress in each iteration and the number of iterations. To discuss the truncated-Newton method, we need to compute the gradient and Hessian matrix of $F(z)$. Before continuing, we introduce the concept of support vector set [7]. Support vector set is the subset of $\mathcal{L}$ with the points that generate a l_2 loss value greater than zero, that is a point x_i belongs to the support vector set if $y_i f(x_i) < 1$. Therefore, we have:

$$\begin{aligned}\nabla_z = \begin{bmatrix}\nabla_\alpha \\ \nabla_b\end{bmatrix} &= \begin{bmatrix}\lambda\mathbf{K}\alpha + \sum_{i\in sv}(1 - y_i(k_i\alpha + b))(-y_i k_i) \\ \sum_{i\in sv}(1 - y_i(k_i\alpha + b))(-y_i)\end{bmatrix} \\ &= \begin{bmatrix}\lambda\mathbf{K}\alpha + \mathbf{K}\mathbf{I}_{sv}(\mathbf{K}\alpha + \mathbf{1}b) - \mathbf{K}\mathbf{I}_{sv}y \\ \mathbf{1}^T\mathbf{I}_{sv}(\mathbf{K}\alpha + \mathbf{1}b) - \mathbf{1}\mathbf{I}_{sv}y\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{K} & \mathbf{0} \\ \mathbf{0}^T & 1\end{bmatrix} \times \begin{bmatrix}\lambda\mathbf{K}\alpha + \mathbf{I}_{sv}(\mathbf{K}\alpha + \mathbf{1}b) - \mathbf{I}_{sv}y \\ \mathbf{1}^T\mathbf{I}_{sv}(\mathbf{K}\alpha + \mathbf{1}b) - \mathbf{1}\mathbf{I}_{sv}y\end{bmatrix}\end{aligned}$$

where $\mathbf{1} \in \mathbb{R}^{n\times 1}, \mathbf{0} \in \mathbb{R}^{n\times 1}$ is a column vector having all elements equal to one and zero respectively, $y \in \mathbb{R}^{n\times 1}$ is the vector that collects the n labeled training points and ∇_z is the gradient of the objective function in Eq. (10). We will use the symbols ∇_α and ∇_b to indicate the gradient with respect to α and b. The matrix $\mathbf{I}_{sv} \in \mathbb{R}^{n\times n}$ is a diagonal matrix where the only elements different from 0 (and equal to 1) along the primal diagonal are in positions corresponding to points of $\mathcal{L}$ that belong to support vector set at the current iteration. The Hessian $\mathbf{H}$ can be computed as

$$\begin{aligned}\mathbf{H} = \begin{bmatrix}\nabla_\alpha^2 & \nabla_\alpha(\nabla_b) \\ \nabla_b(\nabla_\alpha) & \nabla_b^2\end{bmatrix} &= \begin{bmatrix}\mathbf{K}\mathbf{I}_{sv}\mathbf{K} + \lambda\mathbf{K} & \mathbf{K}\mathbf{I}_{sv}\mathbf{1} \\ \mathbf{1}^T\mathbf{I}_{sv}\mathbf{K} & \mathbf{1}^T\mathbf{I}_{sv}\mathbf{1}\end{bmatrix} \\ &= \mathbf{P} \times \begin{bmatrix}\mathbf{I}_{sv}\mathbf{K} + \lambda\mathbf{I} & \mathbf{I}_{sv}\mathbf{1} \\ \mathbf{1}^T I_{sv}\mathbf{K} & \mathbf{1}^T I_{sv}\mathbf{1}\end{bmatrix}\end{aligned}$$

where

$$\mathbf{P} = \begin{bmatrix}\mathbf{K} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{1}\end{bmatrix}$$

Newton direction involves solving the following linear system: $\mathbf{H} \times q = -\nabla_z$ exactly. Therefore, $\mathbf{P}$ can be viewed as the *preconditioner* of the linear system.

Approximate Newton direction: Instead of trying to solve the linear system exactly to obtain the search direction, we only find an approximate solution using Preconditioned Conjugate Gradient ($\mathbf{P}$ is the preconditioner). After indicating ∇^2 the second-order derivative of $F(z)$, the truncated-Newton uses an approximate direction q in every iteration t, requiring only that:

$$\|\nabla^2 F(z^{(t)})q^{(t)} + \nabla F(z^{(t)})\|_2 \leq \eta\|\nabla F(z^{(t)})\|_2 \tag{11}$$

that is, that the linear residual is small. We refer to any vector q that satisfies Eq. (11) with $0 < \eta < 1$ as an approximate Newton direction. The method is also known as a Hessian-free Newton or inexact Newton method, because the Hessian-vector products can be computed without explicitly forming the Hessian. The Newton direction is only computed up to a specific error tolerance. The main operation of certain iterative Newton method is the product between the Hessian matrix and a vector $p \in \mathbb{R}^{(n+1)\times 1}$. After denoting $p = [p_1^T\ p_2]^T$, where $p_1 \in \mathbb{R}^{n\times 1}, p_2 \in \mathbb{R}^{1\times 1}$, we obtain:

$$\begin{aligned}\mathbf{H} \times p = \mathbf{P} \times \begin{bmatrix}\mathbf{I}_{sv}\mathbf{K} + \lambda\mathbf{I} & \mathbf{I}_{sv}\mathbf{1} \\ \mathbf{1}^T I_{sv}\mathbf{K} & \mathbf{1}^T\mathbf{I}_{sv}\mathbf{1}\end{bmatrix} \times \begin{bmatrix}p_1 \\ p_2\end{bmatrix} \\ = \mathbf{P} \times \begin{bmatrix}(\mathbf{I}_{sv}\mathbf{K} + \lambda\mathbf{I})p_1 + (\mathbf{I}_{sv}\mathbf{1})p_2 \\ (\mathbf{1}^T\mathbf{I}_{sv}\mathbf{K})p_1 + (\mathbf{1}^T\mathbf{I}_{sv}\mathbf{1})p_2\end{bmatrix}\end{aligned}$$

Next, we will use conjugate gradient method to minimize the left-hand side of Eq. (11). The complete algorithm of PCG method for approximately finding the Newton direction is named Newton-PCG and summarized in Algorithm 2. Commonly, the numbers of iterations to find the approximate Newton direction strongly depend on the precision η, so Algorithm 2 can commonly terminate after a few iterations. One might be concerned about terminating the Newton-PCG subroutine early, especially because an approximate solution to Eq. (11) will in general not be a descent direction. Indeed, the Newton-PCG subroutine can always yield a descent direction even under early termination if at least one Newton-PCG iteration is performed.

Algorithm 2 Newton-PCG

1: $r = -\nabla f(x), \rho^{(0)} = \|r\|_2^2, k = 1, q = \mathbf{0}$, choose $\eta \in (0, 1)$
2: **while** not converge **do**
3: **if** $k = 1$ **then**
4: $p = r$
5: **else**
6: $p = r + \left(\rho^{(k-1)}/\rho^{(k-2)}\right)p$
7: **end if**
8: $w = H \times p$
9: $\alpha = \rho^{(k-1)}/(p^T w)$
10: $q = q + \alpha\, p$
11: $r = r - \alpha\, w$
12: $\rho_k = \|r\|^2$
13: $k = k + 1$
14: **if** $(\sqrt{\rho^{(k-1)}} < \eta\|r\|$ *or* $k > kmax)$ **then**
15: break;
16: **end if**
17: **end while**
18: output q as the approximate Newton direction

Linear search: Once the approximate Newton direction is found, we need to compute the optimal step. The one-dimensional function $F(z^{(t-1)} + s \times q^{(t-1)})$ is a continuously differentiable, strictly convex, piecewise quadratic function. Following Chapelle [7], we use exact Newton line search to find the optimal steps (Line 1, Algorithm 1). A few Newton iterations to find the optimal step can ensure the global convergence of the optimization problem Eq. (10).

3.2 Optimize the kernel weights d

Several methods can be used to optimize the kernel weights (SMO optimization [4]; Cutting-planes [21]; Projected Gradient Descent [18]). Rakotomamonjy et al. show that the projected gradient descent is more stable since we have a differentiable function to be minimized. In our experiments, we employ the projected gradient descent method for the sake of the stability of the kernel weights. We use the symbol ∇_d to indicate the gradient of d. So ∇_d can be written as:

$$\nabla_{d_m} = \frac{\lambda}{2}\alpha^T K^m \alpha + \sum_{i \in sv}(1 - y_i(k_i\alpha + b))(-y_i k_i^m \alpha)$$

Once the gradient has been computed, d is updated by gradient descent while ensuring that the simplex constraint on d is satisfied. This can be done by l_1-ball projection method [9]. To compute the Euclidean projection onto the simplex, one can solve the optimization problem (12).

$$\min_d \frac{1}{2}\|d - d'\|_2^2 \, s.t. \, d \in \mathcal{S} \tag{12}$$

where d' denotes the last feasible solution of the kernel weight. Such a projection operator has been implemented efficiently in linear time in [9], which is described in Algorithm 3. To achieve faster optimization, we employ Nesterov's optimal gradient method [17] to accelerate the gradient decent. As a brilliant achievement in the optimization field, Nesterov's method has a much faster convergence rate than the traditional methods such as subgradient method or naïve projected gradient descent. We describe the kernel weights update by Nesterov's method in Algorithm 4.

Kernel cache: Note that the update of d involves querying the objective function and the derivative of d thus needs the computation of $\mathbf{K}^m\,\alpha$ and $\alpha^T\,\mathbf{K}^m\,\alpha$. However, we

Algorithm 3 Simplex projection

1: input: A vector $q \in R^{m\times 1}$
2: sort q into v such that $v_1 \geq v_2 \geq \ldots \geq v_m$
3: find $\delta = \max\{j \in [1:m] : v_j - \frac{1}{j}\left(\sum_{r=1}^{j} v_r - 1\right) > 0\}$
4: compute $\theta = \frac{1}{\rho}\left(\sum_{r=1}^{j} v_r - 1\right)$
5: output $\mathbf{x}$ s.t. $x_j = max(z_j - \theta, 0), j = [1{:}m]$

Algorithm 4 Nesterov's projection gradient for simple constraint optimization problem

1: input: $f(d)$, ∇_d, $d^{(0)}$
2: set $d^{(1)} = d^{(0)}$, $\delta^{(-1)} = 0$, $\delta^{(0)} = 1$, $i = 1$
3: Lipschitz parameter: $\gamma^{(0)} = 1$
4: **while** not converge **do**
5: $\alpha = \frac{\delta^{(i-2)}-1}{\delta^{(i-1)}}, s = d^{(i)} + \alpha(d^{(i)} - d^{(i-1)})$
6: **for** $j = 0$ to ... **do**
7: $\gamma = 2^j\gamma^{(i-1)}, u = s - \frac{1}{\gamma}\nabla_s$
8: Project u into the feasible set to obtain $d^{(t)}$
9: **if** $\triangle_d = \|s - d^{(i)}\|_\infty < \chi_1$ **then**
10: return;
11: **end if**
12: define function $g_{\gamma,s}(u) = f(s) + \nabla_v^T(u - s) + \frac{\gamma}{2}\|u - s\|_2^2$
13: **if** $f(d^{(i)}) \leq g_{\gamma,s}(u)$ **then**
14: $\gamma^{(i)} = \gamma$; $d^{(i+1)} = d^{(i)}$; break;
15: **end if**
16: **end for**
17: Set $\delta^{(i)} = \frac{1+\sqrt{1+4(\delta^{(i-1)})^2}}{2}$
18: $i = i + 1$
19: **end while**

can keep the vector in the memory and update it in every iteration of α, making the update of d very cheap.

3.3 Convergence analysis

As we have mentioned before, our multiple kernel leaning algorithm that can be solved by BiConvex optimization exhibits nice convergence property. The BiConvex optimization has been well investigated in [5, 10, 11] under a more general term 'block coordinate descent' or 'Gauss-Seidel' method. The past study [5] has shown that the block coordinate descent is guaranteed to converge to the stationary point for strictly convex problems. However, when the subproblem in Eq. (10) is convex (not strictly convex), the subproblem may have multiple optimal solutions, the convergence of Algorithm 1 may be problematic. Fortunately, for BiConvex optimization (only involves two blocks) [11], have shown that the strict convexity of the subproblem is not required. Therefore, from Corollary 2 of [11], we have the following convergence statement:

Proposition 2 *Suppose at the* tth *and* kth *updates for* z *and* d *respectively. Any limit point of the sequence* $\{z^{(t)}, d^{(k)}\}$, $t = 1\ldots+\infty, k = 1\ldots+\infty$ *generated by Algorithm 1 is a stationary point.*

Apart from the convergence, another issue is whether our BiConvex optimization algorithm has at least one stationary point. In optimization analysis, it is related to the boundedness of the feasible region. Since d is defined in the closed convex set $\mathcal{S}$ in our problem, the feasible region is always bounded. Lastly, in order to shed some theoretical light on some property of the stationary point, we establish the following statement.

Proposition 3 *Suppose that the sequence* $\{z^{(t)}, d^{(k)}\}, t = 1\ldots+\infty, k = 1\ldots+\infty$, *admits a limit point* z^*, d^*, *then we have*

$$\text{(i)}\ \frac{\partial F}{\partial z^*} = 0, \ \textit{and} \tag{13}$$

$$\text{(ii)}\ \frac{\partial F}{\partial d^*}(d - d^*) \geq 0,\ \forall d \in \mathcal{S} \tag{14}$$

Proof (i) Since $F(z)$ is convex and continually differential. Using the descent property, we obtain

$$F(z^{(t+1)}, d^{(k+1)}) \leq F(z^{(t)}, d^{(k+1)}) \leq F(z^{(t)}, d^{(k)})$$

Due to the local convergency, we obtain $F(z^{(t+1)}, d^{(k+1)}) = F(z^{(t)}, d^{(k+1)}) = F(z^{(t)}, d^{(k)})$. The truncated-Newton stops when $\frac{\partial F}{\partial z^*} = 0$, we get immediately Eq. (13). (ii) The proof is by contradiction. Suppose there exists a $d \in \mathcal{S}$ such that $\frac{\partial F}{\partial d^*}(d - d^*) < 0$. Sine $F(d^* + \pi(d - d^*))$ is differentiable at the point d^*, then it has a linear approximation at the point d^*:

$$F(d^* + \pi(d - d^*)) = F(d^*) + \pi(d - d^*)^T \frac{\partial F}{\partial d^*} + \frac{1}{2}\pi^2 (d - d^*)^T \nabla^2 F(\tilde{d})(d - d^*)$$

Here the last term is the Lagrange remainder, where $\tilde{d}$ is a point on the line segment $[d^*, d^* + \pi(d - d^*)]$, the values of π considered are between 0 and 1 so that $\tilde{d}$ is on the line segment $[d^*, d]$ [5]. (i) Since $y = d^* + \pi(d - d^*) = (1 - \pi) d^* + \pi d$ is the convex combination of d and d^*, therefore $y \in \mathcal{S}$. Moreover, since we have $\frac{\partial F}{\partial d^*}(d - d^*) < 0$, there always exists a line search parameter $0 < \pi < 1$ which is sufficiently small such that $F(y) < F(d^*)$ which implying that d^* is not a local minima. Therefore Eq. (14) holds. □

In the proposition above, we use the conditional gradient descent [5] (aka Frank-Wolfe algorithm) in our convergence analysis. This provides a general guideline on the local minima our optimization framework stops at.

4 Semi-supervised multiple kernel learning

We now extend our multiple kernel learning framework to semi-supervised scenario. Before discussing the multiple kernel version, we review the single kernel Laplacian Support Vector Machines (LapSVM). The key idea of LapSVM is to introduce the Laplacian regularizer to the learning problem. Specifically, the LapSVM estimates the target function by minimizing

$$f^* = \arg\min_{f \in \mathcal{H}} \ell(y_i, f(x_i)) + \gamma_A \|f\|_A^2 + \gamma_I \|f\|_I^2$$

where $\|f\|_A^2$ is the canonical Hilbert space regularizer that enforces a smoothness condition on the label examples, $\|f\|_I^2$ is the intrinsic geometry regularizer in the low-dimensional manifold that enforces smoothness on all examples, γ_A and γ_I are the weights of two regularizers. Introducing the graph Lapacian $\mathbf{L}$, we obtain:

$$f^* = \arg\min_{f \in \mathcal{H}} \ell(y_i, f(x_i)) + \gamma_A f^T f + \gamma_I f^T \mathbf{L} f$$

It is reported in Ref. [16] that LapSVM solving directly in the primal achieves much faster convergence than solving in the dual. Introducing the representation theorem, they consider the following unconstrained optimization problem:

$$\min_{\alpha \in R^{n\times 1}, b \in R^{1\times 1}} \frac{1}{2}\left\{ \gamma_A \alpha^T K \alpha + \gamma_I \alpha^T K L K \alpha + \sum_{i=1}^{n} \max(0, 1 - y_i(K_i \alpha + b))^2 \right\} \tag{15}$$

One remarkable advantage of the algorithm is that it allows us to efficiently solve a single problem without the need of a two-step solution. Our work is mainly to extend the primal single kernel LapSVM to the multiple kernel setting.

4.1 Related work

Argyriou et al. [2] proposed another different framework for kernel selection. They consider a greedy algorithm for learning the optimal kernel. Suppose $h(\mathbf{K})$ is the objection function with respect to the kernel $\mathbf{K}$, and $G(\omega)$ is the kernel with respect to the feasible parameter ω (the bandwidth of the Gaussian kernel), they find the optimal step v^* which satisfies

$$\arg\max_{v} h(vG(\omega)+(1-v)K), v \in (0,1]$$

Then the algorithm performs the following update to find the optimal kernel: $K \leftarrow vG(\omega)+(1-v)K$. They use the gradient descent method to find the feasible parameter ω, since the Gaussian kernels are continuously parameterized by a compact set, the algorithm can always find the optimal kernel which is the convex combination of the basis kernel. Due to the success of the kernel selection learning, they apply the algorithm to the semi-supervised learning paradigm [1]. Introducing the Laplacian regularizer, they associate the Laplacian graph with different kernels and train the algorithm by the method proposed in Ref. [2]. They use a gradient descent method to find the feasible bandwidth of the Gaussian kernel. While they only consider the least square loss in their work, we use the quadratic hinge loss for semi-supervised learning.

4.2 Proposed method

One advantage of solving the primal problem is that it can be easily extended to semi-supervised setting. In this section, we will show how the primal Laplacian SVMs [16] can be extended to the multiple kernel setting. The multiple kernel Laplacian SVMs can be formulated as follows:

$$\begin{aligned}\min_{f\in\mathcal{H}} \gamma_A \sum_{m=1}^{M} d_m \|f_m\|^2 + \gamma_I \sum_{m=1}^{M} d_m f_m^T \mathbf{L} f_m \\ + \sum_{i=1}^{l} \ell\left(y_i, \sum_{m=1}^{M} d_m f_m(x_i) + b\right) \\ s.t.\ d \in \mathcal{S}\end{aligned} \tag{16}$$

Proposition 4 The multiple kernel Laplacian regularizer: $\sum_{m=1}^{M} d_m f_m^T \mathbf{L} f_m$ in the optimization problem (16) upper bounds its true regularizer $f^T\mathbf{L}f$.

Proof Our proof is similar to Theorem 1.

$$\begin{aligned} f^T\mathbf{L}f &= \|\sqrt{\mathbf{L}}f\|_2^2 = \|\sqrt{\mathbf{L}}\sum_{m=1}^{M} d_m f_m\|_2^2 \\ &= \|\sum_{m=1}^{M}(\frac{d_m\sqrt{\mathbf{L}}f_m}{d_m^{1/2}})(d_m^{1/2})\|_2^2 \\ &\le \sum_{m=1}^{M}\|(\frac{d_m\sqrt{\mathbf{L}}f_m}{d_m^{1/2}})\|_H^2 \sum_{m=1}^{M}\|d_m^{1/2}\|_2^2 \\ &= \sum_{m=1}^{M} d_m f_m^T\mathbf{L}f_m \sum_{m=1}^{M} d_m = \sum_{m=1}^{M} d_m f_m^T\mathbf{L}f_m \end{aligned} \tag{17}$$

□

Again, we can easily kernelize the optimization problem Eq. (16) in the primal. Introducing the quadratic hinge loss, we obtain the optimization problem Eq. (18):

$$\begin{aligned} &F(\alpha,b,d) = \min_{\alpha,b,d}\frac{1}{2}\left\{\gamma_A\alpha^T\mathbf{K}\alpha + \gamma_I\alpha^T\mathbf{N}\alpha + \sum_{i=1}^{n}\ell(y_i,k_i\alpha+b)\right\} \\ &s.t.\ \mathbf{K} = \sum_{m=1}^{M} d_m\mathbf{K}^m,\ \mathbf{N} = \sum_{m=1}^{M} d_m\mathbf{K}^m\mathbf{L}\mathbf{K}^m, \\ &k_i = \sum_{m=1}^{M} d_m k_i^m,\ d \in \mathcal{S} \end{aligned} \tag{18}$$

The optimization problem Eq. (18) can be solved by truncated-Newton method and Nesterov's projection method alternatively which is similar to the Algorithm 1.

5 Experimental analysis

In this section, we demonstrate the efficiency and accuracy of our multiple kernel learning algorithms on several real-world datasets coming from the UCI repository. All the experiments are performed with matlab 7.8 on a single core of an Intel Celeron E3300 2.5GHZ processor with 2GB memory. Some Matlab scripts and datasets to reproduce the experiments are available at: http://code.google.com/p/primalmkl.

In the following, we will simply refer to the proposed supervised and semi-supervised algorithms as *PrimalMKL* and *LapMKL* respectively. For the categorical attributes, in order to convert them into numeric data, we use c numbers to represent an c-category attribute. Only one of c numbers is one, and others are zeros. In all experiments, we scale linearly each attribute to the range $[-1, +1]$. For the kernel weights d update, the Nesterov's gradient algorithm stops when kernel weights variation between two consecutive iterations is lower than 10^{-5} (i.e., $\chi_1 = 10^{-5}$, see Step 4 in Algorithm 4); for canonical SVMs z update, the truncated-Newton algorithm terminates when the norm of the gradient is lower than 0.1 (i.e., $\chi_2 = 0.1$, see Step 1 in

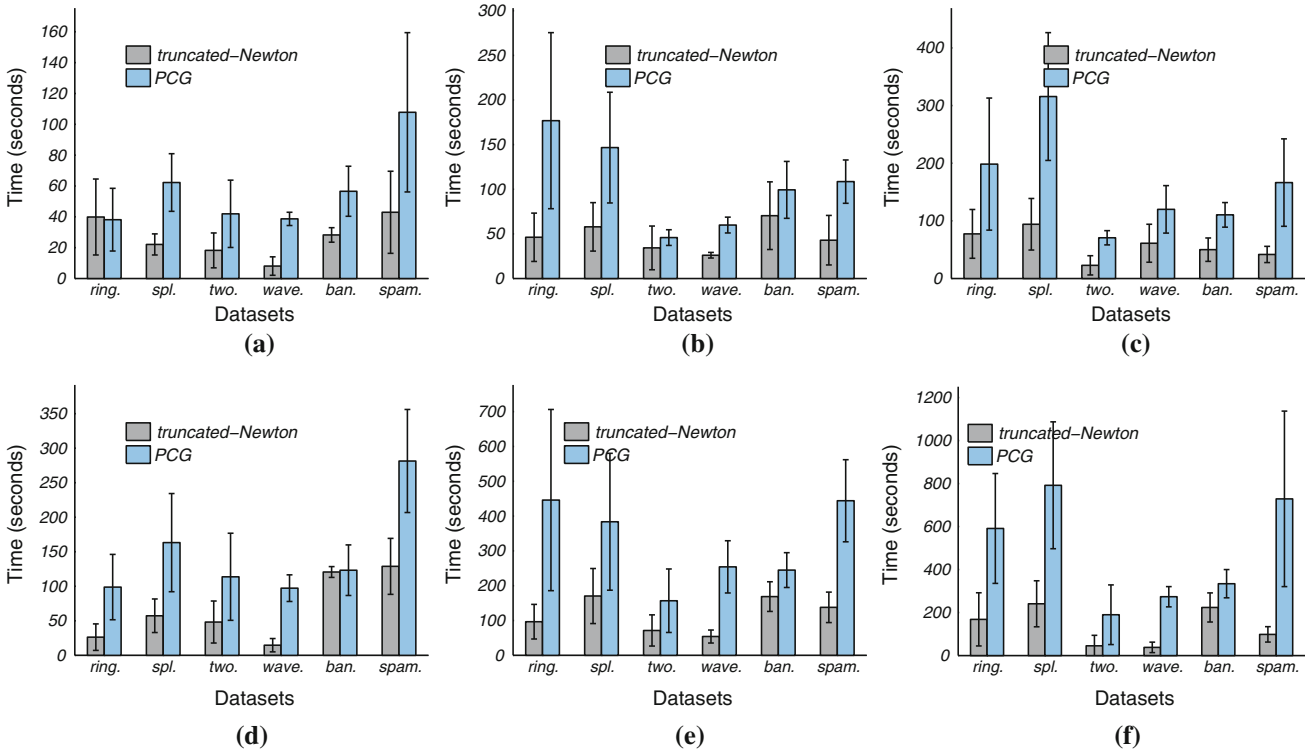


Fig. 1 Training time of PrimalMKL with different number of training patterns (N) and number of kernels (M) for dataset 'Ringnorm', 'Splice', 'Twonorm', 'Waveform', 'Banana', 'Spambase'. **a** $N = 1{,}000$, $M = 11$; **b** $N = 1{,}000$, $M = 22$; **c** $N = 1{,}000$, $M = 33$; **d** $N = 1{,}500$, $M = 11$; **e** $N = 1{,}500$, $M = 22$; **f** $N = 1{,}500$, $M = 33$

Algorithm 1). For the parameter of truncated-Newton method, the choice of $\eta = 0.1$ appears to give good performance for a wide range of problems. For each training set in all our experiments, the parameter λ, γ_A and γ_I are chosen using fivefold cross-validation on the training set. We search over a line of values {0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10}. In all the tables of this section, the best average results are in bold.

5.1 Efficiency

It is reported in Ref. [16] that the Newton method often scales badly with a few thouthands of training patterns due to the expensive Hessian matrix generation and inversion, while Preconditioned Conjugate Gradient (PCG) gives better performance than Newton. In this section, we compare our proposed method truncated-Newton with PCG to check how far our multiple kernel SVM solver has improved in this MKL setting. We use M kernels $(M = 11, 22, 33)$ and select N datapoints $(N = 1{,}000, 1{,}500)$ from the datasets: 'Ringnorm', 'Splice', 'Twonorm', 'Waveform','Spambase', 'Banana' and train the PrimalMKL algorithms. As demonstrated in Fig. 1, the truncated-Newton takes much less time to converge to the stationary point than PCG. Although the update of the PCG direction is very cheap, it takes more iterations to converge than truncated-Newton method and leads to expensive calculation for the multiple kernel learning problem. Note that in every iteration, the algorithm has to update the kernel cache ($K^m \alpha$ and $\alpha^T K^m \alpha$), so the less iteration the method takes, the faster convergence rate it provides. The truncated-Newton method uses PCG method in the inner iteration to find the approximate Newton direction and thus can be viewed as finding a good trade-off between number of iterations and cost in every iteration.

5.2 Accuracy of PrimalMKL

In this section, we conduct extensive experiments to check how accurate is our PrimalMKL to existing multiple kernel SVMs solvers, including SimpleMKL[1] [18], GMKL[2] [22] and SMO-MKL [23]. Before direct comparisons, we observe some publications on multiple kernel learning report their predictive accuracy with unit trace normalization in their experiment settings [18, 23], others considering abandon using the trace normalization [21, 22]. As it is pointed out in [3], group lasso consistency condition is not always satisfied, normalizing kernel matrices to unit trace may lead to

[1] http://asi.insa-rouen.fr/enseignants/~arakotom.

[2] http://research.microsoft.com/en-us/um/people/manik/.

Table 1 Trace-norm versus non-trace-norm for SimpleMKL and PrimalMKL

Dataset	SVMs	SimpleMKL	SimpleMKL(T)	PrimalMKL	PrimalMKL(T)
Breast	71.18 ± 3.00	74.55 ± 4.84	**74.74 ± 4.70**	74.22 ± 5.38	73.76 ± 4.59
Diabetis	76.27 ± 1.24	75.98 ± 1.77	75.37 ± 2.22	**76.50 ± 1.57**	75.90 ± 1.50
Flare	**65.02 ± 1.26**	64.96 ± 1.59	64.84 ± 2.08	64.18 ± 1.81	64.59 ± 1.89
German	75.03 ± 1.97	75.42 ± 2.69	74.95 ± 2.11	**75.52 ± 2.92**	74.33 ± 3.52
Heart	84.20 ± 2.20	83.15 ± 3.60	**84.35 ± 2.70**	83.25 ± 3.43	84.05 ± 3.96
Thyroid	95.07 ± 2.18	95.60 ± 2.74	94.73 ± 1.96	95.73 ± 2.15	**95.87 ± 2.07**
Titanic	77.65 ± 0.67	**77.90 ± 0.60**	77.49 ± 1.34	77.63 ± 1.36	77.22 ± 1.19
Splice	87.45 ± 0.60	87.79 ± 0.90	83.06 ± 5.07	**88.32 ± 0.64**	81.32 ± 8.29
Banana	89.45 ± 0.68	**89.60 ± 4.01**	88.95 ± 0.52	89.46 ± 0.50	89.18 ± 0.53
Waveform	89.45 ± 0.67	90.32 ± 0.45	90.29 ± 0.38	**90.40 ± 0.40**	88.06 ± 5.07

suboptimal predictive performance. In the following, we will verify their results and see whether we should use the unit trace normalization in our multiple kernel settings. We should use the same experiment settings for fair comparisons. We conduct experiments on the well-known Rätsch's benchmarks. Each dataset consists of 100 splits of training and test sets, of which 40 splits are used in this experiment. For the kernel parameter list, we use the following two settings.

- Gaussian kernel with 33 widths $(1.1^{(-5,-4,\ldots,5)}, 1.5^{(-5,-4,\ldots,5)}, 2^{(-5,-4,\ldots,5)})$ on all features and Polynomial kernels of degree 1 to 10 on all features. Each base kernel matrix is normalized to *unit trace*.
- Gaussian kernel with 33 widths $(1.1^{(-5,-4,\ldots,5)}, 1.5^{(-5,-4,\ldots,5)}, 2^{(-5,-4,\ldots,5)})$ on all features. *No* trace normalization is used in the base kernel matrices.

We denote SimpleMKL(T) and PrimalMKL(T) as the two solvers using unit trace normalization, SimpleMKL and PrimalMKL that *abandon* using trace normalization. We run our algorithm for 40 times and get the average accuracy. Table 1 lists classification performance on Rätsch's 10 benchmarks. For all methods performed classification on the dataset, the best result is highlighted by a bold font. For the *Trace-norm* version and *Non-trace-norm* version of SimpleMKL and PrimalMKL, the best result is highlighted by a underline font. With direct comparison of average values, two conclusions can be drawn. (1) PrimalMKL wins five out of ten datasets while SimpleMKL wins four out of ten, which means both of the two MKL solvers give similar performance. (2) *Non-trace-norm* SimpleMKL wins eight out of ten datasets while *Non-trace-norm* PrimalMKL wins seven out of ten dataset. This demonstrates that *Non-trace-norm* version often outperforms the *Trace-norm* version for the two MKL solvers. These results also confirm Bach's statements in [3].

In the following experiments, for fair comparison, we will abandon using unit trace normalization in all multiple kernel learning solvers. We run comparisons with GMKL, SMO-MKL and SimpleMKL on six datasets with N points, D dimensional features: 'Sonar' (N = 210, D = 59), 'Australian' (N = 690, D = 13), 'Liver' (N = 345, D = 5), 'Ionosphere' (N = 351, D = 33), 'Pima' (N = 768, D = 8) 'Wdbc' (N = 569, D = 30). We select 70% of the data points from the dataset for training and the remaining 30% for testing.

For GMKL, we employ the default setting of the solver. We generate D (# of features) kernels from each feature of the dataset and l_1 norm is used to regularize the kernel weights. For SMO-MKL and SimpleMKL, we use the same setting as PrimalMKL and generate 33 kernels. We employ bregman divergence regularizer and l_1 regularizer respectively. We run the three algorithms for 30 times. Table 2 lists classification performances on the 6 datasets. The number of the selected kernels is reported in brackets. As can be seen from Table 2, PrimalMKL achieves comparable accuracy to existing multiple kernel learning methods.

5.3 Accuracy of LapMKL

In this section, we demonstrate the accuracy of our proposed LapMKL by comparing against existing semi-supervised kernel methods, including GLKS-SSL[3] and single kernel LapSVM[4].

Like Argyriou et al. [1], we conduct our experiments on five-digit pairs (1,7), (2,3), (2,7), (3,8), (4,7) from the USPS dataset. The dimensionality of each digit image and the number of digits in each digit class of each set are 256 and 200, respectively. We follow the standard experimental methodology in [1] where 6 examples from each class are labeled. The graph laplacian matrices are created based on the k-nearest-neighbor criterion for $k = 10$ with the Euclidean, affine transformation, and tangent distances[5].

[3] http://ttic.uchicago.edu/∼argyriou/code/index.html

[4] http://www.dii.unisi.it/∼melacci/lapsvmp/

[5] Following Argyriou, We use matlab optimizer 'minconf' to obtain the affine transformation and tangent distances.

Table 2 Predictive accuracy comparisons with existing MKL solvers

Dataset	GMKL	SMO-MKL	SimpleMKL	PrimalMKL
Sonar	87.30 ± 3.17 (24.6)	86.90 ± 5.01 (02.3)	**88.17 ± 3.03 (15.60)**	86.08 ± 4.31 (02.1)
Australian	84.66 ± 1.64 (06.3)	86.41 ± 3.72 (12.1)	**87.31 ± 1.57 (12.35)**	85.58 ± 2.04 (03.4)
Ionosphere	94.25 ± 1.80 (14.7)	92.63 ± 1.41 (05.5)	94.34 ± 2.23 (24.65)	**94.50 ± 2.20 (17.1)**
Liver	65.42 ± 3.64 (06.0)	67.51 ± 2.65 (08.8)	65.67 ± 1.71 (7.30)	**67.81 ± 4.60 (01.1)**
Pima	76.28 ± 2.02 (05.5)	74.51 ± 2.60 (10.0)	76.67 ± 2.42 (17.7)	**76.88 ± 2.28 (15.3)**
Wdbc	97.23 ± 1.22 (10.9)	**97.50 ± 1.10 (13.3)**	96.75 ± 1.01 (24.75)	97.01 ± 0.93 (15.1)

Table 3 Predictive accuracy comparison on USPS dataset with the results in Ref. [1]

Dataset	EUCLIDEAN		TRANSFORMATION		TANGENT	
	GLKS-SSL	LapMKL	GLKS-SSL	LapMKL	GLKS-SSL	LapMKL
1 versus 7	98.45 ± 0.00	**100.00 ± 0.00**	98.49 ± 0.00	**100.00 ± 0.00**	98.97 ± 0.00	**100.00 ± 0.00**
2 versus 3	**96.80 ± 0.76**	94.69 ± 2.13	**97.37 ± 1.12**	95.64 ± 3.14	**98.85 ± 0.14**	98.14 ± 1.96
2 versus 7	96.09 ± 0.29	**96.76 ± 0.84**	**97.27 ± 0.51**	97.08 ± 1.28	**97.89 ± 0.22**	97.56 ± 0.94
3 versus 8	**93.94 ± 0.89**	93.55 ± 1.87	93.04 ± 0.71	**94.80 ± 2.36**	93.80 ± 0.58	**98.19 ± 0.88**
4 versus 7	97.37 ± 0.53	**98.62 ± 0.45**	98.09 ± 0.43	**98.94 ± 0.23**	98.87 ± 0.20	**99.36 ± 2.22**

Table 4 Predictive accuracy comparison on the sixteen benchmarks with supervised PrimalMKL, LapSVM and GLKS-SSL

Dataset	PrimalMKL	LapSVM	GLKS-SSL	LapMKL
Sonar	68.31 ± 7.78	73.13 ± 4.07	70.30 ± 4.59	**75.00 ± 3.63**
Australian	79.22 ± 9.98	85.08 ± 0.65	79.57 ± 1.73	**85.94 ± 1.18**
Ionosphere	91.10 ± 3.04	88.19 ± 1.95	80.28 ± 2.59	**92.09 ± 3.09**
Liver	61.03 ± 3.75	**65.11 ± 4.08**	55.28 ± 2.76	60.30 ± 4.89
Pima	74.87 ± 1.36	74.42 ± 1.21	67.17 ± 1.92	**74.88 ± 1.90**
Wdbc	93.83 ± 7.36	95.59 ± 0.62	94.92 ± 1.09	**95.69 ± 1.15**
Diabetis	**73.60 ± 4.14**	70.87 ± 2.39	66.65 ± 2.52	70.65 ± 2.70
Flare	64.22 ± 1.49	**65.43 ± 2.08**	58.60 ± 2.78	64.50 ± 1.48
Heart	70.48 ± 11.81	**86.51 ± 2.65**	85.40 ± 2.01	85.85 ± 2.34
Thyroid	91.25 ± 2.96	93.93 ± 2.20	86.62 ± 9.40	**95.40 ± 2.03**
Titanic	78.63 ± 1.82	**80.25 ± 3.17**	72.46 ± 3.27	79.92 ± 3.62
Splice	66.51 ± 10.75	**81.68 ± 0.94**	76.62 ± 1.40	80.67 ± 1.01
Banana	85.11 ± 5.13	86.27 ± 2.33	**88.56 ± 2.08**	87.14 ± 1.70
Waveform	83.77 ± 9.43	85.38 ± 2.47	79.27 ± 2.31	**85.88 ± 1.89**
Ringnorm	95.77 ± 1.44	97.69 ± 0.72	96.12 ± 0.30	**98.23 ± 0.48**
Breast	71.00 ± 1.61	**71.71 ± 3.52**	63.62 ± 3.09	69.50 ± 4.17

Each experiment is repeated for 10 times, and the average classification error rates tested on the unlabeled dataset are reported in Table 3. As can be seen from Table 3, LapMKL wins nine out of fifteen on the USPS dataset. The affine transformation and tangent distances achieve better performance than the simple Euclidean distance in the digital recognition classification problems.

In order to further verify the effectiveness of our LapMKL method, we conduct experiments on 16 benchmark datasets. The features and number of data points are described before. We only consider the Euclidean distance and use supervised multiple kernel learning algorithm PrimalMKL as our baseline method in this experiment. Twenty percentage of the data points from each class are labeled. For the single kernel LapSVM, we search over a line of the Gaussian kernel widths $\left(2^{(-7,-6\ldots7)}\right)$. Each experiment is repeated for 20 times, and the average classification error rates tested on the unlabeled dataset are reported in Table 4. LapMKL wins eight out of sixteen benchmarks while LapSVM wins six out of sixteen. LapMKL significantly outperforms GLKS-SSL while gives a little better performance than single kernel LapSVM.

5.4 Scaling properties of PrimalMKL and LapMKL

– *Stability*: To understand the evolution of kernel weights (i.e., d), we plot the evolution curves of the five largest kernel weights for datasets 'Sonar', 'Australian', 'Ionosphere', 'Liver', 'Pima', 'Wdbc' in Fig. 2. It confirms our statement in Sect. 3.2 that the projected gradient descent is stable. Results on the running time of the PrimalMKL for computing the approximate regularization paths are given in Fig. 3. They have been obtained by averaging 5 runs over different training sets.

– *Scalability*: In order to check how the training time scales with the number of kernels, we select M kernels ($M = 1 \sim 33$) to train the classifier. As can be seen from Fig. 3, as the training instances increase, the training time of PrimalMKL scales closer to linearly with the number of the input kernels. They have been obtained by averaging 5 runs over different training sets ('Sonar', 'Australian', 'Ionosphere', 'Liver', 'Pima', 'Wdbc').

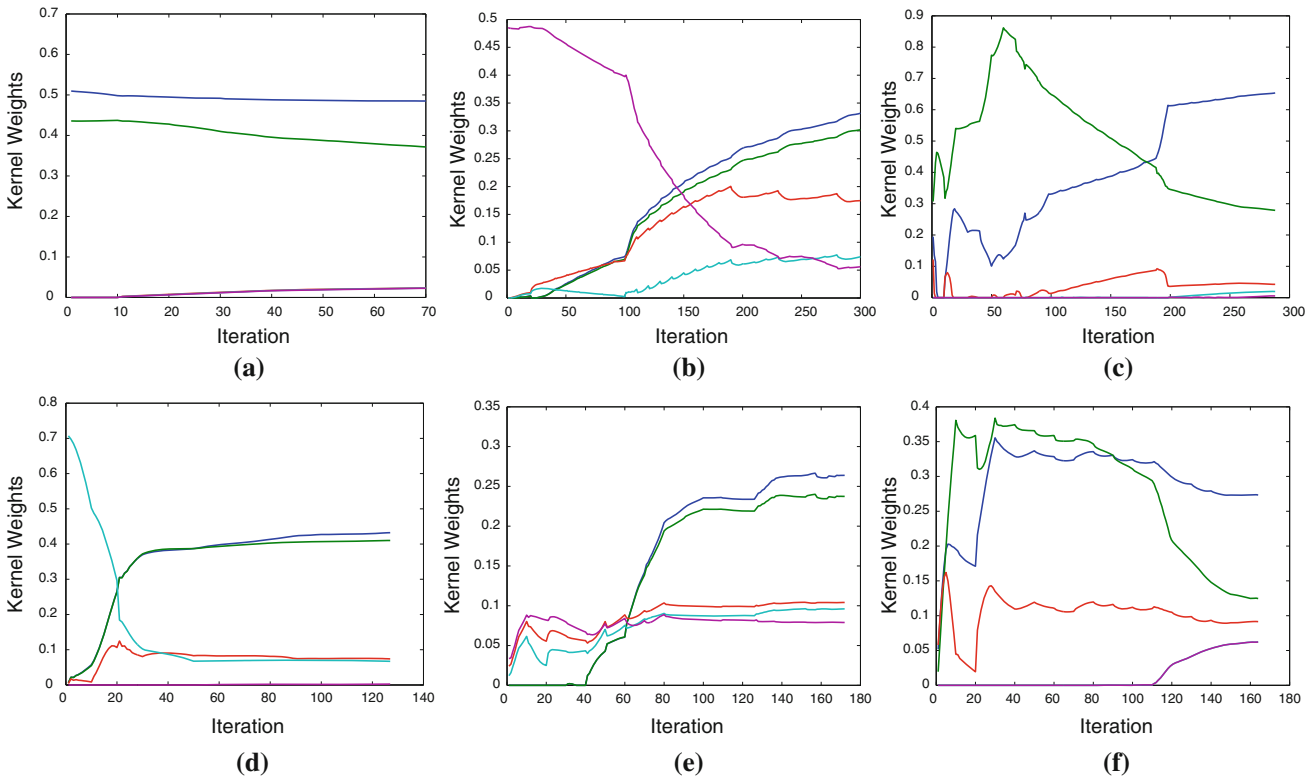
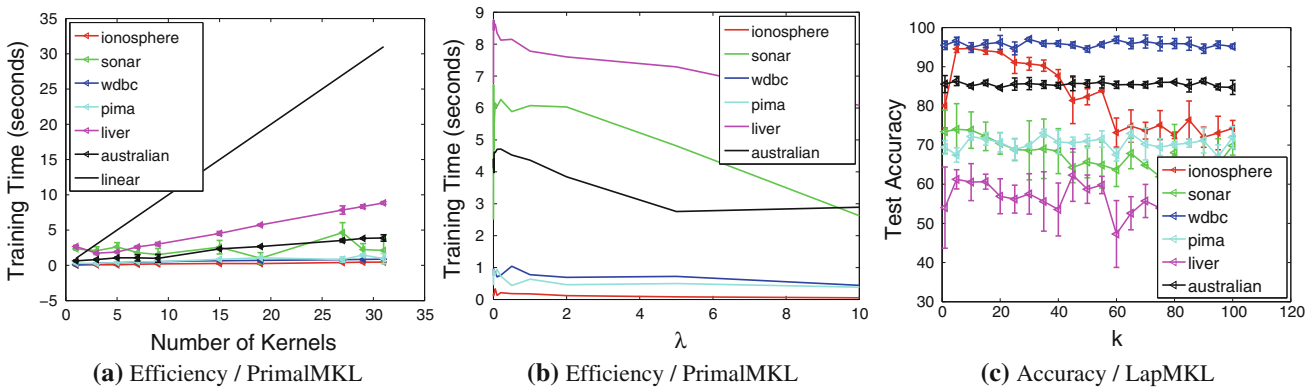


Fig. 2 The evolution curves of the five largest kernel weights for dataset 'Sonar', 'Australian', 'Ionosphere', 'Liver', 'Pima', 'Wdbc' computed by PrimalMKL. **a** *Sonar/PrimalMKL*; **b** *Aust./PrimalMKL*; **c** *Iono./PrimalMKL*; **c** *Liver/PrimalMKL*; **d** *Pima/PrimalMKL*; **e** *Wdbc/PrimalMKL*; **f** *Wdbc/PrimalMKL*



(a) Efficiency / PrimalMKL **(b)** Efficiency / PrimalMKL **(c)** Accuracy / LapMKL

Fig. 3 **a** CPU-time of PrimalMKL versus the number of kernels. **b** CPU-time of PrimalMKL versus λ. **c** Parameter k versus testing accuracy

– *Sensitivity*: One hyperparameter of our proposed semi-supervised multiple kernel learning method is k which is the number of nearest neighborhood of the graph Laplacian matrix. As can be seen in Table 3, this parameter k does not significantly affect the predictive accuracy and generally $k = 5 \sim 20$ seems works well for a wide range of problems.

6 Conclusions

In this paper, we introduce an approach for solving the multiple kernel learning problem. Different from the existing methods that focus on the dual problem, we solve the problem directly in the primal. Our approach uses a weighted l_2-norm regularization while imposing sparsity on the kernel weights. We also accelerate the learning algorithm by truncate-Newton and Nesterov's optimal gradient method. Extensive experiments on some real-world datasets come to two points: (1) unit trace normalization often leads to sub-optimal predictive accuracy and (2) our supervised multiple kernel learning algorithms give comparable results to existing multiple kernel learning solvers, and our semi-supervised version achieves state-of-the-art performance.

Acknowledgments This work is supported by NFS-China (61070033, 61100148), NSF-Guangdong (9251009001000005, S201104000 4804) and the Open Project of Key Laboratory of Symbolic Computation and Knowledge Engineering of the Chinese Ministry of Education (93K-17-2009-K04).

References

1. Argyriou A, Herbster M, Pontil M (2005) Combining graph laplacians for semi-supervised learning. In: NIPS, pp 67–74
2. Argyriou A, Micchelli CA, Pontil M (2005) Learning convex combinations of continuously parameterized basic kernels. In: COLT, pp 338–352
3. Bach FR (2008) Consistency of the group lasso and multiple kernel learning. J Mach Learn Res 9:1179–1225
4. Bach FR, Lanckriet GRG, Jordan MI (2004) Multiple kernel learning, conic duality, and the SMO algorithm. In: ICML, vol 69
5. Bertsekas D (1999) Nonlinear programming
6. Bo L, Wang L, Jiao L (2007) Recursive finite newton algorithm for support vector regression in the primal. Neural Comput 19(4):1082–1096
7. Chapelle O (2007) Training a support vector machine in the primal. Neural Comput 19(5)
8. Cortes C, Mohri M, Rostamizadeh A (2009) L2 regularization for learning kernels. In: UAI, pp 109–116
9. Duchi J, Shwartz SS, Singer Y, Chandra T (2008) Efficient projections onto the L1-ball for learning in high dimensions. In: ICML, pp 272–279
10. Gorski J, Pfeuffer F, Klamroth K (2007) Biconvex sets and optimization with biconvex functions: a survey and extensions. Math Methods Oper Res 66(3):373–407
11. Grippo L, Sciandrone M (2000) On the convergence of the block nonlinear gauss-seidel method under convex constraints. Oper Res Lett 26(3):127–136
12. Keerthi SS, Chapelle O, Decoste D (2006) Building support vector machines with reduced classifier complexity. J Mach Learn Res 7:1493–1515
13. Kelley CT (1999) Iterative methods for optimization. Frontiers in applied mathematics. SIAM, Thailand
14. Kloft M, Brefeld U, Sonnenburg S, Laskov P, Müller K-R, Zien A (2009) Efficient and accurate Lp-Norm multiple kernel learning. In: NIPS, pp 997–1005
15. Lanckriet GRG, Cristianini N, Bartlett P, Ghaoui LE, Jordan MI (2004) Learning the kernel matrix with semidefinite programming. J Mach Learn Res 5:27–72
16. Melacci S, Belkin M (2011) Laplacian support vector machines trained in the primal. J Mach Learn Res 12:1149–1184
17. Nesterov YE (2003) Introductory lectures on convex optimization: a basic course, volume 87 of applied optimization. Kluwer, Boston
18. Rakotomamonjy A, Bach FR, Canu S, Grandvalet Y (2008) Simple MKL. J Mach Learn Res 9:2491–2521
19. Schölkopf B, Smola AJ (2001) Learning with kernels: support vector machines, regularization, optimization, and beyond (adaptive computation and machine learning). The MIT Press, Cambridge
20. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
21. Sonnenburg S, Rätsch G, Schäfer C, Schölkopf B (2006) Large scale multiple kernel learning. J Mach Learn Res 7:1531–1565
22. Varma M, Babu BR (2009) More generality in efficient multiple kernel learning. In: ICML, pp 1065–1072
23. Vishwanathan SVN, Sun Z, Theera-Ampornpunt N, Varma M (December 2010) Multiple kernel learning and the SMO algorithm. In: NIPS
24. Zien A, Ong CS (2007) Multiclass multiple kernel learning. In: ICML, pp 1191–1198