Separable Approximate Optimization of Support Vector Machines for Distributed Sensing

Sangkyun Lee, Marco Stolpe, and Katharina Morik

Fakultät für Informatik, LS VIII
Technische Universität Dortmund
44221 Dortmund, Germany
{sangkyun.lee,marco.stolpe,katharina.morik}@tu-dortmund.de

Abstract. Sensor measurements from diverse locations connected with possibly low bandwidth communication channels pose a challenge of resource-restricted distributed data analyses. In such settings it would be desirable to perform learning in each location as much as possible, without transferring all data to a central node. Applying the support vector machines (SVMs) with nonlinear kernels becomes nontrivial, however.
In this paper, we present an efficient optimization scheme for training SVMs over such sensor networks. Our framework performs optimization independently in each node, using only the local features stored in the respective node. We make use of multiple local kernels and explicit approximations to the feature mappings induced by them. Together they allow us constructing a separable surrogate objective that provides an upper bound of the primal SVM objective. A central coordination is also designed to adjust the weights among local kernels for improved prediction, while minimizing communication cost.

Keywords: distributed features, support vector machines, separable optimization, primal formulation, approximate feature mappings

1 Introduction

Sensor networks have been a very active research topic in recent machine learning and data mining [12, 13]. Sensors are adopted to monitor certain aspects of objects or phenomena that we are interested in, often located in such places hardly accessible by human beings. Various applications include monitoring manufacturing processes, traffic levels, water flows and climate changes over time at different locations, where sensors (or computing nodes embracing local sensors) have to communicate with each other or with a central arbitrator in order to provide useful information for decision making.

Challenges arise in sensor networks when we try to build a predictor collecting information from all sensors, where sensors can afford only minimal communication due to their distance to a central station or low-power requirements. If this is the case, we might prefer to perform learning in a distributed fashion, where each separated part of learning relies on locally stored measurements only.

Learning a global model in such cases requires an approach whose computation can be distributed in a well-defined way, equipped with a global arbitration that can maximize prediction performance without incurring too much information transfer from sensors.

In this paper we suggest a variant of the support vector machines (SVMs) using nonlinear kernels on sensor data. We define kernels that use only locally stored features in sensors, computing explicit forms of approximations to the feature mappings that correspond to each local kernel. For typical error functions of SVMs, we then create a separable surrogate objective function that forms an upper bound on the original primal SVM objective. Each separated part in the objective is designed to use a single local kernel, and therefore can be optimized locally at the sensors.

We also provide an additional central optimization that uses inner product results from the sensors, without requiring an access to local kernel functions or their approximations. The central optimization provides local kernels new weights, that can be fed to the sensors and generate possibly improved local solutions.

2 Separable Optimization

In this section we begin with a general description of the support vector machines (SVMs), shaping it progressively to a form which can be optimized separately for local features in each network node.

2.1 Support Vector Machines

We consider a given data set $\{(\mathbf{x}_u, y_u)\}_{u=1}^m$ which consists of pairs of input feature vectors $\mathbf{x}_u \in \mathbb{R}^p$ and their labels y_u, where $y_u \in \{-1, +1\}$ for classification and $y_u \in \mathbb{R}$ for regression. The SVMs for 1- and 2-class classification and regression can be formulated as an unconstrained convex minimization,

$$\min_{\mathbf{w}\in\mathcal{H},\rho\in\mathbb{R}} \quad \frac{\lambda}{2}\left(\|\mathbf{w}\|_{\mathcal{H}}^2 + \rho\right) + \frac{1}{m}\sum_{u=1}^{m} \ell\left(\langle \mathbf{w}, \phi(\mathbf{x}_u)\rangle, y_u, \rho\right) \tag{1}$$

where $\lambda > 0$ and ℓ is a convex loss function chosen by the task of interest as in Table 1. For readability we ignore the intercept term of a decision plane without loss of generality, which can be easily included by augmenting vectors $\mathbf{w}$ and $\mathbf{x}$ and excluding it from penalization in the first objective term. We call $\phi : \mathbb{R}^p \to \mathcal{H}$, for a Hilbert space $\mathcal{H}$, a *feature mapping* induced by a positive semidefinite kernel k, with the relation that $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}')\rangle$ for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$.

2.2 Multiple Localized Kernels

Now, we consider that the input features are stored in a distributed fashion among n nodes, possibly with overlaps among them. We suppose that there are

Table 1. The loss function ℓ and the range of ρ in the canonical objective of SVMs in the equation (1), and ℓ_i and ρ_i corresponding to the ith summand in the upper bounds of ℓ formed in (6). For the training example indexed by u, we set $z = \langle \mathbf{w}, \phi(\mathbf{x}_u) \rangle$ and $z_i = \mathbf{w}[i]^T \varphi_i(\mathbf{x}_u[i])$. The zero range for ρ and ρ_i implies we ignore them in optimization.

Task	$\ell(z, y, \rho)$	$\ell_i(z_i, y, \rho_i)$	Range ρ, ρ_i
Classification (1-class)	$\max\{0, \rho - z\}$	$\max\{0, \rho_i - z_i\}$	$\mathbb{R}$
Classification (2-class)	$\max\{0, 1 - yz\}$	$\max\{0, 1 - yz_i\}$	0
Regression	$\max\{0, \lvert y - z\rvert - \epsilon\}$	$\max\{0, \lvert y - z_i\rvert - \epsilon\}$	0

p unique features in total, denoting by $\mathcal{S}_i \subset \{1, 2, \ldots, p\}$ the subset of feature indices stored in the ith node, and by $p_i := |\mathcal{S}_i| > 0$ its cardinality, so that $\cup_{i=1}^n \mathcal{S}_i = \{1, 2, \ldots, p\}$ and $\sum_{i=1}^n p_i \geq p$. For convenience, we refer to the feature subvector of $\mathbf{x}_u$ stored in the ith node as $\mathbf{x}_u[i] \in \mathbb{R}^{p_i}$.

For each node we make use of an individual kernel which depends on only the features stored locally in nodes. We denote the kernel for the ith node by $k_i : \mathbb{R}^{p_i \times p_i} \to \mathbb{R}$ and its corresponding feature mapping by $\phi_i : \mathbb{R}^{p_i} \to \mathcal{H}_i$. Then we can construct a *composite* kernel k as a conic combination of local kernels, that is,

$$k(\mathbf{x}, \mathbf{x}') := \sum_{i=1}^n \mu_i^2 k_i(\mathbf{x}[i], \mathbf{x}'[i]). \tag{2}$$

(It will become clear why we use μ_i^2 rather than $\mu_i \geq 0$, as we progress.) The weights for local kernels μ_i will be optimized, which defines our central optimization problem to be discussed later. This setting is very similar to the multiple kernel learning (MKL) and boosting, but our resulting framework will not be exactly the same, as we discuss later in Section 3.

We note that using multiple kernels alone does not lead to a separable objective for SVMs. From the representer theorem [20], the optimal weight $\mathbf{w}$ of SVM (1) is expressed as a linear span of the representers $k(\cdot, \mathbf{x}_v)$. That is,

$$\mathbf{w}(\cdot) = \sum_{v=1}^m \alpha_v k(\cdot, \mathbf{x}_v) \overset{(2)}{=} \sum_{i=1}^n \mu_i^2 \sum_{v=1}^m \alpha_v k_i(\cdot[i], \mathbf{x}_v[i]), \tag{3}$$

Replacing $\mathbf{w}$ into (1) results in the following objective:

$$\min_{\alpha \in \mathbb{R}^m} \ \frac{\lambda}{2} \sum_{i=1}^n \mu_i^2 \sum_{u=1}^m \sum_{v=1}^m \alpha_u \alpha_v k_i(\mathbf{x}_u[i], \mathbf{x}_v[i]) + \frac{1}{m} \sum_{u=1}^m \ell\left(\sum_{i=1}^n \mu_i^2 \sum_{v=1}^m \alpha_v k_i(\mathbf{x}_u[i], \mathbf{x}_v[i]), y_u, \rho \right).$$

We can see that all optimization variables $\alpha_1, \alpha_2, \ldots, \alpha_m$ are coupled with each node $i = 1, 2, \ldots, n$, therefore cannot be split over nodes. This also indicates that we would need alternative ways to incorporate kernels rather than relying on the representer theorem, to achieve separability.

2.3 Approximating Feature Mappings

One observation of the original SVM formulation (1) is that the terms $\|\mathbf{w}\|^2$ and $\langle \mathbf{w}, \phi(\mathbf{x}_u) \rangle$ will be separable over the components of $\mathbf{w}$, if $\mathbf{w}$ and $\phi(\mathbf{x}_u)$ are in a finite dimensional space. (The loss functions ℓ in Table 1 are not separable as well. We will discuss them as the next step in the following section.) Motivated by this, we introduce explicit finite-dimensional approximations to the feature mappings $\phi \in \mathcal{H}$ that correspond to kernel functions $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$. This step is necessary, since the explicit form of ϕ is unavailable in general.

For each node i, suppose that we have obtained an approximate feature mapping $\varphi_i : \mathbb{R}^{p_i} \to \mathbb{R}^{d_i}$ to the original mapping ϕ_i, where $d_i \in (0, \infty)$ is a predefined (possibly small) integer, so that

$$\langle \varphi_i(\mathbf{x}[i]), \varphi_i(\mathbf{x}'[i]) \rangle \;\approx\; k_i(\mathbf{x}[i], \mathbf{x}'[i]) = \langle \phi(\mathbf{x}[i]), \phi(\mathbf{x}'[i]) \rangle.$$

When $d := \sum_{i=1}^n d_i$ is sufficiently large, we can consider the following d-dimensional problem as a good approximation to the original problem (1):

$$\min_{\mathbf{w} \in \mathbb{R}^d, \rho \in R} \;\; \frac{\lambda}{2} \left(\|\mathbf{w}\|_2^2 + \rho \right) + \frac{1}{m} \sum_{u=1}^m \ell \left(\mathbf{w}^T \varphi(\mathbf{x}_u), y_u, \rho \right). \tag{4}$$

Regarding the representation (3), we impose a weight $\mu_i \geq 0$ for the feature mappings in nodes $i = 1, 2, \ldots, n$, so that

$$\varphi(\mathbf{x}) := \begin{bmatrix} \mu_1 \varphi_1(\mathbf{x}[1]) \\ \mu_2 \varphi_2(\mathbf{x}[2]) \\ \vdots \\ \mu_n \varphi_n(\mathbf{x}[n]) \end{bmatrix}, \; \mathbf{w} := \begin{bmatrix} \mathbf{w}[1] \\ \mathbf{w}[2] \\ \vdots \\ \mathbf{w}[n] \end{bmatrix} \;\Rightarrow\; \mathbf{w}^T \varphi(\mathbf{x}) = \sum_{i=1}^n \mu_i \mathbf{w}[i]^T \varphi_i(\mathbf{x}[i]).$$

Here we denote by $\mathbf{w}[i] \in \mathbb{R}^{d_i}$ the subvector of $\mathbf{w} \in \mathbb{R}^d$ for the node i. Again, by $\mathbf{x}[i] \in \mathbb{R}^{p_i}$ we denote the attributes of $\mathbf{x} \in \mathbb{R}^p$ stored in the node i.

In the expansion of $\mathbf{w}^T \varphi(\mathbf{x})$ above, we have μ_i but no μ_i^2 as in (2) or (3). The reason is that we do not have any inner product between images of $\varphi_i(\cdot)$: this becomes a crucial property for deriving a separable optimization problem.

There have been largely two types of approaches to find approximate feature mappings φ. The first type of approaches is based on computing low-rank factor matrices that approximate the original kernel matrices [4, 3, 6]. Although this type allows to use any positive semidefinite kernel matrices, it requires matrix factorization with comparably large memory footprint.

In the second type, we make use of random projections and construct approximate mappings directly [16]. These methods tend to require larger approximation dimensions than the first type [11], but they are much simpler and easier to parallelize. In this paper we focus on the second type approximation of the Gaussian kernels $k_i(\mathbf{x}[i], \mathbf{x}'[i]) = \exp(-\gamma_i \|\mathbf{x}[i] - \mathbf{x}'[i]\|_2^2)$ for some $\gamma_i > 0$ without loss of generality, for which the approximation is given by

$$\varphi_i(\mathbf{x}[i]) = \sqrt{\frac{2}{d_i}} \left[\cos(\mathbf{z}_1^T \mathbf{x}[i] + e_1), \cos(\mathbf{z}_2^T \mathbf{x}[i] + e_2), \ldots, \cos(\mathbf{z}_{d_i}^T \mathbf{x}[i] + e_{d_i}) \right]^T, \tag{5}$$

for each node i, where $\mathbf{z}_j \in \mathbb{R}^{p_i}$ and $e_j \in \mathbb{R}$ are i.i.d. random samples from the Gaussian distribution $\mathcal{N}(\mathbf{0}, 2\gamma_i I)$, I is an identity matrix, and from the uniform distribution on $[0, 2\pi]$, respectively. These are derived from the Fourier transform of the kernel function k_i (for more details see [16]). Note that φ_i uses only local features stored in the ith node, represented as a subvector $\mathbf{x}[i]$.

2.4 A Separable Surrogate Objective

Our final step is to make the loss functions ℓ in Table 1 separable over nodes, using their convexity in the first and the last arguments. For this purpose we impose $\sum_{i=1}^n \mu_i = 1$ in addition to $\mu_i \geq 0$. Then we can derive an upper bound for the hinge loss ℓ of 2-class classification as follows,

$$\begin{aligned}\ell(\mathbf{w}^T\varphi(\mathbf{x}), y, \rho) &= \max\{0, 1 - y\mathbf{w}^T\varphi(\mathbf{x})\} \\ &= \max\{0, \sum_{i=1}^n \mu_i(1 - y\mathbf{w}[i]^T\varphi_i(\mathbf{x}[i]))\} \\ &\leq \sum_{i=1}^n \mu_i \ell_i\left(\mathbf{w}[i]^T\varphi_i(\mathbf{x}[i]), y, \rho_i\right) \qquad (6)\end{aligned}$$

where ℓ_i is listed in the second row of Table 1, and we define ρ_i so that $\rho = \sum_{i=1}^n \mu_i\rho_i$. The upper bounds for 1-class classification and regression tasks can be derived similarly and are presented in the table as well. Summing up the inequalities (6) over training indices $u = 1, 2, \ldots, m$ leads to an upper bound of the objective function in (4):

$$\begin{aligned}&\frac{\lambda}{2}\left(\|\mathbf{w}\|_2^2 + \rho\right) + \frac{1}{m}\sum_{u=1}^m \ell\left(\mathbf{w}^T\varphi(\mathbf{x}_u), y_u, \rho\right) \\ &\leq \sum_{i=1}^n \left[\frac{\lambda}{2}\left(\|\mathbf{w}[i]\|_2^2 + \mu_i\rho_i\right) + \frac{1}{m}\sum_{u=1}^m \mu_i\ell_i\left(\mathbf{w}[i]^T\varphi_i(\mathbf{x}_u[i]), y_u, \rho_i\right)\right]. \qquad (7)\end{aligned}$$

The expression in the right hand side is separable in terms of nodes. Therefore, in each node $i = 1, 2, \ldots, n$ we can solve the following separated problem,

$$\textbf{(Local)} \quad \min_{\mathbf{w}[i]\in\mathbb{R}^{d_i}, \rho_i\in\mathbb{R}} \frac{\lambda}{2}\left(\|\mathbf{w}[i]\|_2^2 + \mu_i\rho_i\right) + \frac{1}{m}\sum_{u=1}^m \mu_i\ell_i\left(\mathbf{w}[i]^T\varphi_i(\mathbf{x}_u[i]), y_u, \rho_i\right). \qquad (8)$$

Although it is possible to construct a global classifier by transferring all local solutions $\mathbf{w}^*[i]$ and ρ_i^* of (8) to a central node for $i = 1, 2, \ldots, n$, it may not be desirable since then the central node should know about φ_i and local features as well. This requires $\mathcal{O}(\sum_{i=1}^n d_i p_i)$ numbers to be transferred, plus $\mathcal{O}(\sum_{i=1}^n p_i)$ per test point $\mathbf{x}$ whose features are stored in a distributed fashion. Instead, we let each node compute and transfer two scalars $\mathbf{w}^*[i]^T\varphi_i(\mathbf{x}[i])$ and ρ_i^* to a central node, where weighted summations $\mathbf{w}^{*T}\varphi(\mathbf{x}) = \sum_{i=1}^n \mu_i\mathbf{w}^*[i]^T\varphi_i(\mathbf{x}[i])$ and $\rho^* = \sum_{i=1}^n \mu_i\rho_i^*$ can be used for global prediction. This approach reduces the communication cost to $\mathcal{O}(n)$ for a test point.

2.5 Minimization of Approximation Gaps

We discuss the quality of two approximations we have made, in using approximate feature mappings and an inequality due to the convexity of loss functions, to arrive the separated local optimization (8) from the nonseparable SVM formulation (1), assuming that both are using multiple localized kernels.

Approximation in Feature Mappings The first approximation has been applied when we use approximate feature mappings in Section 2.3. In the case of the mapping $\varphi_i : \mathbb{R}^{p_i} \to \mathbb{R}^{d_i}$ in (5) approximating a local Gaussian kernel $k_i(\mathbf{x}[i], \mathbf{x}'[i]) = \exp(-\gamma_i \|\mathbf{x}[i] - \mathbf{x}'[i]\|_2^2)$, $\mathbf{x}[i] \in \mathbb{R}^{p_i}$, the following result from [16] quantifies its quality:

$$\mathbb{P}\left[\sup_{\mathbf{x}[i],\mathbf{x}'[i]\in\mathcal{M}} \left|\varphi_i(\mathbf{x}[i])^T\varphi_i(\mathbf{x}'[i]) - k_i(\mathbf{x}[i],\mathbf{x}'[i])\right| \geq \epsilon\right] \leq \mathcal{O}\left(\epsilon^{-2} e^{-\frac{\epsilon^2 d_i}{4(p_i+2)}}\right),$$

where $\mathcal{M} \subset \mathbb{R}^{p_i}$ is a compact set containing all subvectors $\mathbf{x}_u[i]$, $u = 1, 2, \ldots, m$. Therefore, φ_i grants us good approximation as long as we use sufficiently large d_i for its approximation dimension.

Approximation in the Convex Inequality Another approximation takes place in (6) and (7), where we construct separable upper bounds of the nonseparable loss functions ℓ in Table 1. Since the inequality is constructed using the convex combination parametrized by $\mu_1, \mu_2, \ldots, \mu_n$, we can reduce the gap in the inequality by minimizing the right hand side expression of (7) in terms of μ_i's. This defines an optimization problem in a central node,

$$\begin{aligned}
&\min_{\mu:=(\mu_1,\mu_2,\ldots,\mu_n)^T} \frac{1}{m}\sum_{u=1}^{m}\sum_{i=1}^{n} L_{ui}\mu_i + \Psi(\mu) \\
\textbf{(Central)}\quad &\text{s.t.} \sum_{i=1}^{n} L_{ui}\mu_i \geq \ell\left(\sum_{i=1}^{n} Z_{ui}\mu_i, y_u, \sum_{i=1}^{n}\mu_i\rho_i\right),\ u = 1, 2, \ldots, m, \quad (9)\\
&\sum_{i=1}^{n}\mu_i = 1,\ \ \mu_i \geq 0,\ i = 1, 2, \ldots, n.
\end{aligned}$$

Here we have defined

$$\begin{cases} Z_{ui} & := \mathbf{w}[i]^T\varphi_i(\mathbf{x}_u[i]) \\ L_{ui} & := \ell_i(Z_{ui}, y_u, \rho_i) \end{cases}, \quad u = 1, 2, \ldots, m,\ i = 1, 2, \ldots, n. \qquad (10)$$

The constants in Z_{ui} can be computed independently in local nodes and transferred to the central node.

The last term Ψ in the objective of (9) is an optional convex regularization term. This can be chosen as $\Psi(\mu) = \frac{\sigma}{2}\|\mu\|_2^2$ for some $\sigma > 0$ to produce a unique or an evenly distributed solution, $\Psi(\mu) = \sigma'\|\mu\|_1$ to induce elementwise sparsity

in μ (thereby selecting few local kernels important for prediction, similarly to MKL), or $\Psi(\mu) = \sigma'' \sum_{g=1}^{G} \|\mu[g]\|_2$ for subvectors $\mu[g]$ to promote groupwise sparsity (e.g. selecting few clusters of nodes, rather than individual nodes).

3 Related Works

We present two most closely related learning approaches to our framework.

3.1 Multiple Kernel Learning

The multiple kernel learning (MKL) is an extension of the support vector machines for employing multiple kernel functions, instead of a single one as in the standard settings. The current forms and efficient learning methods for MKL have been established in [10, 9, 1, 2]. In MKL we consider a combination of n kernels

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \beta_i k_i(\mathbf{x}, \mathbf{x}'), \quad \beta_i \geq 0, \quad \sum_{i=1}^{n} \beta_i = 1, \tag{11}$$

and k_i's are defined on a certain subset of features. Plugging the composite kernel $k(\mathbf{x}, \mathbf{x}')$ into the standard SVM formulation leads to a semi-definite program (SDP) [10], which is much harder to solve than the standard SVMs. When kernels k_i are normalized, i.e. $k_i(\mathbf{x}, \mathbf{x}) = 1$, it can be reduced to a quadratically constrained quadratic program [9], which can be solved slightly more efficiently than SDPs. Modifications to the SVM formulations lead to further improvement, resulting in a semi-infinite linear program [19], a quadratic program [17], or a much faster interleaved optimization using ℓ_p-norms [8].

The main difference of MKL to our framework is that the objective function of MKL is not separable over nodes. For instance, the MKL formulation in [17] solves the dual problem for fixed weights $\beta_1, \beta_2, \ldots, \beta_n$,

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & -\frac{1}{2} \sum_{i=1}^{n} \beta_i \sum_{u=1}^{m} \sum_{v=1}^{m} \alpha_u \alpha_v k_i(\mathbf{x}_u, \mathbf{x}_v) + \sum_{u=1}^{m} \alpha_u \\ \text{s.t.} \quad & \sum_{u=1}^{m} \alpha_u y_u = 0, \quad 0 \leq \alpha_u \leq 1/(m\lambda), \; u = 1, 2, \ldots, m. \end{aligned}$$

Similar to our discussion in Section 2.2, all variables α_u's in this objective are coupled with each node i, therefore the optimization cannot be separated over nodes. Another difference is the ways to form the convex combinations of kernels. Comparing the convex combinations in (11) and (2), we can interpret our μ_i as $\sqrt{\beta_i}$, and we impose $\sum_{i=1}^{n} \mu_i = 1$, rather than $\sum_{i=1}^{n} \beta_i = 1$.

3.2 Boosting

Boosting with an additive model [5] is also quite similar to our model and MKL. In boosting, we find a linear combination of n basis functions or weak learners

of the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \zeta_i h_i(\mathbf{w}; \mathbf{x}),$$

where $\zeta_i \in \mathbb{R}$, and h_i can be set $h_i(\mathbf{w}; \mathbf{x}) = \mathbf{w}[i]^T \varphi_i(\mathbf{x}[i])$ to make it similar to our setting. The optimal $\mathbf{w}^*[i]$ and ρ_i^* can be found independently for each node $i = 1, 2, \ldots, n$, and the best combination of h_i's can be found by solving

$$\min_{\zeta_1,\ldots,\zeta_n} \sum_{u=1}^{m} \ell \left(\sum_{i=1}^{n} \zeta_i h_i(\mathbf{w}^*; \mathbf{x}_u), y_u, \sum_{i=1}^{n} \zeta_i \rho_i^* \right).$$

This resembles our central problem (9). Despite its similarity, however, the objective here does not provide an upper bound of the MKL objective as in our formulation (7), thereby losing its connection to MKL. Also, unlike our setting and MKL, the local problems in boosting do not depend on the weights ζ_i. That is, the solutions from separated problems cannot be improved any further using updated weight values of ζ_i obtained from the central optimization. Our framework and MKL share the property that subproblems (separated problems in our case, and the nonseparable problem obtained after fixing kernel weights in MKL) are dependent on such weights, therefore we can obtain improved solutions using adjusted weight values.

4 Algorithm

We describe our algorithm that solves the separated problem (8) at each local node, and an additional central optimization (9) that determines the optimal convex combination. The outline of the entire framework is presented in Algorithm 1.

4.1 Local Optimization

To find the solutions of each separated local optimization problem, we use the stochastic gradient descent (SGD) approach. In particular, we adapt the "robust" version of SGD suggested by Nemirovski and Yudin [15], for which a simplified analysis [14] or a regret-based online learning analysis [21] can be found. We sketch the robust SGD algorithm here and refer to the ASSET approach [11] for details, which implements essentially the same idea for the standard SVMs.

To simplify our discussion, we denote the objective function of the ith separated local problem in (8) by f_i:

$$f_i(\mathbf{w}[i], \rho_i) := \frac{\lambda}{2} \left(\|\mathbf{w}[i]\|_2^2 + \mu_i \rho_i \right) + \frac{1}{m} \sum_{u=1}^{m} \mu_i \ell_i \left(\mathbf{w}[i]^T \varphi_i(\mathbf{x}_u[i]), y_u, \rho_i \right).$$

Then in each iteration of the local optimization, we update the variables $\mathbf{w}[i]$ and ρ_i as follows,

$$\begin{bmatrix} \mathbf{w}[i]^{t+1} \\ \rho_i^{t+1} \end{bmatrix} \leftarrow \mathcal{P}_{\mathcal{W}} \left(\begin{bmatrix} \mathbf{w}[i]^t \\ \rho_i^t \end{bmatrix} - \eta_t G_t \right), \quad t = 1, 2, \ldots, T, \tag{12}$$

Algorithm 1: Separable SVM with Approximations to Local Kernels

input : A data set $\{(\mathbf{x}_u, y_u)\}_{u=1}^m$, the number of iterations K ($K > 1$ only if we use central optimization), positive integers T and T_0, and $\mu_0 = 1/n$.
Initialize: $\mu_i \leftarrow \mu_0$, for $i = 1, 2, \ldots, n$;
for $k = 1, 2, \ldots, K$ **do**
  Transmit μ_i to all nodes $i = 1, 2, \ldots, n$;
  (local: in parallel)
    `// Solve a separated local problem in each node` i`, using ASSET`
    input : a weight μ_i and local measurements/labels $\{(\mathbf{x}_u[i], y_u)\}_{u=1}^m$.
    Initialize iterates and averages: $\mathbf{w}[i]^1 \leftarrow \mathbf{0}$, $\rho_i^1 \leftarrow 0$ $\bar{\mathbf{w}}[i] \leftarrow \mathbf{0}$, $\bar{\rho}_i \leftarrow 0$;
    Estimate an optimization constant $\theta_i > 0$;
    for $t = 1, 2, \ldots, T$ **do**
      Select a random training index $\xi_t \in \{1, 2, \ldots, m\}$;
      Compute a steplength $\eta_t = \theta_i/\sqrt{t}$;
      Update $\mathbf{w}[i]$ and ρ_i via (12);
    end
    output: averages of iterates, $\bar{\mathbf{w}}[i]$ and $\bar{\rho}_i$, for the last $(T - T_0)$ iterations.
    output: (optional) transfer $Z_{ui} := \bar{\mathbf{w}}[i]^T \varphi_i(\mathbf{x}_u)$ for $u = 1, 2, \ldots, m$ and $\bar{\rho}_i$ to a central node.
  (end)
  (central: optional)
    `// Solve a central problem, using CPLEX`
    input : Z_{ui} and $\bar{\rho}_i$ for $u = 1, 2, \ldots, m$, $i = 1, 2, \ldots, n$.
    Compute $L_{ui} := \ell_i(Z_{ui}, y_u, \bar{\rho}_i)$ for all $u = 1, 2, \ldots, m$, $i = 1, 2, \ldots, n$;
    Compute $\rho = \sum_{i=1}^n \mu_i \bar{\rho}_i$;
    Solve an equivalent formulation (13);
    output: new weights $\mu_1, \mu_2, \ldots, \mu_n$.
  (end)
end

where $\mathcal{P}_{\mathcal{W}}(\mathbf{z}) := \arg\min_{\mathbf{v} \in \mathcal{W}} \{\frac{1}{2}\|\mathbf{z} - \mathbf{v}\|_2^2\}$ is an Euclidean projection of a vector $\mathbf{z}$ onto a convex set $\mathcal{W}$, G_t is an estimate subgradient of f_i at $(\mathbf{w}[i]^t, \rho_i^t)$, constructed using a training example chosen by a random index $\xi_t \in \{1, 2, \ldots, m\}$, and η_t is a steplength of the form $\eta_t = \theta_i/\sqrt{t}$ for some $\theta_i > 0$. The set $\mathcal{W}$ guides the optimization to avoid taking too large steps, whose formulation can derived analytically from strong duality [18, 11].

The convergence of the robust SGD algorithm is $\mathcal{O}(c(T_0/T)\theta_i/\sqrt{T})$ in terms of objective function values in expectation, where $c(\cdot)$ is a simple function only depending on the ratio T_0/T [14].

4.2 Central Optimization

The central problem (9), for the loss functions ℓ in Table 1, can be formulated as a linear program (LP) or a quadratic program (QP) depending on the choices of the regularizer Ψ. When we consider 2-class problems with $\Psi(\mu) = \frac{\sigma}{2}\|\mu\|_2^2$ for

$\sigma \geq 0$, we can write an equivalent formulation to (9) as follows,

$$\begin{aligned} \min_{\mu \in \mathbb{R}^n} \ & \frac{1}{m} \mathbf{1}_m^T L \mu + \frac{\sigma}{2} \mu^T \mu, \\ \text{s.t.} \ & (L + D_y Z)\mu \geq \mathbf{1}_m, \ \ \mathbf{1}_n^T \mu = 1, \ \ \mu \geq \mathbf{0}, \end{aligned} \tag{13}$$

where the elements of the matrices $L \in \mathbb{R}^{m \times n}$ and $Z \in \mathbb{R}^{m \times n}$ are defined in (10), D_y is a diagonal matrix with elements $y_1, y_2, \ldots, y_m$, and $\mathbf{1}_m := (1, 1, \ldots, 1)^T \in \mathbb{R}^m$. Similar formulations can be derived for 1-class and regression tasks.

The solutions of (13) can be obtained using LP solvers when $\sigma = 0$, or using QP solvers for $\sigma > 0$. In our experiments we use $\sigma = 0.5$, since it has produced slightly better solutions than using $\sigma = 0$. For the solution method we adopt the IBM ILOG CPLEX Optimization Studio Academic Research Edition v.12.4, which provides one of the fastest LP/QP solvers for free for academic institutes.

The total number of elements to be transferred to a central node is $\mathcal{O}(m)$ for each node $i = 1, 2, \ldots, n$. (These elements compose the matrix Z.) This cost can be reduced, trading some potential prediction improvement, by transferring information for a small subsample of size $m' < m$, rather than for the entire training set of size m. This also reduces the number of constraints in the central problem (13) from $\mathcal{O}(m)$ to $\mathcal{O}(m')$. We have used $m' = 5000$ for our experiments.

We set the maximum number of central optimization to $K = 10$, stopping the algorithm earlier if the prediction performance on m' training samples does not improve any further. (Three passes were enough in most cases.)

5 Experiments

We implemented Algorithm 1 based on the open-source `C++` program ASSET [11][1], comparing several different implementations built upon it:

- `Separated`: implements Algorithm 1.
- `Composite`: the standard SVM with a composite kernel (2) consisting of local kernels. We set $\mu_i = \frac{1}{n}$ for all $i = 1, 2, \ldots, n$.
- `Single`: the standard SVM with a single global kernel that uses all features.

All of these make use of approximations to the kernel feature mappings. We also use `SVMLight` with its default parameters to make comparisons to the cases using exact kernel information.

In all runs, we randomly partition the set of features into equal-sized n groups and assign each group to one of n nodes. We use an *overlap* parameter to specify the percentage of features in each node that are sampled from other nodes, simulating peer-to-peer information exchange among nodes. The purpose of such communication will be amending the loss of information due to partitioning.

We use Gaussian kernels of the form $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|_2^2)$ in all experiments, where the parameter γ is tuned by a cross validation using `SVMLight` [7]

[1] Available at `http://pages.cs.wisc.edu/~sklee/asset/`.

Table 2. Data sets and their training parameters.

Name	m (train)	test	p	(density)	λ	γ
ADULT	40701	8141	124	(11.2%)	3.07e-08	0.001
MNIST	58100	11900	784	(19.1%)	1.72e-07	0.01
CCAT	89702	11574	47237	(1.6%)	1.28e-06	1.0
IJCNN	113352	28339	22	(56.5%)	8.82e-08	1.0
COVTYPE	464809	116203	54	(21.7%)	7.17e-07	1.0

for real-world data sets. The parameter λ is tuned in the same way, and both are shown in Table 2. For artificial data we use $\lambda = 0.133$ and $\gamma = 0.001$ found by the `Single` code. Whenever we have localized kernels $k_i = \exp(-\gamma_i \|\mathbf{x}[i] - \mathbf{x}'[i]\|_2^2)$, we set their parameters by $\gamma_i = n\gamma$. The purpose here is to compensate the difference between the orders of magnitude $\|\mathbf{x}[i] - \mathbf{x}'[i]\|_2^2 \in \mathcal{O}(p_i)$ and $\|\mathbf{x} - \mathbf{x}'\|_2^2 \in \mathcal{O}(p)$, in a way that makes the arguments for exponential functions similar, i.e.

$$\gamma_i = \frac{p}{p_i}\gamma \;\approx\; n\gamma \;\Rightarrow\; \gamma_i \|\mathbf{x}[i] - \mathbf{x}'[i]\|_2^2 \in \mathcal{O}(\gamma p).$$

For creating approximate feature mappings for kernels, we set their dimensions to $d = 1000$ and $d_i \approx d/n$ for all experiments.

All experiments have been performed on 64-bit multicore Linux systems, where a thread is created to simulate a node optimizing a separated objective.

5.1 Data

We use an artificial data set and five real-world benchmark data sets.

Artificial Data A data set is created by sampling 7500 (training) and 2500 (testing) $p = 64$ dimensional random vectors from two multivariate Gaussian distributions, $\mathcal{N}_-(\eta_-, \Sigma)$ and $\mathcal{N}_+(\eta_+, \Sigma)$ for two classes. We fix $\eta_- = (-1, \ldots, -1)^T$ and $\eta_+ = (1, \ldots, 1)^T$. The two distributions share a covariance matrix Σ, which is constructed with controlling the maximum number of nonzero entries (the ratio is specified by the *correlation ratio* $r \in [0, 1]$). To construct a positive semidefinite matrix Σ, we first sample a random matrix $S \in \mathbb{R}^{p \times p}$, computing its QR decomposition, $S = QR$. Then we replace a fraction r of the rows of Q by normalized random vectors of length p. Finally we set $\Sigma = QQ^T$, so that Σ will contain up to $p(rp + (1 - r)(rp + 1))$ nonzero entries.

Real-World Data Sets Five real-world benchmark data sets[2] in Table 2 are prepared as follows. `ADULT` is from the UCI machine learning repository, randomly split into training and test sets. `MNIST` is prepared for classifying the digits 0-4 from 5-9. `CCAT` is from the RCV1 collection, classifying the category

[2] Available at the UCI Repository `http://archive.ics.uci.edu/ml/`, or at the LIBSVM website `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

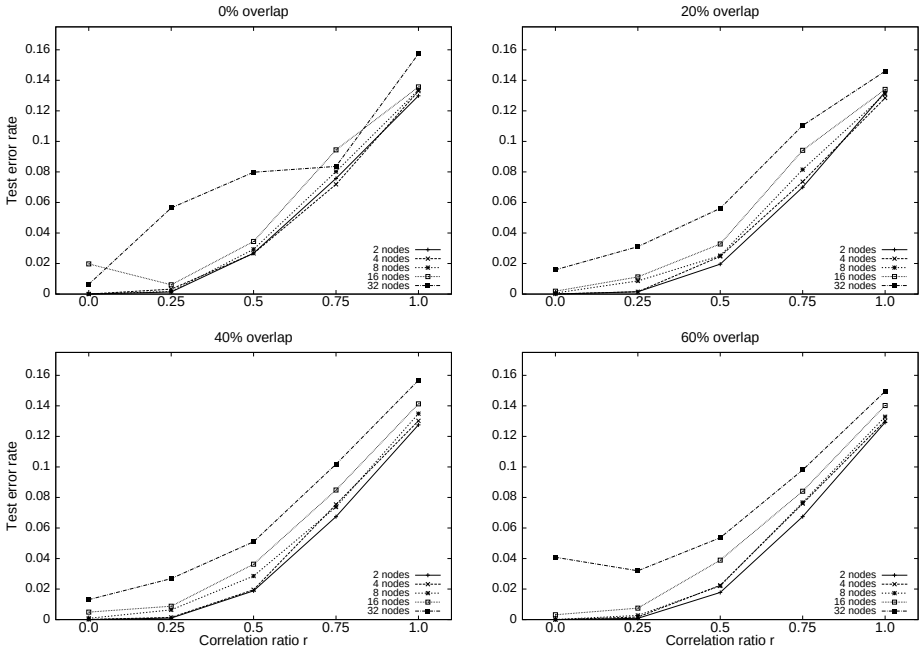

Fig. 1. Test error rates for different overlaps of features (100%: all features are available at each local node), numbers of nodes, and correlation ratios (1.0: nonzero correlation between all pairs of features). All measurements are averages over 20 runs with random splits of features and random approximations of φ.

CCAT from the others, where we use the original test set as our training set and the original training set as our test set. `IJCNN` is from the IJCNN 2001 Challenge data set. `COVTYPE` classifies type 1 against the other forest cover types.

5.2 Artificial Data

Locality of information vs. prediction performance We first evaluated how the locality of features affects the prediction performance of our algorithm using localized kernels.

In Figure 1, we show the test error rates for different overlaps of attributes, numbers of nodes, and correlation ratio r (averaged values over 20 repetitions using randomized splits of features and approximations of φ). In each plot, for a fixed number of nodes, say $n = 8$, we can check that the error rate increases as the correlation ratio increases. This is something expected, since as more features are correlated, we are likely to lose more information by partitioning features into groups and treating them separately.

For a fixed correlation ratio, say $r = 0.5$ (the midpoints of the four plots in Figure 1), the error rate tends to increase with the number of nodes. This will be also due to the loss of information by partitioning. But it seems that such

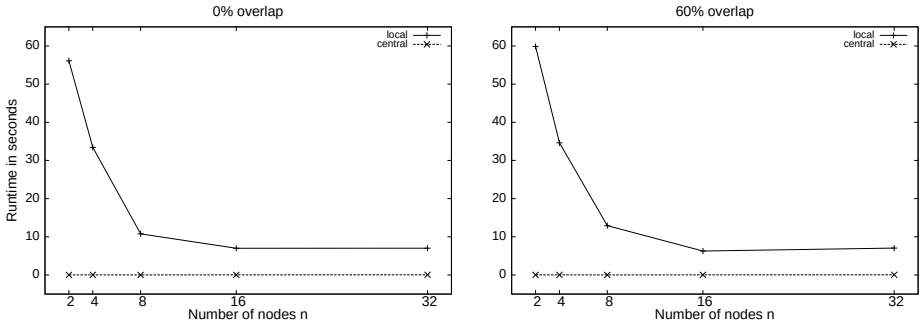

Fig. 2. Runtime (in seconds) at the central node and average runtime (in seconds) for the local optimizations, for different overlaps of features (100%: all features are available at each local node) and numbers of nodes. The correlation ratio is fixed at $r = 0.25$. All values are averages over 20 runs with random splits of features and random approximations of φ.

loss could be compensated by providing nodes some "overlapping" features from other nodes, which can be observed as we scan through the top left, top right, bottom left, and bottom right plots, for 32 nodes with correlation ratio $r = 0.5$ for instance.

Scalability using multiple nodes Figure 2 shows the average runtime (in seconds) taken in the local and the central optimization of our algorithm. Since the runtime values are not affected much by the correlation ratio r, we show only the plots for $r = 0.25$ here.

The runtime for the local optimization keeps improving as we use more nodes up to $n = 16$, since all optimizations can be done in parallel (the machine had 32 physical cores). Considering the test error rates reported in the corresponding plots (top left and bottom right) in Figure 1, at correlation ratio $r = 0.25$, using more nodes would not harm too much the prediction performance. An exception will be $n = 32$, which seems to make each group too small, deteriorating both scalability and accuracy by a noticeable amount. The runtime for central optimization was almost negligible in this case. The optimization took slightly longer for 60% overlap than the case of no overlap, since the former had to handle larger number of attributes.

5.3 Benchmark on Real-World Data Sets

For benchmark we fix the number of nodes to $n = 8$, since it has showed a good accuracy and speed tradeoff in the experiments with our artificial data. In local optimization, we set the number of SGD iterations to $10m$, ten times of the number of training examples.

Table 3 shows the runtime and test error rate values over 20 repeated runs, except for `CCAT` where we use 12 runs due to its long runtime, of all methods

Table 3. Training CPU time (in seconds, h:hours) and test error rate (mean and standard deviation %) in parentheses. The features of input vectors are distributed in two different ways: (i) each node contains disjoint set of features (no overlap), or (ii) each node has 25% of its features as copies from other nodes (25% overlap).

No Overlap	ASSET-SVM				SVMLight
	Separated		Composite	Single	
	+ Central	− Central			
ADULT	28(20.0±0.02)	9(20.6±1.17)	44(15.5±0.64)	235(15.7±0.74)	966(15.1)
MNIST	101(11.1±0.39)	87(11.1±0.39)	539 (7.0±0.72)	1539 (7.0±0.45)	1031 (1.1)
CCAT	1h(26.3±1.00)	1h(26.3±1.00)	8h(20.9±0.63)	-	2h (4.2)
IJCNN	67 (9.1±0.88)	20 (9.5±0.07)	86 (4.1±0.53)	177 (1.6±0.13)	687 (0.7)
COVTYPE	234(29.3±2.76)	82(35.2±1.05)	373(21.1±0.61)	938(18.0±0.78)	23h (7.4)

25% Overlap	ASSET-SVM				SVMLight
	Separated		Composite	Single	
	+ Central	− Central			
ADULT	28(19.0±1.69)	9(19.7±1.34)	48(15.5±0.49)	235(15.7±0.74)	966(15.1)
MNIST	112(11.1±0.39)	94(12.1±0.72)	486 (7.3±0.50)	1539 (7.1±0.45)	1031 (1.1)
CCAT	2h(29.5±1.01)	2h(29.5±1.01)	10h(23.8±0.80)	-	2h (4.2)
IJCNN	75 (8.6±1.05)	20 (9.4±0.63)	107 (3.8±0.56)	177 (1.6±0.13)	687 (0.7)
COVTYPE	219(29.6±2.76)	94(33.7±1.34)	466(20.8±0.63)	938(18.0±0.78)	23h (7.4)

for the five benchmark data sets, without and with 25% overlap of features. In each run we randomize the partitioning of features and the projections for constructing approximate feature mappings, if applicable. For `Single` ASSET, the overlap parameter has no effect, so the results are copied in both tables for readability. The results for `Single` on `CCAT` are unavailable for its impractically long runtime.

Gap from the separation of optimization We first compare the results of `Separated` and `Composite` in Table 3. The difference here occurs because of our construction of separable surrogate objective functions using the convex inequality in (7).

Comparing to the third column (`Composite`) of Table 3, the test error rates in the second column (`Separated` without central optimization) has been increased by 1.63 (no overlap) and 1.65 (25% overlap) times on average. Considering that `Composite` has the access to all feature information, through a single composite kernel consisting of all localized kernels, such increments seem to be moderate. In `Separated`, features and optimizations are distributed among the nodes, and thereby the SVM can be solved in much shorter time (about 5 times faster on average) but sacrificing accuracy.

Improvement by central optimization The first two columns of Table 3 show the potential improvement and cost of an additional central optimization. The improvements in test error rates seem to be marginal ($6 \sim 7\%$), but recall that these are obtained using a small subsample (5000) from each training set

for the central problem, rather than using the entire set, simulating a limited communication bound.

Gap from using localized kernels In the third and fourth columns of Table 3, the information of all features is accessed through either a composite kernel consisting of local kernels (`Composite`), or a single kernel using all features directly (`Single`). The `Composite` approach is very similar to the standard MKL, except that here we are using fixed weights among the local kernels. As we can see, the performance in terms of error rates is not very different in these two approaches.

The overall runtime of `Composite` is shorter than `Single`, although they have essentially the same time complexity. The savings in `Composite` might have come from the fact that the input vectors are sparse in our benchmark sets, where subvectors of them tend to be more sparse, reducing the time for projections on random directions in constructing approximate feature mappings.

Gap from approximating kernels The difference in the last two columns of Table 3 is resulted from that the feature mappings of kernels are approximated in `Single`, whereas `SVMLight` uses exact kernels. The error rates of `Single` are comparable in most cases, except for `CCAT` and `COVTYPE`. We believe the results will improve with larger approximation dimensions in general. Also, we can consider using different types of approximations for `CCAT`, and using more iterations in the local optimization for `COVTYPE`. We refer to [16, 11] for more extensive comparison in this respect.

6 Conclusion

We suggest a separable optimization framework for solving the support vector machines on distributed sensor measurements, minimizing communication cost to construct a global predictor. While sacrificing some accuracy, our framework provides a transparent way to derive a separable function that becomes an upper bound of the original SVM objective, based on the convexity of loss functions and approximations to kernel feature mappings.

Acknowledgements

The authors acknowledge the support of Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project B3 and C1.

References

1. Bach, F.R., Lanckriet, G.R.G., Jordan, M.I.: Multiple kernel learning, conic duality, and the SMO algorithm. In: Proceedings of the 21st International Conference on Machine Learning (2004)

2. Bi, J., Zhang, T., Bennett, K.P.: Column-generation boosting methods for mixture of kernels. In: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 521–526 (2004)
3. Drineas, P., Mahoney, M.W.: On the nystrom method for approximating a gram matrix for improved kernel-based learning. Journal of Machine Learning Research 6, 2153–2175 (2005)
4. Fine, S., Scheinberg, K.: Efficient svm training using low-rank kernel representations. Journal of Machine Learning Research 2, 243–164 (2001)
5. Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning. Springer (2001)
6. Joachims, T., Finley, T., Yu, C.N.: Cutting-plane training of structural svms. Machine Learning 77(1), 27–59 (2009)
7. Joachims, T.: Making large-scale support vector machine learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning, chap. 11, pp. 169–184. MIT Press, Cambridge, MA (1999)
8. Kloft, M., Brefeld, U., Sonnenburg, S., Zien, A.: ℓ_p-norm multiple kernel learning. Journal of Machine Learning Research 12, 953–997 (2011)
9. Lanckriet, G.R.G., De Bie, T., Cristianini, N., Jordan, M.I., Noble, W.S.: A statistical framework for genomic data fusion. Bioinformatics 20(16), 2626–2635 (2004)
10. Lanckriet, G., Cristianini, N., Bartlett, P., E.G., L., Jordan, M.: Learning the kernel matrix with semidefinite programming. In: Proceedings of the 19th International Conference on Machine Learning (2002)
11. Lee, S., Wright, S.J.: ASSET: Approximate stochastic subgradient estimation training for support vector machines. In: International Conference on Pattern Recognition Applications and Methods (2012)
12. Lippi, M., Bertini, M., Frasconi, P.: Collective traffic forecasting. In: Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part II. pp. 259–273 (2010)
13. Morik, K., Bhaduri, K., Kargupta, H.: Introduction to data mining for sustainability. Data Mining and Knowledge Discovery 24(2), 311–324 (2012)
14. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM Journal on Optimization 19(4), 1574–1609 (2009)
15. Nemirovski, A., Yudin, D.B.: Problem complexity and method efficiency in optimization. John Wiley (1983)
16. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems, vol. 20, pp. 1177–1184. MIT Press (2008)
17. Rakotomamonjy, A., Bach, F., Canu, S., Grandvalet., Y.: More efficiency in multiple kernel learning. In: Proceedings of the 24th international conference on machine learning (2007)
18. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. Mathematical Programming, Series B 127(1), 3–30 (2011)
19. Sonnenburg, S., Rätsch, G., Schäfer, C., Schölkopf, B.: Large scale multiple kernel learning. Journal of Machine Learning Research 7, 1531–1565 (July 2006)
20. Wahba, G.: Splines Models for Observational Data, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 59. SIAM (1990)
21. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the 20th International Conference on Machine Learning. pp. 928–936 (2003)