A Randomized Mirror Descent Algorithm for Large Scale Multiple Kernel Learning

Arash Afkanpour AFKANPOU@UALBERTA.CA
András György GYORGY@UALBERTA.CA
Csaba Szepesvári SZEPESVA@UALBERTA.CA
Michael Bowling BOWLING@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8 Canada

Abstract

We consider the problem of simultaneously learning to linearly combine a very large number of kernels and learn a good predictor based on the learnt kernel. When the number of kernels d to be combined is very large, multiple kernel learning methods whose computational cost scales linearly in d are intractable. We propose a randomized version of the mirror descent algorithm to overcome this issue, under the objective of minimizing the group p-norm penalized empirical risk. The key to achieve the required exponential speed-up is the computationally efficient construction of low-variance estimates of the gradient. We propose importance sampling based estimates, and find that the ideal distribution samples a coordinate with a probability proportional to the magnitude of the corresponding gradient. We show that in the case of learning the coefficients of a polynomial kernel, the combinatorial structure of the base kernels to be combined allows sampling from this distribution in $O(\log(d))$ time, making the total computational cost of the method to achieve an ϵ-optimal solution to be $O(\log(d)/\epsilon^2)$, thereby allowing our method to operate for very large values of d. Experiments with simulated and real data confirm that the new algorithm is computationally more efficient than its state-of-the-art alternatives.

1. Introduction

We look into the computational challenge of finding a good predictor in a multiple kernel learning (MKL) setting where the number of kernels is very large. In particular, we are interested in cases where the base kernels come from a space with combinatorial structure and thus their number d could be exponentially large. Just like some previous works (e.g. Rakotomamonjy et al., 2008; Xu et al., 2008; Nath et al., 2009) we start with the approach that views the MKL problem as a nested, large scale convex optimization problem, where the first layer optimizes the weights of the kernels to be combined. More specifically, as the objective we minimize the group p-norm penalized empirical risk. However, as opposed to these works whose underlying iterative methods have a complexity of $\Omega(d)$ for just any one iteration, following (Nesterov, 2010; 2012; Shalev-Shwartz and Tewari, 2011; Richtárik and Takáč, 2011) we use a randomized coordinate descent method, which was effectively used in these works to decrease the per iteration complexity to $O(1)$. The role of randomization in our method is to use it to build an unbiased estimate of the gradient at the most recent iteration. The issue then is how the variance (and so the number of iterations required) scales with d. As opposed to the above mentioned works, in this paper we propose to make the distribution over the updated coordinate dependent on the history. We will argue that sampling from a distribution that is proportional to the magnitude of the gradient vector is desirable to keep the variance (actually, second moment) low and in fact we will show that there are interesting cases of MKL (in particular, the case of combining kernels coming from a polynomial family of kernels) when efficient sampling (i.e., sampling at a cost of $O(\log d)$) is feasible from this distribution. Then, the variance is controlled by the a priori weights put on the kernels, making it potentially independent of d. Under these favorable conditions (and in particular, for the polynomial kernel set with some specific prior weights), the complexity of the method as a function of d becomes logarithmic, which makes our MKL algorithm feasible even for large scale problems. This is to be contrasted

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

to the approach of Nesterov (2010; 2012) where a fixed distribution is used and where the *a priori* bounds on the method's convergence rate, and, hence, its computational cost to achieve a prescribed precision, will depend linearly on d (note that we are comparing upper bounds here, so the actual complexity could be smaller). Our algorithm is based on the mirror descent (or mirror descent) algorithm (similar to the work of Richtárik and Takáč (2011) who uses uniform distributions).

It is important to mention that there are algorithms designed to handle the case of infinitely many kernels, for example, the algorithms by Argyriou et al. (2005; 2006); Gehler and Nowozin (2008). However, these methods lack convergence rate guarantees, and, for example, the consistency for the method of Gehler and Nowozin (2008) works only for "small" d. The algorithm of Bach (2008), though practically very efficient, suffers from the same deficiency. A very interesting proposal by Cortes et al. (2009) considers learning to combine a large number of kernels and comes with guarantees, though their algorithm restricts the family of kernels in a specific way.

The rest of the paper is organized as follows. The problem is defined formally in Section 2. Our new algorithm is presented and analyzed in Section 3, while its specialized version for learning polynomial kernels is given in Section 4. Finally, experiments are provided in Section 5.

2. Preliminaries

In this section we give the formal definition of our problem. Let $\mathcal{I}$ denote a finite index set, indexing the predictors (features) to be combined, and define the set of predictors considered over the input space $\mathcal{X}$ as $\mathcal{F} = \left\{ f_w : \mathcal{X} \to \mathbb{R} \,:\, f_w(x) = \sum_{i\in\mathcal{I}} \langle w_i, \phi_i(x)\rangle\,, \quad x \in \mathcal{X} \right\}$. Here $\mathcal{W}_i$ is a Hilbert space over the reals, $\phi_i : \mathcal{X} \to \mathcal{W}_i$ is a feature-map, $\langle x, y\rangle$ is the inner product over the Hilbert space that x, y belong to and $w = (w_i)_{i\in\mathcal{I}} \in \mathcal{W} \doteq \times_{i\in\mathcal{I}}\mathcal{W}_i$ (as an example, $\mathcal{W}_i$ may just be a finite dimensional Euclidean space). The problem we consider is to solve the optimization problem

$$\text{minimize } L_n(f_w)+\text{Pen}(f_w) \quad \text{subject to } w \in \mathcal{W}\,, \quad (1)$$

where $\text{Pen}(f_w)$ is a penalty that will be specified later, and $L_n(f_w) = \frac{1}{n}\sum_{t=1}^n \ell_t(f_w(x_t))$is the empirical risk of predictor f_w, defined in terms of the convex losses $\ell_t : \mathbb{R} \to \mathbb{R}$ $(1 \le t \le n)$ and inputs $x_t \in \mathcal{X}$ $(1 \le t \le n)$. The solution w^* of the above penalized empirical risk minimization problem is known to have favorable generalization properties under various conditions, see, e.g., Hastie et al. (2009). In supervised learning problems $\ell_t(y) = \ell(y_t, y)$ for some loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, such as the squared-loss, $\ell(y_t, y) = \frac{1}{2}(y - y_t)^2$, or the hinge-loss, $\ell_t(y_t, y) = \max(1 - yy_t, 0)$, where in the former case $y_t \in \mathbb{R}$, while in the latter case $y_t \in \{-1, +1\}$. We note in passing that for the sake of simplicity, we shall sometimes abuse notation and write $L_n(w)$ for $L_n(f_w)$ and even drop the index n when the sample-size is unimportant.

As mentioned above, in this paper we consider the special case in (1) when the penalty is a so-called group p-norm penalty with $1 \le p \le 2$, a case considered earlier, e.g., by Kloft et al. (2011). Thus our goal is to solve

$$\underset{w\in\mathcal{W}}{\text{minimize}} \; L_n(w) + \frac{1}{2}\left(\sum_{i\in\mathcal{I}} \rho_i^p \|w_i\|_2^p\right)^{\frac{2}{p}}\,, \quad (2)$$

where the scaling factors $\rho_i > 0, i \in \mathcal{I}$, are assumed to be given. We introduce the notation $u = (u_i) \in \mathbb{R}^{\mathcal{I}}$ to denote the column vector obtained from the values u_i.

The rationale of using the squared weighted p-norm is that for $1 \le p < 2$ it is expected to encourage sparsity at the group level which should allow one to handle cases when $\mathcal{I}$ is very large (and the case $p = 2$ comes for free from the same analysis). The actual form, however, is also chosen for reasons of computational convenience. In fact, the reason to use the 2-norm of the weights is to allow the algorithm to work even with infinite-dimensional feature vectors (and thus weights) by resorting to the kernel trick. To see how this works, just notice that the penalty in (2) can also be written as

$$\left(\sum_{i\in\mathcal{I}} \rho_i^p \|w_i\|_2^p\right)^{\frac{2}{p}} = \inf\left\{\sum_{i\in\mathcal{I}} \frac{\rho_i^2\|w_i\|_2^2}{\theta_i} \,:\, \theta \in \Delta_{\frac{p}{2-p}}\right\}\,,$$

where for $\nu \ge 1$, $\Delta_\nu = \{\theta \in [0,1]^{|\mathcal{I}|} : \|\theta\|_\nu \le 1\}$ is the positive quadrant of the $|\mathcal{I}|$-dimensional ℓ^ν-ball (see, e.g., Micchelli and Pontil, 2005, Lemma 26). Hence, defining

$$J(w,\theta) = L(w) + \frac{1}{2}\sum_{i\in\mathcal{I}} \frac{\rho_i^2\|w_i\|_2^2}{\theta_i}$$

for any $w \in \mathcal{W}, \theta \in [0,1]^{|\mathcal{I}|}$, an equivalent form of (2) is

$$\underset{w\in\mathcal{W},\theta\in\Delta_\nu}{\text{minimize}} \; J(w,\theta)$$

where $\nu = p/(2-p) \in [1,\infty)$ and we define $0/0 = 0$ and $u/0 = \infty$ for $u > 0$, which implies that $w_i = 0$ if $\theta_i = 0$. That this minimization problem is indeed equivalent to our original task (2) for the chosen value of ν follows from the fact that $J(w,\theta)$ is jointly convex in (w,θ).

Let $\kappa_i : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be the reproducing kernel underlying ϕ_i: $\kappa_i(x,x') = \langle \phi_i(x), \phi_i(x')\rangle$ $(x, x' \in \mathcal{X})$ and let $\mathcal{H}_i = H_{\kappa_i}$ the corresponding reproducing kernel Hilbert space (RKHS). Then, for any given fixed value of θ, the above problem becomes an instance of a standard

penalized learning problem in the RKHS $\mathcal{H}_\theta$ underlying the kernel $\kappa_\theta = \sum_{i\in\mathcal{I}} \theta_i \rho_i^{-2} \kappa_i$. In particular, by the theorem on page 353 in Aronszajn (1950), the problem of finding $w \in \mathcal{W}$ for fixed θ can be seen to be equivalent to minimize$_{f\in\mathcal{H}_\theta}$ $L(f) + \frac{1}{2}\|f\|^2_{\mathcal{H}_\theta}$, and thus (2) is seen to be equivalent to minimize$_{f\in\mathcal{H}_\theta, \theta\in\Delta_\nu}$ $L(f) + \frac{1}{2}\|f\|^2_{\mathcal{H}_\theta}$. Thus, we see that the method can be thought of as finding the weights of a kernel κ_θ and a predictor minimizing the $\mathcal{H}_\theta$-norm penalized empirical risk. This shows that our problem is an instance of multiple kernel learning (for an exhaustive survey of MKL, see, e.g., Gönen and Alpaydın, 2011 and the references therein).

3. The new approach

When $\mathcal{I}$ is small, or moderate in size, the joint-convexity of J allows one to use off-the-shelf solvers to find the joint minimum of J. However, when $\mathcal{I}$ is large, off-the-shelf solvers might be slow or they may run out of memory. Targeting this situation we propose the following approach: Exploiting again that $J(w, \theta)$ is jointly convex in (w, θ), find the optimal weights by finding the minimizer of

$$J(\theta) \doteq \inf_w J(w, \theta),$$

or, alternatively, $J(\theta) = J(w^*(\theta), \theta)$, where $w^*(\theta) \doteq \arg\min_w J(w, \theta)$ (here we have slightly abused notation by reusing the symbol J). Note that $J(\theta)$ is convex by the joint convexity of $J(w, \theta)$. Also, note that $w^*(\theta)$ exists and is well-defined as the minimizer of $J(\cdot, \theta)$ is unique for any $\theta \in \Delta_\nu$ (see also Proposition 3.2 below). Again, exploiting the joint convexity of $J(w, \theta)$, we find that if θ^* is the minimizer of $J(\theta)$, then $w^*(\theta^*)$ will be an optimal solution to the original problem (2). To optimize $J(\theta)$ we propose to use stochastic gradient descent with artificially injected randomness to avoid the need to fully evaluate the gradient of J. More precisely, our proposed algorithm is an instance of a randomized version of the *mirror descent algorithm* (Rockafellar, 1976; Martinet, 1978; Nemirovski and Yudin, 1998), where in each time step only one coordinate of the gradient is sampled.

3.1. A randomized mirror descent algorithm

Before giving the algorithm, we need a few definitions. Let $d = |\mathcal{I}|$, $A \subset \mathbb{R}^d$ be nonempty with a convex interior A°. We call the function $\Psi : A \to \mathbb{R}$ a *Legendre* (or barrier) *potential* if it is strictly convex, its partial derivatives exist and are continuous, and for every sequence $\{x_k\} \subset A$ approaching the boundary of A, $\lim_{k\to\infty} \|\nabla\Psi(x_k)\| = \infty$. Here ∇ is the gradient operator: $\nabla\Psi(x) = (\frac{\partial}{\partial x}\Psi(x))^\top$ is the gradient of Ψ. When ∇ is applied to a non-smooth convex function $J'(\theta)$ (J may be such without additional assumptions) then

Algorithm 1 Randomized mirror descent algorithm

1: **Input:** $A, K \subset \mathbb{R}^d$, where K is closed and convex with $K \cap A \neq \emptyset$, $\Psi : A \to \mathbb{R}$ Legendre, step sizes $\{\eta_k\}$, a subroutine, `GradSampler`, to sample the gradient of J at an arbitrary vector $\theta \geq 0$
2: **Initialization:** $\theta^{(0)} = \arg\min_{\theta\in K\cap A} \Psi(\theta)$.
3: **for** $k = 1, 2, \ldots$ **do**
4: Obtain $\hat{g}_k =$ `GradSampler`$(\theta^{(k-1)})$
5: $\tilde{\theta}^{(k)} = \arg\min_{\theta\in A} \left\{\eta_{k-1}\langle \hat{g}_k, \theta\rangle + D_\Psi(\theta, \theta^{(k-1)})\right\}$.
6: $\theta^{(k)} = \Pi_{\Psi,K}(\tilde{\theta}^{(k)})$.
7: **end for**

$\nabla J'(\theta)$ is defined as any subgradient of J' at θ. The corresponding Bregman-divergence $D_\Psi : A \times A^\circ \to \mathbb{R}$ is defined as $D_\Psi(\theta, \theta') = \Psi(\theta) - \Psi(\theta') - \langle\nabla\Psi(\theta'), \theta - \theta'\rangle$. The Bregman projection $\Pi_{\Psi,K} : A^\circ \to K$ corresponding to the Legendre potential Ψ and a closed convex set $K \subset \mathbb{R}^d$ such that $K \cap A \neq \emptyset$ is defined, for all $\theta \in A^\circ$ as $\Pi_{\Psi,K}(\theta) = \arg\min_{\theta'\in K\cap A} D_\psi(\theta', \theta)$.

Algorithm 1 shows a randomized version of the standard mirror descent method with an unbiased gradient estimate. By assumption, $\eta_k > 0$ is deterministic. Note that step 1 of the algorithm is well-defined since $\tilde{\theta}^{(k)} \in A^\circ$ by the assumption that $\|\nabla\Psi(x)\|$ tends to infinity as x approaches the boundary of A. The performance of Algorithm 1 is bounded in the next theorem. The analysis follows the standard proof technique of analyzing the mirror descent algorithm (see, e.g., Beck and Teboulle, 2003), however, in a slightly more general form than what we have found in the literature. In particular, compared to (Nemirovski et al., 2009; Nesterov, 2010; 2012; Shalev-Shwartz and Tewari, 2011; Richtárik and Takáč, 2011), our analysis allows for the conditional distribution of the noise in the gradient estimate to be history dependent. The proof is omitted due to space limitations and is given in (Afkanpour et al., 2013).

Theorem 3.1. *Assume that Ψ is α-strongly convex with respect to some norm $\|\cdot\|$ (with dual norm $\|\cdot\|_*$) for some $\alpha > 0$, that is, for any $\theta \in A^\circ, \theta' \in A$*

$$\Psi(\theta') - \Psi(\theta) \geq \langle\nabla\Psi(\theta), \theta' - \theta\rangle + \tfrac{\alpha}{2}\|\theta' - \theta\|^2.$$

Suppose, furthermore, that Algorithm 1 is run for T time steps. For $0 \leq k \leq T-1$ let $\mathcal{F}_k$ denote the σ-algebra generated by $\theta_1, \ldots, \theta_k$. Assume that, for all $1 \leq k \leq T$, $\hat{g}_k \in \mathbb{R}^d$ is an unbiased estimate of $\nabla J(\theta^{(k-1)})$ given $\mathcal{F}_{k-1}$, that is, $\mathbb{E}[\hat{g}_k | \mathcal{F}_{k-1}] = \nabla J(\theta^{(k-1)})$. Further, assume that there exists a deterministic constant $B \geq 0$ such that for all $1 \leq k \leq T$, $\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1}\right] \leq B$ a.s. Finally, assume that $\delta = \sup_{\theta'\in K\cap A} \Psi(\theta') - \Psi(\theta^{(0)})$ is finite. Then, if $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{BT}}$ for all $k \geq 1$, it holds that

$$\mathbb{E}\left[J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right)\right] - \inf_{\theta\in K\cap A} J(\theta) \leq \sqrt{\frac{2B\delta}{\alpha T}}.$$

Furthermore, if $\|\hat{g}_k\|_*^2 \le B'$ *a.s. for some deterministic constant* B' *and* $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{B'T}}$ *for all* $k \ge 1$ *then, for any* $0 < \epsilon < 1$, *it holds with probability at least* $1-\epsilon$ *that*

$$J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right) - \inf_{\theta\in K\cap A} J(\theta) \le \sqrt{\frac{2B'\delta}{\alpha T}} + 4\sqrt{\frac{B'\delta\log\frac{1}{\epsilon}}{\alpha T}}.$$

The convergence rate in the above theorem can be improved if stronger assumptions are made on J, for example if J is assumed to be strongly convex, see, for example, (Hazan et al., 2007; Hazan and Kale, 2011).

Efficient implementation of Algorithm 1 depends on efficient implementations of steps 1-1, namely, computing an estimate of the gradient, solving the minimization for $\tilde{\theta}^{(k)}$, and projecting it into K. The first problem is related to the choice of gradient estimate we use, which, in turn, depends on the structure of the feature space, while the last two problems depend on the choice of the Legendre function. In the next subsections we examine how these choices can be made to get a practical variant of the algorithm.

3.2. Application to multiple kernel learning

It remains to define the gradient estimates $\hat{g}_k$ in Algorithm 1. We start by considering importance sampling based estimates. First, however, let us first verify whether the gradient exist. Along the way, we will also derive some explicit expressions which will help us later.

Closed-form expressions for the gradient. Let us first consider how $w^*(\theta)$ can be calculated for a fixed value of θ. As it will turn out, this calculation will be useful not only when the procedure is stopped (to construct the predictor $f_{w^*(\theta)}$ but also during the iterations when we will need to calculate the derivative of J with respect to θ_i. The following proposition summarizes how $w^*(\theta)$ can be obtained. Note that this type of result is standard (see, e.g., Shawe-Taylor and Cristianini, 2004; Schölkopf and Smola, 2002), thus we include it only for the sake of completeness.

Proposition 3.2. *For* $1 \le t \le n$, *let* $\ell_t^* : \mathbb{R} \to \mathbb{R}$ *denote the convex conjugate of* ℓ_t: $\ell_t^*(v) = \sup_{\tau\in\mathbb{R}}\{v\tau - \ell_t(\tau)\}$, $v \in \mathbb{R}$. *For* $i \in \mathcal{I}$, *recall that* $\kappa_i(x,x') = \langle\phi_i(x),\phi_i(x')\rangle$, *and let* $\mathcal{K}_i = (\kappa_i(x_t,x_s))_{1\le t,s\le n}$ *be the* $n\times n$ *kernel matrix underlying* κ_i *and let* $\mathcal{K}_\theta = \sum_{i\in\mathcal{I}}\frac{\theta_i}{\rho_i^2}\mathcal{K}_i$ *be the kernel matrix underlying* $\kappa_\theta = \sum_{i\in\mathcal{I}}\frac{\theta_i}{\rho_i^2}\kappa_i$. *Then, for any fixed* θ, *the minimizer* $w^*(\theta)$ *of* $J(\cdot,\theta)$ *satisfies*

$$w_i^*(\theta) = \frac{\theta_i}{\rho_i^2}\sum_{t=1}^{n}\alpha_t^*(\theta)\phi_i(x_t), \quad i\in\mathcal{I},$$

where

$$\alpha^*(\theta) = \arg\min_{\alpha\in\mathbb{R}^n}\left\{\tfrac{1}{2}\alpha^\top\mathcal{K}_\theta\alpha + \tfrac{1}{n}\sum_{t=1}^{n}\ell_t^*(-n\alpha_t)\right\}.$$

Based on this proposition, we can compute the predictor $f_{w^*(\theta)}$ using the kernels $\{\kappa_i\}_{i\in\mathcal{I}}$ and the dual variables $(\alpha_t^*(\theta))_{1\le t\le n}$: $f_{w^*(\theta)}(x) = \sum_{i\in\mathcal{I}}\langle w_i^*(\theta),\phi_i(x)\rangle = \sum_{t=1}^{n}\alpha_t^*(\theta)\kappa_\theta(x_t,x)$.

Let us now consider the differentiability of $J = J(\theta)$ and how to compute its derivatives. Under proper conditions with standard calculations (e.g., Rakotomamonjy et al., 2008) we find that J is differentiable over Δ and its derivative can be written as

$$\frac{\partial}{\partial\theta}J(\theta) = -\left(\frac{\alpha^*(\theta)^\top\mathcal{K}_i\alpha^*(\theta)}{\rho_i^2}\right)_{i\in\mathcal{I}}. \tag{3}$$

Importance sampling based estimates. Let $d = |\mathcal{I}|$ and let e_i, $i\in\mathcal{I}$ denote the i^{th} unit vector of the standard basis of $\mathbb{R}^d$, that is, the i^{th} coordinate of e_i is 1 while the others are 0. Introduce

$$g_{k,i} = \left\langle\nabla J(\theta^{(k-1)}), e_i\right\rangle, \quad i\in\mathcal{I}$$

to denote the i^{th} component of the gradient of J in iteration k (that is, $g_{k,i}$ can be computed based on (3)). Let $s_{k-1}\in[0,1]^{\mathcal{I}}$ be a distribution over $\mathcal{I}$, computed in some way based on the information available up to the end of iteration $k-1$ of the algorithm (formally, s_{k-1} is $\mathcal{F}_{k-1}$-measurable). Define the importance sampling based gradient estimate to be

$$\hat{g}_{k,i} = \frac{\mathbb{I}_{\{I_k=i\}}}{s_{k-1,I_k}}g_{k,I_k}, \quad i\in\mathcal{I}, \text{ where } I_k\sim s_{k-1,\cdot}. \tag{4}$$

That is, the gradient estimate is obtained by first sampling an index from $s_{k-1,\cdot}$ and then setting the gradient estimate to be zero at all indices $i\in\mathcal{I}$ except when $i = I_k$ in which case its value is set to be the ratio $\frac{g_{k,I_k}}{s_{k-1,I_k}}$. It is easy to see that as long as $s_{k-1,i} > 0$ holds whenever $g_{k,i}\neq 0$, then it holds that $\mathbb{E}[\hat{g}_k|\mathcal{F}_{k-1}] = \nabla J(\theta^{(k-1)})$ a.s.

Let us now derive the conditions under which the second moment of the gradient estimate stays bounded. Define $C_{k-1} = \left\|\nabla J(\theta^{(k-1)})\right\|_1$. Given the expression for the gradient of J shown in (3), we see that $\sup_{k\ge1}C_{k-1} < \infty$ will always hold provided that $\alpha^*(\theta)$ is continuous since $(\theta^{(k-1)})_{k\ge1}$ is guaranteed to belong to a compact set (the continuity of α^* is discussed in Afkanpour et al., 2013).

Define the probability distribution $q_{k-1,\cdot}$ as follows: $q_{k-1,i} = \frac{1}{C_{k-1}}|g_{k,i}|$, $i\in\mathcal{I}$. Then it holds that $\|\hat{g}_k\|_*^2 = \frac{1}{s_{k-1,I_k}^2}g_{k,I_k}^2\|e_{I_k}\|_*^2 = \frac{q_{k-1,I_k}^2}{s_{k-1,I_k}^2}C_{k-1}^2\|e_{I_k}\|_*^2$.

Algorithm 2 Projected stochastic gradient algorithm.

1: **Initialization:** $\Psi(x) = \frac{1}{2}\|x\|_2^2$, $\theta_i^{(0)} = 0$ for all $i \in \mathcal{I}$, step sizes $\{\eta_k\}$.
2: **for** $k = 1, 2, \ldots$ **do**
3: Sample a gradient estimate $\hat{g}_k$ of $g(\theta^{(k-1)})$ randomly according to (4).
4: $\theta^{(k)} = \Pi_{\Psi,\Delta_2}(\theta^{(k-1)} - \eta_{k-1}\hat{g}_k)$.
5: **end for**

Therefore, it also holds that $\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1}\right] = C_{k-1}^2 \sum_{i\in\mathcal{I}} \frac{q_{k-1,i}^2}{s_{k-1,i}}\|e_i\|_*^2 \le C_{k-1}^2 \max_{i\in\mathcal{I}} \frac{q_{k-1,i}}{s_{k-1,i}}\|e_i\|_*^2$. This shows that $\sup_{k\ge1}\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1}\right] < \infty$ will hold as long as $\sup_{k\ge1}\max_{i\in\mathcal{I}} \frac{q_{k-1,i}}{s_{k-1,i}} < \infty$ and $\sup_{k\ge1} C_{k-1} < \infty$. Note that when $s_{k-1} = q_{k-1}$, the gradient estimate becomes $\hat{g}_{k,i} = C_{k-1}\mathbb{I}_{\{I_t=i\}}$. That is, in this case we see that in order to be able to calculate $\hat{g}_{k,i}$, we need to be able to calculate C_{k-1} efficiently.

Choosing the potential Ψ. The efficient sampling of the gradient is not the only practical issue, since the choice of the Legendre function and the convex set K may also cause some complications. For example, if $\Psi(x) = \sum_{i\in\mathcal{I}} x_i(\ln x_i - 1)$, then the resulting algorithm is exponential weighting, and one needs to store and update $|\mathcal{I}|$ weights, which is clearly infeasible if $|\mathcal{I}|$ is very large (or infinite). On the other hand, if $\Psi(x) = \frac{1}{2}\|x\|_2^2$ and we project to $K = \Delta_2$, the positive quadrant of the ℓ^2-ball (with $A = [0,\infty)^{\mathcal{I}}$), we obtain a stochastic projected gradient method, shown in Algorithm 2. This is in fact the algorithm that we use in the experiments. Note that in (2) this corresponds to using $p = 4/3$. The reason we made this choice is because in this case projection is a simple scaling operation. Had we chosen $K = \Delta_1$, the ℓ^2-projection would very often cancel many of the nonzero components, resulting in an overall slow progress. Based on the above calculations and Theorem 3.1 we obtain the following performance bound for our algorithm.

Corollary 3.3. *Assume that $\alpha^*(\theta)$ is continuous on Δ_2. Then there exists a $C > 0$ such that $\|\frac{\partial}{\partial\theta}J(\theta)\|_1 \le C$ for all $\theta \in \Delta_2$. Let $B = \frac{1}{2}C^2 \max_{i\in\mathcal{I},1\le k\le T} \frac{q_{k-1,i}}{s_{k-1,i}}$. If Algorithm 2 is run for T steps with $\eta_{k-1} = \eta = 1/\sqrt{BT}, k = 1,\ldots,T$, then, for all $\theta \in \Delta_2$,*

$$\mathbb{E}\left[J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right)\right] - J(\theta) \le \sqrt{\frac{B}{T}}.$$

Note that to implement Algorithm 2 efficiently, one has to be able to sample from $s_{k-1,\cdot}$ and compute the importance sampling ratio $g_{k,i}/s_{k,i}$ efficiently for any k and i.

4. Example: Polynomial kernels

In this section we show how our method can be applied in the context of multiple kernel learning. We provide an example when the kernels in $\mathcal{I}$ are tensor products of a set of base kernels (this we shall call learning polynomial kernels). The importance of this example follows from the observation of Gönen and Alpaydın (2011) that the non-linear kernel learning methods of Cortes et al. (2009), which can be viewed as a restricted form of learning polynomial kernels, are far the best MKL methods in practice and can significantly outperform state-of-the-art SVM with a single kernel or with the uniform combination of kernels.

Assume that we are given a set of base kernels $\{\kappa_1,\ldots,\kappa_r\}$. In this section we consider the set K_D of product kernels of degree at most D: Choose $\mathcal{I} = \{(r_1,\ldots,r_d) : 0 \le d \le D, 1 \le r_i \le r\}$ and the multi-index $r_{1:d} = (r_1,\ldots,r_d) \in \mathcal{I}$ defines the kernel $\kappa_{r_{1:d}}(x,x') = \prod_{i=1}^d \kappa_{r_i}(x,x')$. For $d = 0$ we define $\kappa_{r_{1:0}}(x,x') = 1$. Note that indices that are the permutations of each other define the same kernel. On the language of statistical modeling, $\kappa_{r_{1:d}}$ models interactions of order d between the features underlying the base kernels $\kappa_1,\ldots,\kappa_r$. Also note that $|\mathcal{I}| = \Theta(r^D)$, that is, the cardinality of $\mathcal{I}$ grows exponentially fast in D.

We assume that $\rho_{r_{1:d}}$ depends only on d, the order of interactions in $\kappa_{r_{1:d}}$. By abusing notation, we will write ρ_d in the rest of this section to emphasize this.[1] Our proposed algorithm to sample from $q_{k-1,\cdot}$ is shown in Algorithm 3. The algorithm is written to return a multi-index $(z_1,\ldots,z_d)$ that is drawn from $q_{k-1,\cdot}$. The key idea underlying the algorithm is to exploit that $(\sum_{j=1}^r \kappa_j)^d = \sum_{r_{1:d}\in\mathcal{I}} \kappa_{r_{1:d}}$. The correctness of the algorithm is shown in Section 4.1. In the description of the algorithm $\odot$ denotes the matrix entrywise product (a.k.a. Schur, or Hadamard product) and $A^{\odot s}$ denotes $\underbrace{A \odot \ldots \odot A}_{s}$, and we set the priority of $\odot$ to be higher than that of the ordinary matrix product (by definition, all the entries of $A^{\odot 0}$ are 1).

Let us now discuss the complexity of Algorithm 3. For this, first note that computing all the Hadamard products $S^{\odot d'}, d' = 0,\ldots,D$ requires $O(Dn^2)$ computations. Multiplication with M_{k-1} can be done in $O(n^2)$ steps. Finally, note that each iteration of the for loop takes $O(rn^2)$ steps, which results in the overall worst-case complexity of $O(rn^2D)$ if $\alpha^*(\theta_{k-1})$ is readily available. The computational complexity of determining $\alpha^*(\theta_{k-1})$ depends on the exact form of ℓ_t, and can be done efficiently in many situations: if, for example, ℓ_t is the squared loss, then α^* can be

[1] Using importance sampling, more general weights can also be accommodated, too without effecting the results as long as the range of weights ($\rho_{r_{1:d}}$) is kept under control for all d.

Algorithm 3 Polynomial kernel sampling.

Input: $\alpha \in \mathbb{R}^n$, the solution to the dual problem; kernel matrices $\{\mathcal{K}_1, \ldots, \mathcal{K}_r\}$; the degree D of the polynomial kernel, the weights $(\rho_0^2, \ldots, \rho_D^2)$.
$S \leftarrow \sum_{j=1}^r \mathcal{K}_j$, $M \leftarrow \alpha\alpha^\top$
$\delta(d') \leftarrow \rho_{d'}^{-2} \left\langle M, S^{\odot d'} \right\rangle, \quad d' \in \{0, \ldots, D\}$
Sample d from $\delta(\cdot)/\sum_{d'=0}^D \delta(d')$
for $i = 1$ to d **do**
  $\pi(j) \leftarrow \frac{\mathrm{tr}(M\, S^{\odot(d-i)} \odot \mathcal{K}_j)}{\mathrm{tr}(M\, S^{\odot(d-i+1)})}, \quad j \in \{1, \ldots, r\}$
  Sample z_i from $\pi(\cdot)$
  $M \leftarrow M \odot \mathcal{K}_{z_i}$
end for
return $(z_1, \ldots, z_d)$

computed in $O(n^3)$ time. An obvious improvement to the approach described here, however, would be to subsample the empirical loss L_n, which can bring further computational improvements. However, the exploration of this is left for future work.

Finally, note that despite the exponential cardinality of $|\mathcal{I}|$, due to the strong algebraic structure of the space of kernels, C_{k-1} can be calculated efficiently. In fact, it is not hard to see that with the notation of the algorithm, $C_{k-1} = \sum_{d'=0}^D \delta(d')$. This also shows that if ρ_d decays "fast enough", C_{k-1} can be bounded independently of the cardinality of $\mathcal{I}$.

4.1. Correctness of the sampling procedure

In this section we prove the correctness of Algorithm 3.

As said earlier, we assume that $\rho_{r_{1:d}}$ depends only on d, the order of interactions in $\kappa_{r_{1:d}}$ and, by abusing notation, we will write ρ_d to emphasize this. Let us now consider how one can sample from $q_{k-1,\cdot}$. The implementation relies on the fact that $(\sum_{j=1}^r \kappa_j)^d = \sum_{r_{1:d} \in \mathcal{I}} \kappa_{r_{1:d}}$.

Remember that we denoted the kernel matrix underlying some kernel k by $\mathcal{K}_k$, and recall that $\mathcal{K}_k$ is an $n \times n$ matrix. For brevity, in the rest of this section for $\kappa = \kappa_{r_{1:d}}$ we will write $\mathcal{K}_{r_{1:d}}$ instead of $\mathcal{K}_{\kappa_{r_{1:d}}}$. Define $M_{k-1} = \alpha^*(\theta_{k-1})\alpha^*(\theta_{k-1})^\top$. Thanks to (3) and the rotation property of trace, we have $g_{k,r_{1:d}} = -\rho_d^{-2}\mathrm{tr}(M_{k-1}\,\mathcal{K}_{r_{1:d}})$. The plan to sample from $q_{k-1,\cdot} = |g_{k,\cdot}|/\sum_{r_{1:d}\in\mathcal{I}} |g_{k,r_{1:d}}|$ is as follows: We first draw the order of interactions, $0 \le \hat{d} \le D$. Given $\hat{d} = d$, we restrict the draw of the random multi-index $R_{1:d}$ to the set $\{r_{1:d} \in \mathcal{I}\}$. A multi-index will be sampled in a $\hat{d}$-step process: in each step we will randomly choose an index from the indices of base kernels according to the following distributions. Let $S = \mathcal{K}_1 + \ldots + \mathcal{K}_r$, let

$$\mathbb{P}\left(\hat{d} = d | \mathcal{F}_{k-1}\right) = \frac{\rho_d^{-2}\mathrm{tr}(M_{k-1}S^{\odot d})}{\sum_{d'=0}^D \rho_{d'}^{-2}\mathrm{tr}(M_{k-1}S^{\odot d'})}$$

and, with a slight abuse of notation, for any $1 \le i \le d$ define

$$\mathbb{P}\left(R_i = r_i | \mathcal{F}_{k-1}, \hat{d} = d, R_{1:i-1} = r_{1:i-1}\right) = \frac{\mathrm{tr}\left(M_{k-1} \odot \left(\odot_{j=1}^i \mathcal{K}_{r_j}\right) \odot S^{\odot(d-i)}\right)}{\sum_{r_i'=1}^r \mathrm{tr}\left(M_{k-1} \odot \left(\odot_{j=1}^{i-1} \mathcal{K}_{r_j}\right) \odot \mathcal{K}_{r_i'} \odot S^{\odot(d-i)}\right)}$$

where we used the sequence notation (namely, $s_{1:p}$ denotes the sequence $(s_1, \ldots, s_p)$). We have, by the linearity of trace and the definition of S that

$$\sum_{r_i'=1}^r \mathrm{tr}\left(M_{k-1} \odot \left(\odot_{j=1}^{i-1} \mathcal{K}_{r_j}\right) \odot \mathcal{K}_{r_i'} \odot S^{\odot(d-i)}\right) = \mathrm{tr}\left(M_{k-1} \odot \left(\odot_{j=1}^{i-1} \mathcal{K}_{r_j}\right) \odot S^{\odot(d-i+1)}\right)$$

Thus, by telescoping,

$$\mathbb{P}\left(\hat{d} = d, R_{1:d} = r_{1:d} | \mathcal{F}_{k-1}\right) = \frac{\rho_d^{-2}\mathrm{tr}(M_{k-1}\,\mathcal{K}_{r_1} \odot \ldots \odot \mathcal{K}_{r_{d-1}} \odot \mathcal{K}_{r_d})}{\sum_{d'=0}^D \rho_{d'}^{-2}\mathrm{tr}(M_{k-1}S^{\odot d'})}.$$

as desired. An optimized implementation of drawing these random variables is shown as Algorithm 3. The algorithm is written to return the multi-index $R_{1:d}$.

5. Experiments

In this section we apply our method to the problem of multiple kernel learning in regression with the squared loss: $L(w) = \frac{1}{2}\sum_{t=1}^n (f_w(x_t) - y_t)^2$, where $(x_t, y_t) \in \mathbb{R}^r \times \mathbb{R}$ are the input-output pairs in the data. In these experiments our aim is to learn polynomial kernels (cf. Section 4).

We compare our method against several kernel learning algorithms from the literature on synthetic and real data. In all experiments we report mean squared error over test sets. A constant feature is added to act as offset, and the inputs and output are normalized to have zero mean and unit variance. Each experiment is performed with 10 runs in which we randomly choose training, validation, and test sets. The results are averaged over these runs.

5.1. Convergence speed

In this experiment we examine the speed of convergence of our method and compare it against one of the fastest standard multiple kernel learning algorithms, that is, the p-norm multiple kernel learning algorithm of Kloft et al. (2011) with $p = 2$,[2] and the uniform coordinate

[2] Note that $p = 2$ in Kloft et al. (2011) notation corresponds to $p = 4/3$ or $\nu = 2$ in our notation, which gives the same objective function that we minimize with Algorithm 2.

descent algorithm that updates one coordinate per iteration uniformly at random (Nesterov, 2010; 2012; Shalev-Shwartz and Tewari, 2011; Richtárik and Takáč, 2011). We aim to learn polynomial kernels of up to degree 3 with all algorithms. Our method uses Algorithm 3 for sampling with $D = 3$. The set of provided base kernels is the linear kernels built from input variables, that is, $\kappa_{(i)}(x, x') = x_{(i)}x'_{(i)}$, where $x_{(i)}$ denotes the i^{th} input variable. For the other two algorithms the kernel set consists of product kernels from monomial terms for $D \in \{0, 1, 2, 3\}$ built from r base kernels, where r is the number of input variables. The number of distinct product kernels is $\binom{r+D}{D}$. In this experiment for all algorithms we use ridge regression with its regularization parameter set to 10^{-5}. Experiments with other values of the regularization parameter achieved similar results.

We compare these methods in four datasets from the UCI machine learning repository (Frank and Asuncion, 2010) and the Delve datasets[3]. We run all algorithms for a fixed amount of time and measure the value of the objective function (1), that is, the sum of the empirical loss and the regularization term. Figure 1 shows the performance of these algorithms. In this figure STOCH represents our algorithms, KLOFT represents the algorithm of Kloft et al. (2011), and UCD represents the uniform coordinate descent algorithm.

The results show that our method consistently outperforms the other algorithms in convergence speed. Note that our stochastic method updates one kernel coefficient per iteration, while KLOFT updates $\binom{r+D}{D}$ kernel coefficients per iteration. The difference between the two methods is analogous to the difference between stochastic gradient vs. full gradient algorithms. While UCD also updates one kernel coefficient per iteration its naive method of selecting coordinates results in a slower overall convergence compared to our algorithm. In the next section we compare our algorithm against several representative methods from the MKL literature.

5.2. Synthetic data

In this experiment we examine the effect of the size of the kernel space on prediction accuracy and training time of MKL algorithms on synthetic data. Experiments with real datasets also give promising results. Due to the lack of space these results are presented in (Afkanpour et al., 2013). We generated data for a regression problem. Let r denote the number of dimensions of the input space. The inputs are chosen uniformly at random from $[-1, 1]^r$. The output of each instance is the uniform combination of 10 monomial terms of degree 3 or less. These terms are chosen uniformly at random among all possible terms. The outputs are noise free. We generated data for $r \in \{5, 10, 20, \ldots, 100\}$, with 500 training and 1000 test points. The regularization parameter of the ridge regression algorithm was tuned from $\{10^{-8}, \ldots, 10^{2}\}$ using a separate validation set with 1000 data points.

We compare our method (STOCH) against the algorithm of Kloft et al. (2011) (KLOFT), the nonlinear kernel learning method of Cortes et al. (2009) (CORTES), and the hierarchical kernel learning algorithm of Bach (2008) (BACH).[4] The set of base kernels consists of r linear kernels built from the input variables. Recall that the method of Cortes et al. (2009) only considers kernels of the form $\kappa_\theta = (\sum_{i=1}^{r} \theta_i \kappa_i)^D$, where D is a predetermined integer that specifies the degree of nonlinear kernel. Note that adding a constant feature is equivalent to adding polynomial kernels of degree less than D to the combination too. We provide all possible product kernels of degree 0 to D to the kernel learning method of Kloft et al. (2011). For our method and the method of Bach (2008) we set the maximum kernel degree to $D = 3$.

The results are shown in Figure 2, the mean squared errors are on the left plot, while the training times are on the right plot. In the training-time plot the numbers inside brackets indicate the total number of distinct product kernels for each value of r. This is the number of kernels fed to the KLOFT algorithm. Since this method deals with a large number of kernels, it was possible to precompute and keep the kernels in memory (8GB) for $r \leq 25$. Therefore, we ran this algorithm for $r \leq 25$. For $r > 25$, we could use on-the-fly implementation of this algorithm, however that further increases the training time. Note that the computational cost of this method depends linearly on the number of kernels, which in this experiment, is cubic in the number of input variables since $D = 3$. While the standard MKL algorithms, such as KLOFT, cannot handle such large kernel spaces, in terms of time and space complexity, the

[3] www.cs.toronto.edu/~delve/data/datasets.html

[4] While several fast MKL algorithms are available in the literature, such as those of Sonnenburg et al. (2006); Rakotomamonjy et al. (2008); Xu et al. (2010); Orabona and Luo (2011); Kloft et al. (2011), a comparison of the reported experimental results shows that from among these algorithms the method of Kloft et al. (2011) has the best performance overall. Hence, we decided to compare against only this algorithm. Also note that the memory and computational cost of all these methods still scale linearly with the number of kernels, making them unsuitable for the case we are most interested in. Furthermore, to keep the focus of the paper we compare our algorithm to methods with sound theoretical guarantees. As such, it remains for future work to compare with other methods, such as the infinite kernel learning of Gehler and Nowozin (2008), which lack such guarantees but exhibit promising performance in practice.

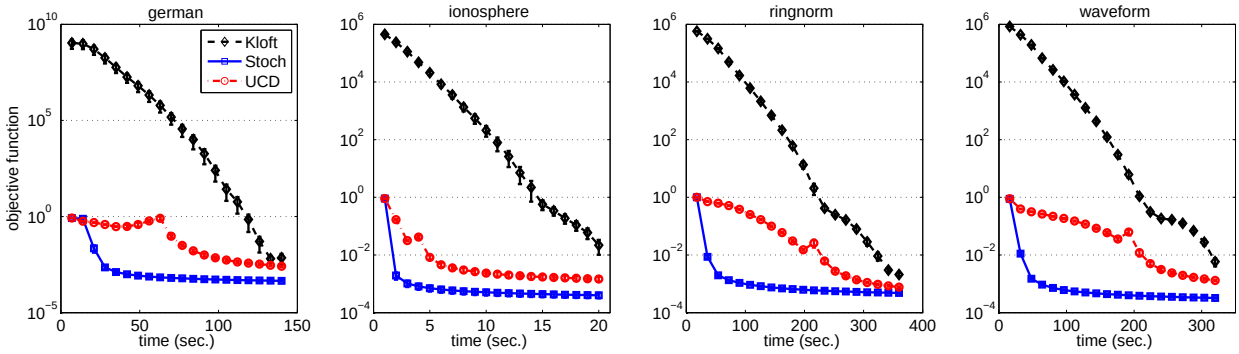

Figure 1. Convergence comparison of our method and other algorithms.

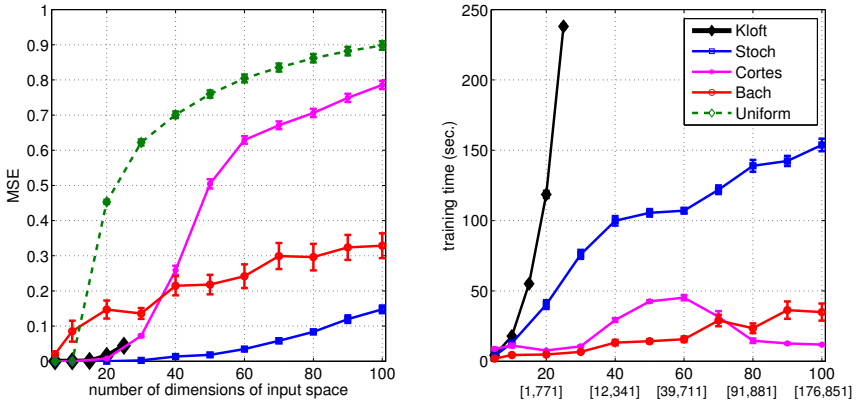

Figure 2. Comparison of kernel learning methods in terms of test error (left) and training time (right).

other three algorithms can efficiently learn kernel combinations. However their predictive accuracies are quite different. Note that the performance of the method of Cortes et al. (2009) starts to degrade as r increases. This is due to the restricted family of kernels that this method considers. The method of Bach (2008), which is well-suited to learn sparse combination of product kernels, performs better than Cortes et al. (2009) for higher input dimensions. Among all methods, our method performs best in predictive accuracy while its computational cost is close to that of the other two competitors.

6. Conclusion

We introduced a new method for learning a predictor by combining exponentially many linear predictors using a randomized mirror descent algorithm. We derived finite-time performance bounds that show that the method efficiently optimizes our proposed criterion. Our proposed method is a variant of a randomized stochastic coordinate descent algorithm, where the main trick is the careful construction of an unbiased randomized estimate of the gradient vector that keeps the variance of the method under control, and can be computed efficiently when the base kernels have a certain special combinatorial structure. The efficiency of our method was demonstrated for the practically important problem of learning polynomial kernels on a variety of synthetic and real datasets comparing to a representative set of algorithms from the literature. For this case, our method is able to compute an optimal solution in polynomial time as a function of the *logarithm* of the number of base kernels. To our knowledge, ours is the first method for learning kernel combinations that achieve such an exponential reduction in complexity while satisfying strong performance guarantees, thus opening up the way to apply it to extremely large number of kernels. Furthermore, we believe that our method is applicable beyond the case studied in detail in our paper. For example, the method seems extendible to the case when infinitely many kernels are combined, such as the case of learning a combination of Gaussian kernels. However, the investigation of this important problem remains subject to future work.

Acknowledgements

This work was supported by Alberta Innovates Technology Futures and NSERC.

References

Afkanpour, A., György, A., Szepesvári, C., and Bowling, M. H. (2013). A randomized mirror descent algorithm for large scale multiple kernel learning. *CoRR*, abs/1205.0288.

Argyriou, A., Hauser, R., Micchelli, C., and Pontil, M. (2006). A DC-programming algorithm for kernel selection. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 41–48.

Argyriou, A., Micchelli, C., and Pontil, M. (2005). Learning convex combinations of continuously parameterized basic kernels. In *Proceedings of the 18th Annual Conference on Learning Theory*, pages 338–352.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

Bach, F. (2008). Exploring large feature spaces with hierarchical multiple kernel learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 105–112.

Beck, A. and Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175.

Cortes, C., Mohri, M., and Rostamizadeh, A. (2009). Learning non-linear combinations of kernels. In *Advances in Neural Information Processing Systems*, volume 22, pages 396–404.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

Gehler, P. and Nowozin, S. (2008). Infinite kernel learning. Technical Report 178, Max Planck Institute For Biological Cybernetics.

Gönen, M. and Alpaydın, E. (2011). Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, 2nd edition.

Hazan, E., Agarwal, A., and Kale, S. (2007). Logarithmic regret algorithms for online convex optimization. *Machine Learning Journal*, 69(2-3):169–192.

Hazan, E. and Kale, S. (2011). Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. In *Proceedings of the 24th Annual Conference on Learning Theory*, volume 19 of *JMLR Workshop and Conference Proceedings*, pages 421–436.

Kloft, M., Brefeld, U., Sonnenburg, S., and Zien, A. (2011). l_p-norm multiple kernel learning. *Journal of Machine Learning Research*, 12:953–997.

Martinet, B. (1978). Perturbation des méthodes d'optimisation. Applications. *RAIRO Analyse Numérique*, 12:153–171.

Micchelli, C. and Pontil, M. (2005). Learning the kernel function via regularization. *Journal of Machine Learning Research*, 6:1099–1125.

Nath, J., Dinesh, G., Raman, S., Bhattacharyya, C., Ben-Tal, A., and Ramakrishnan, K. (2009). On the algorithmics and applications of a mixed-norm based kernel learning formulation. In *Advances in Neural Information Processing Systems*, volume 22, pages 844–852.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM J. Optimization*, 4:1574–1609.

Nemirovski, A. and Yudin, D. (1998). *Problem Complexity and Method Efficiency in Optimization*. Wiley.

Nesterov, Y. (2010). Efficiency of coordinate descent methods on huge-scale optimization problems. *CORE Discussion paper*, (2010/2).

Nesterov, Y. (2012). Subgradient methods for huge-scale optimization problems. *CORE Discussion paper*, (2012/2).

Orabona, F. and Luo, J. (2011). Ultra-fast optimization algorithm for sparse multi kernel learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 249–256.

Rakotomamonjy, A., Bach, F., Canu, S., and Grandvalet, Y. (2008). SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521.

Richtárik, P. and Takáč, M. (2011). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. (revised July 4, 2011) submitted to Mathematical Programming.

Rockafellar, R. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(1):877–898.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Shalev-Shwartz, S. and Tewari, A. (2011). Stochastic methods for l_1-regularized loss minimization. *Journal of Machine Learning Research*, 12:1865–1892.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge Univ Press.

Sonnenburg, S., Rätsch, G., Schäfer, C., and Schölkopf, B. (2006). Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565.

Xu, Z., Jin, R., King, I., and Lyu, M. (2008). An extended level method for efficient multiple kernel learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 1825–1832.

Xu, Z., Jin, R., Yang, H., King, I., and Lyu, M. R. (2010). Simple and efficient multiple kernel learning by group lasso. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1175–1182.