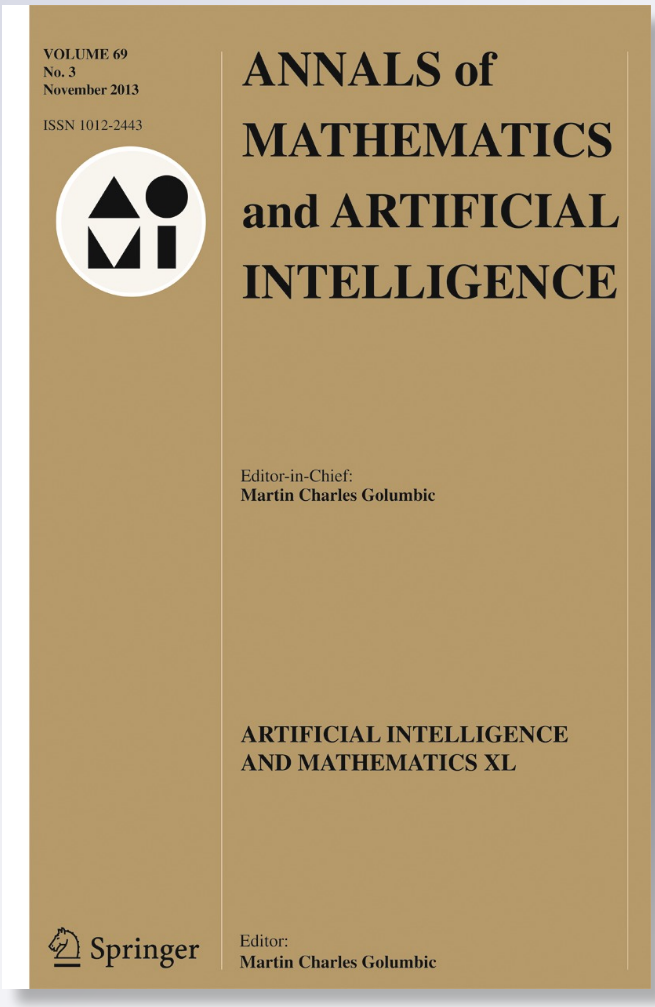
Annals of Mathematics and Artificial Intelligence

ISSN 1012-2443
Volume 69
Number 3

Ann Math Artif Intell (2014) 69:225-241
DOI 10.1007/s10472-013-9331-4



Ann Math Artif Intell (2013) 69:225–241
DOI 10.1007/s10472-013-9331-4

Analytic center cutting plane method for multiple kernel learning

Sharon Wulff · Cheng Soon Ong

Published online: 5 September 2013

Abstract Multiple Kernel Learning (MKL) is a popular generalization of kernel methods which allows the practitioner to optimize over convex combinations of kernels. We observe that many recent MKL solutions can be cast in the framework of oracle based optimization, and show that they vary in terms of query point generation. The popularity of such methods is because the oracle can fortuitously be implemented as a support vector machine. Motivated by the success of centering approaches in interior point methods, we propose a new approach to optimize the MKL objective based on the analytic center cutting plane method (accpm). Our experimental results show that accpm outperforms state of the art in terms of rate of convergence and robustness. Further analysis sheds some light as to why MKL may not always improve classification accuracy over naive solutions.

Keywords Multiple kernel learning · accpm · Oracle methods · Machine learning

Mathematics Subject Classification (2010) 68T05

1 Introduction

Kernel methods, for example the support vector machine (SVM), are a class of algorithms that consider only the similarity between examples [1]. A kernel function

S. Wulff
Department of Computer Science, ETH, Zürich, Switzerland
e-mail: sharon.wulff@inf.ethz.ch

C. S. Ong (✉)
NICTA, The University of Melbourne, Melbourne, Australia
e-mail: chengsoon.ong@unimelb.edu.au

k implicitly maps examples $\mathbf{x}$ to a feature space given by a feature map Φ via the identity $k(\mathbf{x}_i, \mathbf{x}_j) = \langle\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle$. It is often unclear what the most suitable kernel for the task at hand is, and hence the user may wish to combine several possible kernels. One problem with simply adding kernels is that using uniform weights is possibly not optimal. An extreme example is the case that one kernel is not correlated with the labels at all, in such cases giving it positive weight just adds noise [2]. Multiple kernel learning (MKL) is a way of optimizing kernel weights while training the SVM. In addition to leading to good classification accuracies, MKL can also be useful for identifying relevant and meaningful features [2–5]. MKL finds a convex combination of kernels [6, 7], that is to find a classifier

$$f_{\mathbf{w},b,\boldsymbol{\beta}}(\mathbf{x}, y) = \sum_{k=1}^{p} \beta_k \langle \mathbf{w}_k, \Phi_k(x)\rangle + b$$

where β are the kernel weights corresponding to the p kernels, and $\mathbf{w}$, b are the SVM parameters. The MKL problem was first formulated as a semidefinite programming SDP problem [6]. One can exploit the known structure of the MKL problem to speed up the optimization, for example using an SMO-like approach [8]. However, this requires a full reimplementation of the solver. MKL has also been recently shown to be equivalent to group LASSO [9].

Recently, leveraging on the existence of efficient software for solving the SVM optimization problem, a semi-infinite linear programming (SILP) approach was developed in [7]. Their solution is based on the concept of cutting planes. Cutting planes methods alternate between choosing query points and calling an oracle. Given a query point, the oracle returns halfspaces to be included in the current set of constraints, forming the feasible set. In the context of MKL, the oracle is an SVM solver and the query points are kernel weights. One criticism of the SILP is that it requires many cutting planes (and hence calls to the SVM) before convergence. This gave rise to a subgradient based approach [10] and a bundle method [11].

Our contributions in this paper are as follows. We review MKL optimization in an oracle based optimization framework and demonstrate that they are essentially different ways of generating query points. We propose the use of the analytic center as a query point since it would in expectation halve the hypothesis space. We conduct experiments comparing the performance of several MKL solutions on UCI data. We show that our more“regularized” approach often requires fewer iterations, and is more robust to variations in data. The experimental results are followed by a discussion about the correlation between the MKL objective and test accuracy.

2 Oracle based methods

Oracle based methods are widely used for solving integer programming problems. Here we focus on using oracle based methods for solving convex optimization problems [12]. The goal of oracle based algorithms is to find a point in a convex set Z, or to determine if the set is empty (refer to Algorithm 1 for a pseudocode description). In an optimization problem, Z is the set of ε-suboptimal points. The method does not assume any direct access to the description of Z, such as the objective and constraint functions, except through an oracle. Instead the method generates a query point q (Section 2.2) which upon satisfying $q \notin Z$ is passed to the oracle (Section 2.1). The oracle returns a hyperplane which separates q from the set Z. This hyperplane is called a "cut" since it eliminates a halfspace from our search, which is the reason oracle based methods are also known as cutting plane methods.

2.1 Oracles and cuts

For a convex optimization problem with m constraints,

$$\begin{array}{ll} \min_z & f_0(z) \\ \text{s.t.} & f_i(z) \leqslant 0, \qquad i = 1, \dots, m \end{array} \tag{1}$$

where $f_0, \dots, f_m$ are convex and differentiable, the target set Z is the optimal (or ε-optimal) set. Given a query point q, the oracle first checks for feasibility. If q is not feasible, then we choose one of the violated constraints $f_j(q) > 0$ and form a cut

$$f_j(q) + \nabla f_j(q)^\top (z - q) \leqslant 0 \tag{2}$$

This cut (2) is called a *feasibility cut* for the problem (1) since it cuts away a halfspace of points known to be infeasible (since they violate the jth constraint). If q is feasible, we construct the cutting plane

$$\nabla f_0(q)^\top (z - q) \leqslant 0, \tag{3}$$

which is called the *objective cut* for (1). This cuts out the halfspace

$$\left\{z \mid \nabla f_0(q)^\top (z - q) > 0\right\}$$

since all such points have an objective value larger than $f_0(q)$ and hence cannot be optimal. If q is feasible and $\nabla f_0(q) = 0$ then q is optimal. In general, for non-differentiable problems, the gradients $\nabla f_j(z)$ can be replaced by subgradients.

Algorithm 1 Cutting plane algorithm for optimization

```
Require: an initial polyhedron $\mathcal{P}_0$ containing $Z$.
  $t = 0$
  repeat
    Generate a query point $q^{(t+1)}$ in $\mathcal{P}_t$
    Query the oracle at $q^{(t+1)}$,
    Oracle returns a cutting plane $a_{t+1}^\top z \leqslant b_{t+1}$.
    Update the constraints:
      $\mathcal{P}_{t+1} = \mathcal{P}_t \cap \{z | a_{t+1}^\top z \leqslant b_{t+1}\}$.
    $t = t + 1$
  until convergence or $\mathcal{P}_{t+1} = \emptyset$
```

2.2 Generating query points

In principle, we would like the query point $q^{(t+1)}$ corresponding to the current polyhedron $\mathcal{P}_t$ (containing the optimal set Z) to be generated such that the resulting cut reduces the size of $\mathcal{P}_{t+1}$ as much as possible. When querying the oracle with $q^{(t+1)}$, we do not know in which direction the generated cut will exclude, but we do know that $q^{(t+1)}$ will be in the excluded halfspace. One approach is to greedily use the vertex of the current polytope which minimizes the objective, leading to the following method.

2.2.1 Method of Kelley–Cheney–Goldstein

For the optimization problem (1), the piecewise linear function

$$f^{(t)}(z) = \max_{i \leqslant t} f_0(z_i) + \nabla f_0(z_i)^\top (z - z_i)$$

is a lower approximation to $f_0(z)$. The next query point $q^{(t+1)}$ is found by solving

$$\begin{aligned} &\min \theta \\ &\text{s.t.} \;\; \theta \geqslant f_0(q_i) + \nabla f_0(q_i)^\top (z - q_i), \quad \forall i \leqslant t \\ &\qquad A_t z \leqslant b_t, \end{aligned} \tag{4}$$

where A_t, b_t are the set of existing cutting planes which define the current polyhedron $\mathcal{P}_t$. In the rest of the paper we refer to the above method as the Kelley–Cheney–Goldstein method (KCG).

Using $\frac{\nabla f_0(q_i)}{\|\nabla f_0(q_i)\|}$ instead of $\nabla f_0(q_i)$ in (4), results in finding the center of the largest sphere. This variant of (4) is called the Chebyshev center method. This modification, where the gradients are scaled to unit length, has significantly better convergence properties [12]. This already shows the power of centering, which is exploited by the following method.

2.2.2 Analytic center cutting plane method

The analytic center is a concept which has been popularized by interior point methods. Given a constraint $a_i^\top z \leqslant b_i$, define the slack $s_i \in \mathbb{R}$ as $s_i = b_i - a_i^\top z$, that is, s_i is a measure of how close or how far the current solution is from the constraint. An interior point of the feasible set is a point for which all the slacks are strictly positive. The analytic center is defined as the unique maximizer of the function $f(s) = \prod_{i=1}^{t} s_i$ where $s \in \mathbb{R}^t$ is the vector of slacks of the current set of constraints $\{a_i^\top z \leqslant b_i, i = 1, \ldots, t\}$. The geometrical interpretation of the analytic center is the point that maximizes the product of distances to all of the faces of the polytope. Maximizing the product of the slacks (e.g. instead of sum), ensures that every slack is strictly positive. We can rewrite the analytic center as

$$\operatorname*{argmax}_z f(s) = \operatorname*{argmax}_z \prod_{i=1}^{t} s_i = \operatorname*{argmax}_z \sum_{i=1}^{t} \log(b_i - a_i^\top z) \tag{5}$$

This function is also known as the logarithmic barrier, and its unique maximizer can be efficiently found using Newton iterations [12].

Going back to the cutting planes framework, computing a point which is as far as possible from the border of the feasible set, ensures that in every iteration the resulting cut yields a substantial reduction of the size of the set. Theoretical analysis of convergence in terms of oracle calls has been shown in [12].

3 Multiple kernel learning

In this section, we briefly review MKL and derive the oracle function. We detail our approach of query point generation using the analytic center and then review recent MKL approaches in the framework of oracle based methods.

3.1 Review of multiple kernel learning

We follow the setting of finding a convex combination of kernels that performs well in a SVM binary classification task [6, 7, 13], that is for a given training dataset $\{(x_i, y_i)\}_{i=1,\dots,n}$, find a classifier

$$f_{\mathbf{w},b,\boldsymbol{\beta}}(x) = \sum_{k=1}^{p} \beta_k \langle \mathbf{w}_k, \Phi_k(x)\rangle + b$$

where the kernel weights $\boldsymbol{\beta}$ and the SVM weights $\mathbf{w}$, b are found by optimizing[1]

$$\begin{aligned} &\min_{\boldsymbol{\beta},\mathbf{w},b,\xi} \frac{1}{2}\left(\sum_{k=1}^{p} \beta_k \|\mathbf{w}_k\|\right)^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.t. } \forall i: \ y_i \sum_{k=1}^{p} \beta_k \langle \mathbf{w}_k, \Phi_k(x)\rangle + b \geqslant 1 - \xi_i \text{ and } \xi_i \geq 0. \end{aligned} \tag{6}$$

In this section, the indices i, j are used to range over the number of data examples n. The p mixing coefficients $\boldsymbol{\beta}$ (indexed by k) should reflect the utility of the respective feature map for the classification task, and are normalized to be on the simplex, i.e.,

$$\boldsymbol{\beta} \in \Delta^p := \left\{ \boldsymbol{\beta} \,\middle|\, \sum_{k=1}^{p} \beta_k = 1, \forall k: \beta_k \geqslant 0 \right\},$$

giving them the flavor of probabilities. This $\mathcal{L}_1$ regularizer on $\boldsymbol{\beta}$ promotes sparsity, and hence we are trying to select a subset of kernels [9]. We refer to (6) as the primal problem. The dual formulation of the above problem is given by [6, 10] a quadratically constrained quadratic program (QCQP),

$$\begin{aligned} &\min_{\boldsymbol{\alpha},\gamma} \gamma - \sum_i \alpha_i \\ &\text{s.t. } \ \forall k: \gamma \geqslant \frac{1}{2}\|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 \\ &\qquad \forall i: 0 \leqslant \alpha_i \leqslant C \\ &\qquad \textstyle\sum_i y_i\alpha_i = 0, \end{aligned} \tag{7}$$

where the margin term for the kth kernel is given by

$$\|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 = \sum_{i,j} \alpha_i\alpha_j y_i y_j k_k(\mathbf{x}_i, \mathbf{x}_j).$$

[1]Equation (6) is not identical to the original formulation in [6, 7], but has been shown to be equivalent [13].

Following [7], we can move the sum of alphas into the constraints by changing $\gamma' = \gamma - \sum_i \alpha_i$, and then convert the QCQP (7) by a second (partial, only w.r.t γ) dualization. The derivation w.r.t γ recovers the simplex constraint on the kernel coefficients $\boldsymbol{\beta}$.

$$\begin{aligned} &\max_{\boldsymbol{\beta}\in\Delta^p} \min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_k \beta_k \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 - \sum_i \alpha_i \\ &\text{s.t.} \qquad \forall i : 0 \leqslant \alpha_i \leqslant C \\ &\qquad\qquad \textstyle\sum_i y_i \alpha_i = 0. \end{aligned} \tag{8}$$

We further derive our solution based on (8).

3.2 MKL solution in an oracle based framework

To remain consistent with (1) we change the objective sign, and define the following function of the kernel coefficients

$$\begin{aligned} g_0(\boldsymbol{\beta}) = \max_{\boldsymbol{\alpha}} &\sum_i \alpha_i - \frac{1}{2}\sum_k \beta_k \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 \\ &\text{s.t.} \forall i : 0 \leqslant \alpha_i \leqslant C \\ &\textstyle\sum_i y_i \alpha_i = 0. \end{aligned} \tag{9}$$

Using the above definition, (8) becomes

$$\begin{aligned} &\min_{\boldsymbol{\beta}} g_0(\boldsymbol{\beta}) \\ &\text{s.t.} \ \ \boldsymbol{\beta} \in \Delta^p \end{aligned} \tag{10}$$

Observe that for a given $\tilde{\boldsymbol{\beta}}$, evaluating $g_0(\tilde{\boldsymbol{\beta}})$ amounts to an SVM call [7]. That is, $g_0(\boldsymbol{\beta})$ is exactly the dual formulation of the SVM, where the kernel is a linear combination of the p kernels weighted by the coefficients $\boldsymbol{\beta}$. This allows us to use an SVM solver as the oracle. The constraints on the dual variables $\boldsymbol{\alpha}$ in (8) are automatically satisfied by the SVM. Furthermore, since we use a feasible $\tilde{\boldsymbol{\beta}}$ for each SVM call, we only ever perform objective cuts in the oracle based framework.

3.2.1 ACCPM for MKL

Motivated by the properties of the analytic center discussed in Section 2.2.2, we propose to use it as the query point generation criteria.

Let $\boldsymbol{\beta}^l, \boldsymbol{\alpha}^l$ where $l = 1, \ldots, t-1$ denote the previous query points and the corresponding SVM solver solutions. At iteration t, the analytic center (5) is given by

$$\boldsymbol{\beta}^t = \operatorname*{argmax}_{\boldsymbol{\beta}\in\Delta^p} \sum_{l=1}^{t-1} \log \frac{1}{2}\left(\sum_k \beta_k^l \left\|\mathbf{w}_k(\boldsymbol{\alpha}^l)\right\|^2 - \sum_k \beta_k \left\|\mathbf{w}_k(\boldsymbol{\alpha}^l)\right\|^2\right) \tag{11}$$

The oracle then solves the single kernel SVM problem with $\boldsymbol{\beta}^t$ as the weighting coefficients, to obtain $\boldsymbol{\alpha}^t$. Next, the oracle computes the gradient of $g_0(\boldsymbol{\beta})$, given by

$-\frac{1}{2}\|\mathbf{w}_k(\boldsymbol{\alpha})\|^2$ with $\boldsymbol{\alpha} = \boldsymbol{\alpha}^t$. Note that since we restrict the generation of $\boldsymbol{\beta}$'s to $\boldsymbol{\beta} \in \Delta^p$, the oracle only ever returns objective cuts (3), hence the new cut is given by

$$\left[\frac{1}{2}\left\|\mathbf{w}_k(\boldsymbol{\alpha}^t)\right\|^2\right]^\top (\boldsymbol{\beta} - \boldsymbol{\beta}^t) \leqslant 0. \tag{12}$$

Where the first term is a vector of the same length as $\boldsymbol{\beta}$ with values $\frac{1}{2}\|\mathbf{w}_k(\boldsymbol{\alpha})\|^2$.

A pseudocode description of the analytic center cutting planes algorithm for multiple kernel learning is given in Algorithm 2.

Algorithm 2 ACCPM for multiple kernel learning

```
Require: A set of $p$ kernels, labeled sample $\{(\mathbf{x}_i, y_i)\}$
  initialize $\boldsymbol{\beta}^0 = \frac{1}{p}\mathbf{1}$, $t = 0$.
  repeat
    Solve $\boldsymbol{\alpha}^t = g_0(\boldsymbol{\beta}^t)$, an SVM call with $\boldsymbol{\beta}^t$ as the kernel coefficients.
    Compute the gradient as $-\frac{1}{2}\left\|\mathbf{w}_k(\boldsymbol{\alpha}^t)\right\|^2$ for each kernel $k = 1, \ldots, p$
    and the objective cut given in (12).
    Solve $\boldsymbol{\beta}^{t+1}$ the analytic center from (11).
    t = t+1
  until Duality gap (Section 3.2.3) is smaller than $\epsilon$.
```

3.2.2 Related work

In this section we review recent MKL work. We follow the order of presentation in [12], Section 3.2 At iteration t, the lower approximation of g_0 (9), calculated by the KCG method (Section 2.2.1) is

$$g_t(\boldsymbol{\beta}) = \max_{l \le t} g_0(\boldsymbol{\beta}^l) + \nabla g_0(\boldsymbol{\beta}^l)^T (\boldsymbol{\beta} - \boldsymbol{\beta}^l) \tag{13}$$

where $\boldsymbol{\beta}^l$ is the vector of kernel coefficients chosen in the lth iteration. Let $\boldsymbol{\alpha}^l$ denote the maximum over $\boldsymbol{\alpha}$ corresponding to $\boldsymbol{\beta}^l$ i.e.,

$$\boldsymbol{\alpha}^l := \operatorname*{argmax}_{\boldsymbol{\alpha}} \sum_i \alpha_i - \frac{1}{2}\sum_k \beta_k^l \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2$$

Using the above definitions, one can verify that $g_t(\boldsymbol{\beta})$ can be rewritten as

$$g_t(\boldsymbol{\beta}) = \max_{l \le t} \sum_i \alpha_i^l - \frac{1}{2}\sum_k \beta_k \left\|\mathbf{w}_k(\boldsymbol{\alpha}^l)\right\|^2 \tag{14}$$

Therefore, the KCG query points are found by optimizing

$$\boldsymbol{\beta}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\beta} \in \Delta^p} g_t(\boldsymbol{\beta}) \tag{15}$$

The KCG method was applied to MKL in [5, 7] where (10) was further transformed into a semi-infinite linear programming (SILP).

A subgradient based approach which use the gradient with respect to the kernel weights was suggested in [10] (simpleMKL algorithm). One difficulty with subgradient methods is determining the optimal step size to be taken in the direction of the subgradient. They use a one dimensional line search, which involves several calls to the SVM solver. The subgradient method is memoryless, it does not utilize the gradient computed in previous iterations. Since previous search directions could be useful in finding a new search direction, a bundle method which projects the SILP solution to the level set was proposed [11]. The extended level set bundle method has been shown to converge faster than subgradient.

The level-set method chooses the projection of the previous query point into a level set that is a weighted combination of the piece-wise linear lower approximation of $g_t(\boldsymbol{\beta})$ and the tightest upper bound discovered so far.

$$\boldsymbol{\beta}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\beta}\in\Delta^p} \left\{ \left\| \boldsymbol{\beta} - \boldsymbol{\beta}^t \right\|^2 : g_t(\boldsymbol{\beta}) \le L_t \right\}$$

$$L_t = \lambda \bar{f}^t + (1-\lambda)\underline{f}^t \tag{16}$$

$\bar{f}^t := \min_{1\le l\le t} g_0(\boldsymbol{\beta}^l)$	best upper bound discovered so far.
$\underline{f}^t := \operatorname*{argmin}_{\boldsymbol{\beta}\in\Delta^p} g_t(\boldsymbol{\beta})$	is the point chosen by KCG.

3.2.3 Duality gap

The convergence of the duality gap is a natural stopping criteria for convex optimization. Recall the primal MKL formulation in (6), for a specific value of the coefficients vector $\boldsymbol{\beta}$ and the corresponding dual variables $\boldsymbol{\alpha}$, the primal objective is given by

$$\frac{1}{2}\left(\sum_{k=1}^{p} \boldsymbol{\beta}_k \sum_{i,j} \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2\right) + C\sum_{i=1}^{n} \xi_i \tag{17}$$

The slack variables ξ_i can be retrieved from

$$\xi_i = \max\left(0, 1 - y_i\left(\sum_{k=1}^{p} \boldsymbol{\beta}_k \sum_j \boldsymbol{\alpha}_j k_k(\mathbf{x}_i, \mathbf{x}_j)\right)\right)$$

The dual objective is given in (7)

$$\max_k \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 - \sum_i \alpha_i \tag{18}$$

Hence the duality-gap corresponds to: (17) – (18).

3.3 Implementation issues

One of the advantages of casting MKL as an oracle based method, lies in the resulting modular solution structure of an oracle-SVM component and a query point generator. Our implementation is a framework written in Python, having interfaces defining an oracle and a query point generator, such that the actual implementation can be easily replaced.

We used shogun[2] as the oracle-SVM solver and for computing the kernels. In this paper we only show accpm in the settings of binary classification. In principle, since we use shogun as the SVM solver, we would easily be able to solve other learning tasks supported by shogun. We used the software package OBOE[3] for finding the analytic center. The query points that OBOE selects are minimizers of (5) plus a proximal term. The optimization of the logarithmic barrier function is done using an infeasible start Newton method. More details of the implementation can be found in [14]. We used SWIG[4] for creating a Python interface to OBOE (which is written in C).

For comparison purposes we implemented the Kelley–Cheney–Goldstein query point generation in our framework. We used mosek's[5] python interface to solve the linear program.

The software we used is available on the author's homepage.

4 Computational results

In this section we benchmark recent MKL implementations using some UCI datasets and compare their performance. We then empirically test some more general properties of MKL optimization.

The benchmarked MKL methods are the simpleMKL [10], the extended level set method [11] (level-set) and our oracle based software with two different query point generators. The analytic center (accpm) and our KCG method implementations (kcg). The SILP [7] algorithm is another MKL solution based on the KCG method, but we do not compare directly with this implementation.

As a point of reference for non MKL based approaches, we compare with a single kernel SVM classifier. The kernel in use is the averaged sum of the candidate MKL kernels, equivalent to uniform $\boldsymbol{\beta}$ weights. This solver is denoted here as "average".

4.1 Experiments overview

We conduct our experiments on UCI repository datasets and consider three different aspects of the solutions.

1. **Performance analysis** We compare the performance of the different implementations. We measure the number of SVM solver calls, the accuracy of the solution, the actual running-time of computing a single MKL solution and the number of kernels which are assigned non-zero weights. A similar experiment is described in [10], where the performance of the SILP and the simpleMKL implementations are compared. The experiment was conducted again by [11], where the extended level set method was introduced. To be consistent with these experiments, we follow the same general settings and test the performance on the same UCI datasets.

[2] http://www.shogun-toolbox.org

[3] https://projects.coin-or.org/OBOE

[4] http://www.swig.org

[5] http://www.mosek.com/

2. **Duality-gap convergence** All of the MKL methods we consider incorporate an iterative procedure while seeking the optimal solution. In every iteration a new kernel coefficients vector is computed. For each such intermediate solution we store the corresponding duality gap and compare the algorithm's convergence behavior.
3. **Accuracy vs. duality-gap** We look at the relation between the accuracy and the duality gap at points gathered up during the run of the algorithms. While achieving high accuracy is a measure of generalization ability and is desirable in every supervised learning task, the duality gap is the mathematical objective which is actually being optimized. To our best knowledge the connection between the two measures has never been studied before.

4.2 Experimental settings

We ran the five different implementations on eight UCI repository datasets: wbpc, ionosphere, sonar, bupa, pima, vote, heart and wdpc (the first 5 were used by [10] and the rest added in [11]). Similarly to [10] the candidate kernels were: Gaussian kernels with 10 different width values σ and Polynomial kernels of degree 1 to 3. All kernels were computed with respect to all features and to every single feature separately. Thus the total number of kernels per dataset is $13(d+1)$, where d is the number of features. We determined the widths of the Gaussian kernels by taking the Euclidean distances between the training data points. We sorted the pairwise distances in an increasing order and selected 10 kernel widths which are uniformly spaced on a log scale between the 10 % and 90 % quantiles of this range. These values reflect the data spread and are therefore potentially good candidate kernels (note that the Gaussian kernel widths are different from [10], where the widths are fixed for all datasets).

As is common practice [7, 10, 11] all kernel matrices were normalized to have unit trace and are computed prior to the run of the algorithm. Normalizing the kernel matrices is necessary to prevent kernel matrices with larger values from artificially getting more weight. Intuitively, each kernel can be seen as a different measure of similarity. Having a larger similarity between the same elements lead to a larger margin and is therefore preferred over a smaller one. An absurd case would be to compare two kernel matrices, one which is just a scaled version of the other. The C hyperparameter was set to 100, this value is the same as in [10] and [11]. The optimality of this value in terms of solution accuracy was verified by means of cross-validation. As a starting point all of the candidate kernels were assigned equal weights. Training examples were normalized to zero mean and unit variance. The evaluation technique we used was 20 random (70 %, 30 %) splits of the data.

The stopping criteria we used was the convergence of the duality gap beyond a threshold of 5e-03, or the number of iterations exceeds 500. Ideally the stopping criteria would have been much lower, however the runs of level-set and simpleMKL did not converge below this value on several training-set permutations of some of the datasets. This is visible in Fig. 1 where the curves of the duality-gap along the iterations of the algorithms are presented. A slightly higher threshold value was used in [10] and [11].

In the performance comparison experiment, we report the accuracy, the run-time, the number of SVM solver calls and the number of chosen kernels. The accuracy is the average percentage of correct predictions made on the 20 test-sets. The

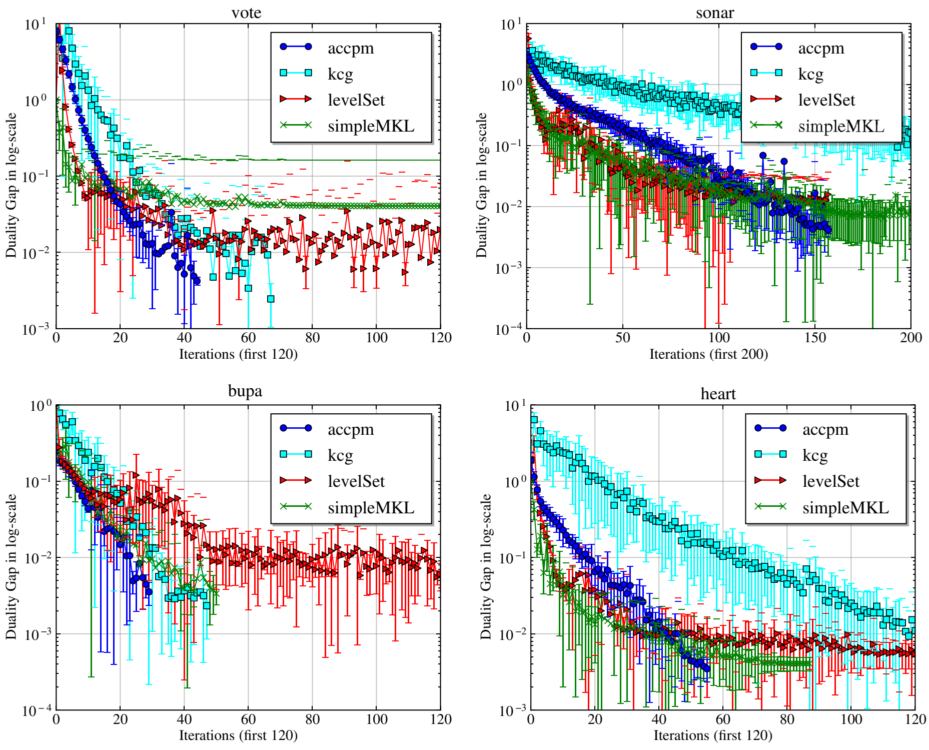


Fig. 1 Duality-gap convergence along the iterations for various datasets. The points (*bars*) are computed as the average (and standard deviation) value over the 20 data splits of the duality gap at a particular iteration

running time is the average running time in seconds until convergence, without the accumulated time of the SVM solver. We subtract the SVM solver time in the comparison since we use different implementations which use different SVM solvers. In fact, since the datasets used here are rather small, the SVM solver run-time for most of the implementation is negligible. Often MKL is used in applications where the datasets are much larger and the SVM solver time becomes the dominant run-time component. We therefore compare the average number of SVM calls used by the algorithms to reach a single solution. The reported number of kernels is the number of kernels assigned non-vanishing weights in the end of the run. These are also the kernels that take part in prediction.

All of the implementations were run on an Intel(R) Core(TM)2 2.83GHz computer running Redhat Linux.

4.3 Results

4.3.1 Performance analysis

Table 1 shows the performance results obtained by the different implementations. For each dataset, n and p, denote the number of examples and number of kernels respectively.

Table 1 Performance comparison of the MKL solutions accpm, level-set [11], simpleMKL [10] and kcg and the non-MKL, average, on UCI datasets using random data splits

Algorithm	# SVM calls	Time (s)	# Kernels	# SVM calls	Time (s)	# Kernels
	wpbc, $n = 138$, $p = 442$			pima, $n = 537$, $p = 117$		
accpm	55.1 ± 4.1	8.4 ± 0.8	12.9 ± 2.2	28.6 ± 2.6	6.1 ± 0.6	12.7 ± 3.2
Level-set	153.6 ± 25.4	40.4 ± 11.7	13.9 ± 1.7	58.6 ± 8.6	27.5 ± 4.5	11.7 ± 1.5
simpleMKL	788.2 ± 201.3	17.4 ± 1.3	13.5 ± 2.1	466.9 ± 128.8	49.8 ± 7.6	11.2 ± 1.9
kcg	109.1 ± 25.7	13.9 ± 5.8	10.3 ± 1.7	66.2 ± 14.6	13.5 ± 3.1	8.2 ± 1.1
Average	1.0	0.04 ± 0.0	442	1.0	0.22 ± 0.0	117
	Ionosphere, $n = 245$, $p = 455$			Vote, $n = 304$, $p = 221$		
accpm	100.4 ± 9.4	58.5 ± 17.0	15.7 ± 2.6	34.75 ± 2.49	2.7 ± 0.4	14.7 ± 5
Level-set	73.3 ± 31.9	25.9 ± 18.5	18.4 ± 1.8	115.3 ± 84.3	51.2 ± 92.4	10 ± 2.5
simpleMKL	1273 ± 420.6	56.0 ± 5.9	19 ± 2.3	4792 ± 7708	87.2 ± 111.7	9.2 ± 3.2
kcg	255.5 ± 54.7	108.6 ± 83.1	14.8 ± 2.4	46.6 ± 12.4	2.9 ± 0.9	5.2 ± 2.1
Average	1.0	00.14 ± 0.0	455	1.0	0.06 ± 0.0	221
	Sonar, $n = 145$, $p = 793$			Heart, $n = 189$, $p = 182$		
accpm	133.4 ± 14.3	180.1 ± 68.6	21.7 ± 1.7	46.2 ± 4.7	3.1 ± 0.5	11.3 ± 1.3
Level-set	127.5 ± 93	55.2 ± 50.5	29.2 ± 1.9	81.3 ± 34.7	8.8 ± 6.6	14.7 ± 1.6
simpleMKL	4464 ± 2211	81.2 ± 15.4	21.9 ± 2	650.2 ± 436.1	7.4 ± 2.2	13.7 ± 1.5
kcg	450.4 ± 48.3	602.7 ± 222.4	21.2 ± 2.2	116.5 ± 29.6	10.2 ± 4.7	10.5 ± 1.6
Average	1.0	0.14 ± 0.0	793	1.0	0.04 ± 0.0	182
	Bupa, $n = 241$, $p = 91$			wdbc, $n = 398$, $p = 403$		
accpm	25.2 ± 2.8	1.4 ± 0.1	6.6 ± 1.7	72.4 ± 4.1	24.4 ± 3.7	13.3 ± 1.4
Level-set	123.9 ± 19.4	13.8 ± 3.5	7.7 ± 1.3	115.7 ± 38.1	94.9 ± 38.8	9.8 ± 1.5
simpleMKL	332.7 ± 142.6	4.3 ± 1.4	7.6 ± 1.8	1843.8 ± 1572.6	178.2 ± 34	10.5 ± 1.3
kcg	33 ± 7.2	2.1 ± 0.4	5.7 ± 0.8	119.6 ± 14.3	28.8 ± 4.2	10.4 ± 1.1
Average	1.0	0.05 ± 0.0	91	1.0	0.23 ± 0.01	403

Time is the average running time in seconds, number of SVM calls is the average number of calls made during one run and the number of kernels is the number of kernels assigned non-vanishing weights in the end of the run

In term of number of SVM solver calls, accpm achieves significantly better results on most datasets. On 6 out of 8 datasets accpm uses the lowest number of SVM calls, on the remaining 2 datasets accpm is either competitive or second to level-set. The deviations in the number of solver calls shows that accpm is more robust with respect to different data splits which is the cause of variation in the results of the other methods. A very likely explanation is the centering approach.

We observe that the number of SVM solver calls made by simpleMKL is significantly higher compared to the other methods. This can be attributed to the fact that simpleMKL performs a line search, which involves several calls to the SVM solver during the computation of a single new solution (update of the weights). In all other implementations there is a one-to-one correspondence between an SVM call and kernel coefficients update. The SVM calls invoked during the line search can become cheap using warm start. For this reason as well as the fact that the SVM implementations are different, we report the running time without the SVM computations factor.

Table 2 Accuracy results of the MKL methods vs. the average kernel

Dataset	MKL accuracy (%)	"Average" accuracy (%)
wpbc	79.4 ± 6.4	78.5 ± 5.5
ionosphere	93.1 ± 1.9	82.9 ± 2.9
sonar	77.8 ± 5.5	68.8 ± 5.2
bupa	66.7 ± 3.4	58.8 ± 2.7
pima	76.1 ± 2.4	75.0 ± 2.5
vote	95.6 ± 1.8	94.2 ± 2.1
heart	83.6 ± 3.4	84.3 ± 3.1
wdbc	96.6 ± 1.2	93.9 ± 1.6

The numbers in the table represent the accuracy of accpm, however level-set, simpleMKL and kcg method, achieve the same accuracy within the error range

Comparing the running time without the SVM computation time, accpm is the fastest on 5 out of 8 datasets and is as fast as the fastest solution, kcg, on an additional one. On the remaining 2 datasets level-set is the fastest algorithm. We will revisit this fact later when considering the convergence behaviour.

The kcg method assigns less kernels with non-vanishing coefficients on most datasets.

The accuracy results are presented in Table 2. All of the MKL solutions achieved the same accuracy within the error range, hence the table only shows results of accpm and the non-MKL solution "average".

Considering the accuracy results, it is clear that for some datasets e.g. wbpc, pima and heart, learning the kernel weights has no advantage over simply adding the candidate kernels. In heart dataset the MKL approach is even slightly worse than the averaging one (although still within the error range). We further discuss this point when comparing the duality gap and accuracy.

4.3.2 Duality-gap convergence

We analyse the convergence behaviour of the various methods by comparing the average duality gap at each iteration. Figure 1 shows the curves of changes in duality-gap along the iterations of bupa, sonar, heart and vote.

The level-set method as well as simpleMKL, exhibit a sharp reduction in the duality gap in the beginning of the iterative procedure.

However in most datasets after some iterations the duality-gap remains almost at the same level or slightly increases, before it converges. It can be seen that a lower convergence threshold value than the one used here, will not be reached on some of data splits.

Accpm on the other hand convergences in a very steady manner. This can be further seen in the convergence plots of the remaining datasets which are given in the supplementary material. Again, this can be a result of choosing the center of the set, where in each iteration, the size of the remaining feasible set is expected to be reduced by roughly half. However, the point in which the two strategies cross, is sometimes in a region of already sufficiently small duality-gap, resulting in an advantage to a less regularized approach. For example in the curve depicting the run on sonar.

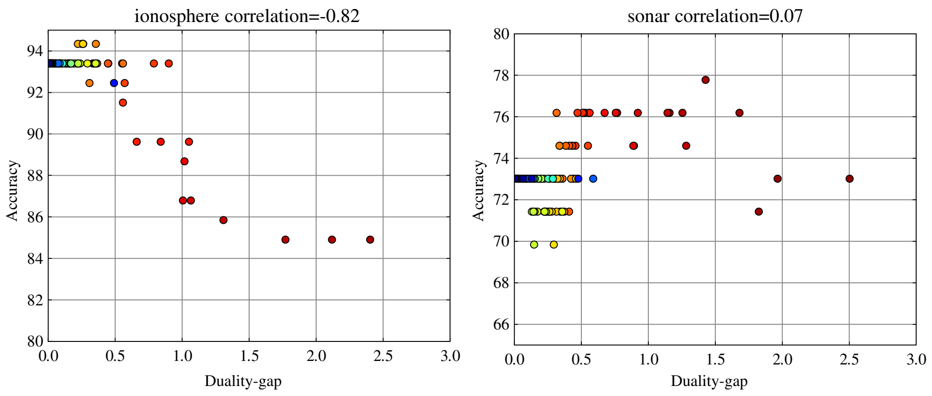


Fig. 2 Duality-gap vs. accuracy as generated by one run of accpm on ionosphere (*left side*) and sonar (*right side*)

4.3.3 Accuracy vs. duality-gap

The aim of this experiment was to test the degree to which optimizing for the MKL objective function yields good accuracy results. For each intermediate solution we computed the accuracy and the duality-gap. Plots of the resulting accuracy-duality-gap pairs of ionosphere and sonar are shown in Fig. 2. We choose to present these two datasets as they demonstrate different correlation trends. We also computed the average Pearson correlation scores, over 20 different data splits. The mean correlation values are shown in Table 3. The results differ between the methods as the intermediate solutions are different.

The results indicate that for some datasets there is no corresponding improvement in test accuracy for a reduction in duality gap. From the accpm column of Table 3, heart has the least (absolute) correlation, and indeed according to Table 2 heart performs better with the average kernel. The other two datasets that show small accuracy improvements with MKL (wpbc and pima) have also small absolute correlations. It is not the whole story, as sonar which has a slightly higher correlation, shows increase in accuracy with MKL.

The hope is that we can a priori (before running MKL) identify datasets where MKL does not improve accuracy. Further, we may wish to change the MKL formulation such that the objective function is a better guide to test accuracy.

Table 3 Mean Pearson correlation values between duality-gap and accuracy, on various UCI data sets

Data set	accpm	levelset	simpleMKL	kcg
wpbc	-0.25 ± 0.5	-0.3 ± 0.3	-0.24 ± 0.3	-0.33 ± 0.3
iono	-0.84 ± 0.0	-0.53 ± 0.1	-0.74 ± 0.1	-0.7 ± 0.0
sonar	-0.3 ± 0.3	0.1 ± 0.3	-0.32 ± 0.2	-0.49 ± 0.1
bupa	-0.74 ± 0.2	-0.57 ± 0.2	-0.62 ± 0.2	-0.63 ± 0.1
pima	-0.25 ± 0.5	-0.17 ± 0.5	-0.91 ± 0.1	-0.82 ± 0.2
vote	-0.56 ± 0.4	-0.01 ± 0.1	-0.79 ± 0.1	-0.78 ± 0.2
heart	-0.05 ± 0.4	0.03 ± 0.4	-0.69 ± 0.4	-0.76 ± 0.1
wdbc	-0.75 ± 0.2	-0.43 ± 0.2	-0.58 ± 0.1	-0.89 ± 0.1

4.4 Relevance of performance measures

An SVM solver call is a fundamental step in the run of all of the benchmarked MKL methods. The number of solver call is an important evaluation measure since this would be the dominant computational cost when the size of the training set is larger than the number of kernels in the linear combination, which is often the case in real applications. This is simply due to the fact that the SVM is quadratic in the number of examples whereas the query point generation only scales with the number of kernels. Hence one may use a more expensive approach to choose the query points without adversely affecting the total computational cost.

The running time can be useful in identifying trends in the run-time behavior of the algorithms. In our experiments however, it can be misleading since the sizes of the datasets are small which distorts the cost proportions and the compared methods use different SVM solvers. To compensate for this we report the running time without including the time taken by the SVM.

In this work we compare our method with several MKL solutions, which are considered "state of the art" in this rapidly progressing field. Two approaches can be taken when conducting such a comparison. One is reimplementing all the algorithms in the same framework, and the other is reusing published code. Each approach has its pros and cons in terms of providing the fairest comparison, this is not an easy tradeoff. We chose to use the original software of SimpleMKL and level-set as we believe it will be difficult to obtain an equally optimized code and wish to avoid using a poor implementation. The main disadvantage of this approach is that comparing the running-time is less meaningful.

5 Summary and future work

In general, oracle based methods can be used to solve cone programming problems which covers a large class of machine learning tasks. From an implementation viewpoint, what is required is a decomposition of the problem such that it is easy to implement the oracle or there is already an existing implementation.

We have shown the benefit of choosing a central point when using an alternating optimization method for multiple kernel learning. The experiments demonstrate that our more "regularized" approach often requires fewer iterations, and is more robust to variations in data. Further, empirically it has a smooth convergence curve, in contrast to previous methods. With the availability of software for computing both the oracle (shogun) and the analytic center (OBOE), we also demonstrate the synergies of software reuse and open source software.

In addition, our work offers some meaningful insights concerning MKL optimization. We believe that this line of research can potentially lead to means of characterizing data as such than can benefit from the use of kernel learning.

Appendix: Supplemental material

In the performance comparison experiment we report the number of SVM calls used by each of the compared solutions before reaching a single solution (see Table 1).

SimpleMKL performs subgradient descent and uses multiple SVM solver calls in each subgradient iteration. Therefore the number of SVM calls does not reflect the number of subgradient iterations used by simpleMKL.

Table 4 presents the number of subgradient iterations used by simpleMKL in the computation of a single solution for each of the benchmarked datasets.

Table 4 number of simpleMKL subgradient iterations

Dataset	# Subgradient iterations
wpbc	33.0 ± 15.5
Ionosphere	68.4 ± 19.4
Sonar	187.2 ± 87.3
Bupa	24.2 ± 11.7
Pima	39.1 ± 12.6
Vote	134.70 ± 184.43
Heart	40.35 ± 22.30
wdbc	79.30 ± 55.37

The plots in Fig. 3 show the changes in duality-gap along the iterations of accpm, level-set method, simpleMKL and kcg.

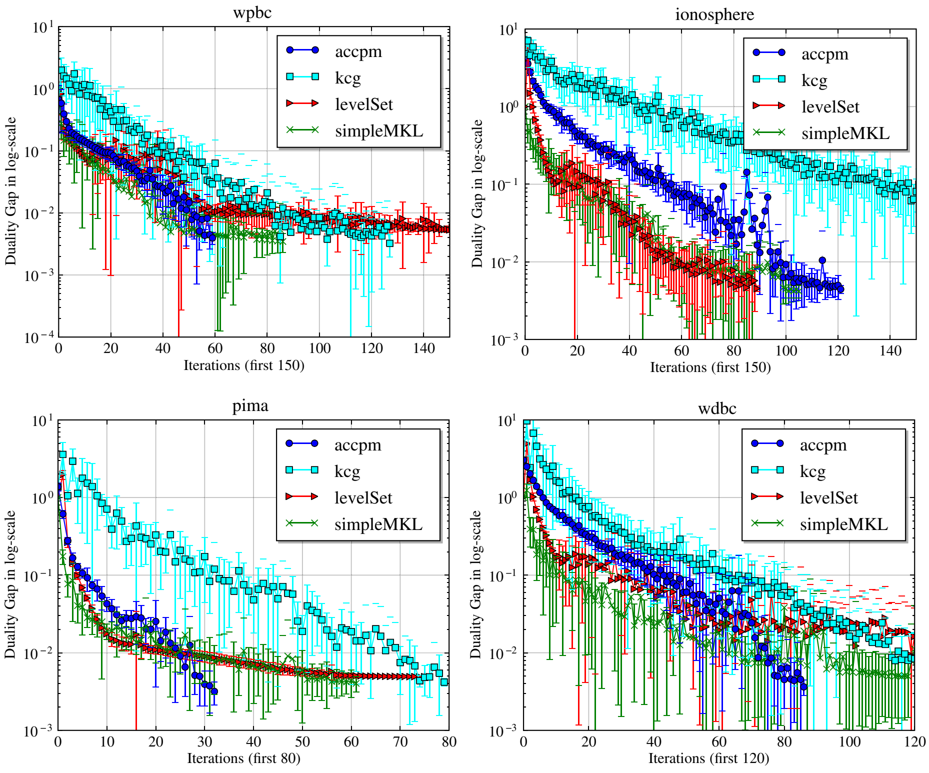


Fig. 3 Duality-gap vs. iterations on UCI datasets which were not included in the paper

References

1. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge, MA (2002)
2. Lanckriet, G., De Bie, T., Cristianini, N., Jordan, M.I., Stafford Noble, W.: A statistical framework for genomic data fusion. Bioinfomatics **20**(16), 2626–2635 (2004).
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.-P.: Protein function prediction via graph kernels. In: Proceedings of the International Conference on Intelligent Systems for Molecular Biology (2005)
4. Sonnenburg, S., Rätsch, G., Schäfer, C.: A general and efficient multiple kernel learning algorithm. In: Neural Information Processings Systems (2005)
5. Ong, C.S., Zien, A.: An automated combination of kernels for predicting protein subcellular localization. In: Crandall, K.A., Lagergren, J. (eds.) WABI, pp. 86–197. Springer (2008)
6. Lanckriet, G., Cristianini, N., Bartlett, P., El Ghaoui, L., Jordan, M.I.: Learning the kernel matrix with semi-definite programming. J. Mach. Learn. Res. **5**, 27–72 (2004).
7. Sonnenburg, S., Rätsch, G., Schäfer, C., Schölkopf, B.: Large scale multiple kernel learning. J. Mach. Learn. Res. **7**, 1531–1565 (2006)
8. Bach, F.R., Lanckriet, G.R.G., Jordan, M.I.: Multiple kernel learning, conic duality, and the SMO algorithm. In: ICML (2004)
9. Bach, F.R.: Consistency of the group lasso and multiple kernel learning. J. Mach. Learn. Res. **9**, 1179–1225 (2008)
10. Rakotomamonjy, A., Bach, F.R., Canu, S., Grandvalet, Y.: Simplemkl. J. Mach. Learn. Res. **9**, 2491–2521 (2008)
11. Xu, Z., Jin, R., King, I., Lyu, M.R.: An extended level method for efficient multiple kernel learning. In: NIPS (2008)
12. Vial, J., Goffin, J.: Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method. Optim. Methods Softw. **17**(5), 805–867 (2002)
13. Zien, A., Ong, C.S.: Multiclass multiple kernel learning. In: ICML (2007)
14. Babonneau, F., Beltran, C., Haurie, A., Tadonki, C., Vial, J.-P.: Proximal-ACCPM: a versatile oracle based optimization method. In: Kontoghiorghes, E.J., Gatu, C. (eds.) Optimisation, Econometric and Financial Analysis (2007)