2-2018

Sparse Passive-Aggressive learning for bounded online kernel methods

Jing LU
Singapore Management University, jing.lu.2014@phdis.smu.edu.sg

Doyen SAHOO
Singapore Management University, doyens@smu.edu.sg

Peilin ZHAO
South China University of Technology

Steven C. H. HOI
Singapore Management University, CHHOI@smu.edu.sg

DOI: https://doi.org/10.1145/3156684

Citation

LU, Jing; SAHOO, Doyen; ZHAO, Peilin; and HOI, Steven C. H.. Sparse Passive-Aggressive learning for bounded online kernel methods. (2018). *ACM Transactions on Intelligent Systems and Technology*. 9, (4), 45:1-27. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/3999

Sparse Passive-Aggressive Learning for Bounded Online Kernel Methods

JING LU and DOYEN SAHOO, School of Information Systems, Singapore Management University, Singapore
PEILIN ZHAO, School of Software Engineering, South China University of Technology, China
STEVEN C. H. HOI, School of Information Systems, Singapore Management University, Singapore

One critical deficiency of traditional online kernel learning methods is their unbounded and growing number of support vectors in the online learning process, making them inefficient and non-scalable for large-scale applications. Recent studies on scalable online kernel learning have attempted to overcome this shortcoming, e.g., by imposing a constant budget on the number of support vectors. Although they attempt to bound the number of support vectors at each online learning iteration, most of them fail to bound the number of support vectors for the final output hypothesis, which is often obtained by averaging the series of hypotheses over all the iterations. In this article, we propose a novel framework for bounded online kernel methods, named "Sparse Passive-Aggressive (SPA)" learning, which is able to yield a final output kernel-based hypothesis with a bounded number of support vectors. Unlike the common budget maintenance strategy used by many existing budget online kernel learning approaches, the idea of our approach is to attain the bounded number of support vectors using an efficient stochastic sampling strategy that samples an incoming training example as a new support vector with a probability proportional to its loss suffered. We theoretically prove that SPA achieves an optimal mistake bound in expectation, and we empirically show that it outperforms various budget online kernel learning algorithms. Finally, in addition to general online kernel learning tasks, we also apply SPA to derive bounded online multiple-kernel learning algorithms, which can significantly improve the scalability of traditional Online Multiple-Kernel Classification (OMKC) algorithms while achieving satisfactory learning accuracy as compared with the existing unbounded OMKC algorithms.

CCS Concepts: • **Theory of computation → Online learning theory**; • **Computing methodologies → Kernel methods**;

Additional Key Words and Phrases: Online learning, kernel methods, online multiple-kernel learning

ACM Reference format:
Jing Lu, Doyen Sahoo, Peilin Zhao, and Steven C. H. Hoi. 2018. Sparse Passive-Aggressive Learning for Bounded Online Kernel Methods. *ACM Trans. Intell. Syst. Technol.* 9, 4, Article 45 (January 2018), 27 pages.
https://doi.org/10.1145/3156684

1 INTRODUCTION

Online learning with kernels represents an important family of machine-learning techniques for learning nonlinear hypotheses in large-scale machine-learning tasks [17, 23, 35]. Due to the

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its International Research Centres in Singapore Funding Initiative.
Authors' addresses: J. Lu, D. Sahoo, and S. C. H. Hoi, School of Information Systems, Singapore Management University, Singapore; emails: {jing.lu.2014, doyens, chhoi}@smu.edu.sg; P. Zhao, School of Software Engineering, South China University of Technology, Guangzhou, China; email: peilinzhao@hotmail.com.

curse of kernelization [45], a major deficiency of many online kernel learning techniques is their unbounded number of support vectors, which can explode for large-scale learning applications. This has raised a huge challenge when applying these techniques in practical applications, since computational complexity (of both time and space) for an online kernel learning algorithm is proportional to the support vector size.

Recent years have witnessed a variety of emerging studies for budget online kernel learning. Examples include Budget Perceptron [11], Randomized Budget Perceptron (RBP) [4], Forgetron [14], Projectron [30], Budget Passive-Aggressive (BPA) learning [47], Bounded Online Gradient Descent (BOGD) [45, 53], among others. Although budget online kernel learning has been actively studied, most existing algorithms suffer from two key drawbacks as follows.

First, existing budget online kernel learning algorithms are not suitable for online-to-batch conversion, a process that aims to convert online classifiers for batch classification purposes [5, 13, 15]. Specifically, one of the most commonly used approaches for online-to-batch conversion is to take the *averaging classifier*, that is the mean of all the online classifiers at every online iteration, as the final classifier for batch classification. This simple technique is not only computationally efficient, but also enjoys theoretical superiority in generalization performance compared to the classifier obtained in the last online iteration [38]. Unfortunately, most existing budget online kernel learning algorithms only guarantee that the support vector size at each online iteration is bounded, but they fail to yield a sparse averaging classifier after online-to-batch conversion.

Second, conventional budget online kernel learning algorithms may not be effective for bounding the support vector size of the kernel classifiers learned by OMKL [20, 28], a technique that learns multiple-kernel classifiers and their linear combination weights in the online learning process simultaneously. Although one may apply an existing budget online algorithm to bound each individual single-kernel classifier in OMKL with a uniform budget, such a simple approach is not desirable, since it treats all the kernels equally and wastes resources in learning with poor kernels, which could result in a large total budget due to many kernels (or equally very poor learning performance if the given total budget is small).

In this article, we present a novel method for bounded online kernel learning method, named "Sparse Passive-Aggressive" (SPA) learning, which extends the Passive-Aggressive (PA) learning method [10] to the context of bounded online kernel learning. Specifically, the basic idea of our method is to explore a simple yet effective stochastic sampling strategy, which turns an incoming training example into a new Support Vector (SV) with a probability proportional to the loss suffered by the example. Unlike many other existing budget online kernel learning methods, our algorithm ensures that the final output averaging classifier has a bounded number of support vectors after online-to-batch conversion. In addition, this algorithm can accelerate OMKL for classification by not only limiting the number of SVs but also focusing on the best kernel function in an online fashion. We theoretically prove that the proposed algorithm not only bounds the number of support vectors but also achieves an optimal mistake bound in expectation, in both single-kernel and multiple-kernel settings. Finally, we conduct an extensive set of empirical studies that show that the proposed algorithm outperforms a variety of budget online kernel learning algorithms. Note that a short version of this journal manuscript was published in our previous conference article [25]. However, this journal manuscript has been extended significantly and rewritten carefully by including a substantial amount of new content.

The rest of this article is organized as follows. Section 2 reviews the background and related work. Section 3 formulates the problem and then presents the proposed SPA algorithm. Section 4 is an extension of the SPA algorithm to address the bounded OMKL problem. Section 5 gives our theoretical analysis. Section 6 presents our experimental studies and empirical observations, and finally Section 7 concludes this article.

2 RELATED WORK

Our work is closely related to three major categories of machine-learning studies in literature: online learning, kernel methods, and multiple-kernel learning. Below, we briefly review some representative related work in each category.

2.1 Online Learning

Online learning represents a family of efficient and scalable machine-learning algorithms [5, 10, 12, 21, 32]. Unlike conventional batch learning methods that assume all training instances are available prior to the learning task, online learning algorithms receive data in a stream and repeatedly update the predictive models sequentially, which are more appropriate for large-scale real-world applications. In literature, a variety of online learning methods have been proposed under varied settings. In the following, we focus on reviewing some important work in the context of online classification tasks. More comprehensive surveys of online learning can be found in Reference [36].

For (online) classification tasks, a classical online learning technique is the Perceptron algorithm [17, 32], which updates the model by adding a new example with some constant weight onto the current model when the example is misclassified. Another notable and important technique in this category is the online PA learning method [10], which updates the classification function when a new example is misclassified or its classification score does not exceed some predefined margin. PA algorithms are very successful and popular for solving many real-world applications.

More generally, the popular family of online learning algorithms for online classification, referred to as "online linear learning," learn a linear predictive model in the input feature space. The key limitation of these algorithms lies in that the linear model sometimes is restricted to making effective classification only if training data are linearly separable in the input feature space, which is not a common scenario for many real-world classification tasks. This limitation motivates the studies of "online kernel learning" [23], which aims to learn kernel-based predictive models for solving the challenging tasks of classifying sequentially arriving linearly nonseparable instances.

2.2 Kernel Methods

Kernel methods [19, 39, 40], such as Support Vector Machine (SVM) [9, 34], have proven to be powerful for many problems. With the application of kernel functions, which could be regarded as the inner product between two instances in high dimensional feature space, kernel methods can learn nonlinear hypothesis for complicated patterns without explicit computation in the high dimensional feature space.

Earlier studies on kernel methods usually follow the batch setting [3], where all the instances are available prior to training. Along with the increasing demand for learning with stream data in real-time applications, there are increasing studies on efficient kernel learning methods in online settings. Some pioneering works in online kernel learning [23, 52] usually deal with relatively small datasets because of "the curse of kernelization" [45]. Namely, the number of SV's grows linearly with the number of received instances, which results in a large amount of memory cost for storing SVs and a high computational cost for prediction per iteration, making it unsuitable for large-scale applications.

This shortcoming motivates the active studies of bounded online kernel learning [11], which aims to bound the number of SVs in online learning. One common strategy is through budget online learning, which adopts some budget maintenance strategies to impose a budget on the total number of SVs, such as SV removal (Randomized Budget Perceptron (RBP) [4], Forgetron [14], Bounded Online Gradient Descent (BOGD) [53]), SV projection (Projectron [30], Budget Passive-Aggressive Projection [47]), and SV merging (Twin Support Vector Machine [46]). While budget algorithms have shown encouraging scalability to pure online classification tasks, they are not suitable for online-to-batch conversion, since the averaging classifier over all iterations is not sparse.

Recently, some emerging work has attempted to address the above challenge, which are able to yield a sparse output classifier [50, 51]. However, one of the main limitations with the existing work is that their settings are restrictive by assuming only several smooth loss functions. Such settings are restrictive for many scenarios (e.g., SVM) where non-smooth loss functions (e.g., hinge loss) are commonly used. Unlike the restrictive assumptions by Reference [51], we assume more relaxed settings with non-smooth loss functions. Besides, compared with the existing work in Reference [51], our theoretical analyses are cleaner, simpler, and easier to understand by following standard theoretical analysis of online learning algorithms.

2.3 Multiple-Kernel Learning

Multiple-Kernel Learning (MKL) [2, 24] aims to find the optimal (linear) combination of a pool of predefined kernel functions in learning kernel-based predictive models. In comparison to single-kernel learning, MKL not only is able to avoid heuristic manual selection of best kernels, but it also is able to achieve better performance whenever there are multiple kernels that are complementary for training a better predictive model by combining them, particularly for learning from data with heterogeneous representations. MKL has achieved great successes in many applications, ranging from multimedia [48], signal processing [42], biomedical data fusion [49], mobile app mining [7, 8], and beyond.

Although batch MKL methods have been extensively studied recently [1, 18, 31, 41, 44], unfortunately, the generalization from batch MKL to its online counterpart is far from straightforward. First, different from batch MKL, which can in principle be solved via cross-validation, online learning with multiple kernels [20, 22] has no foresight on the best kernel function before data arrival but needs to learn kernel classifiers and their combination weights simultaneously from sequential data. Second, online learning with multiple kernels suffers more from the curse of kernelization, because more kernel classifiers getting updated would result in even greater complexity and higher computation cost.

Recent years have witnessed a variety of emerging studies that successfully addressed the first challenge of OMKL and learnt effective multiple-kernel classifiers from data stream [20, 22, 26–29, 33, 48]. Despite the active explorations, it remains an open challenge of making these algorithms scalable for large-scale applications. In this work, we aim to study bounded online kernel learning to ensure the number of SVs bounded and therefore make the resulting OMKL algorithm scalable. To the best of our knowledge, few existing works have solved the problem of bounded online kernel learning with multiple kernels in a systematic way.

3 SPARSE PASSIVE-AGGRESSIVE LEARNING WITH KERNELS

In this section, we first formulate the problem of online learning with kernels, then review the online PA algorithm [10], and finally present the proposed SPA algorithm for bounded online kernel learning.

3.1 Problem Setting and Preliminaries

We consider the problem of online learning by following online convex optimization settings. Our goal is to learn a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where instance $\mathbf{x}_t \in \mathbb{R}^d$ and class label $y_t \in \mathcal{Y}$. We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \rightarrow \mathbb{R}\}$. We will use $\ell(f; (\mathbf{x}, y)) : \mathcal{H} \times (\mathbb{R}^d \times \mathcal{Y}) \rightarrow \mathbb{R}$ as the loss function that penalizes the deviation of estimating $f(\mathbf{x})$ from observed labels y. Further, we consider $\mathcal{H}$ a Reproducing Kernel Hilbert Space (**RKHS**) endowed with a kernel function $\kappa(\cdot,\cdot):\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ [43] implementing the inner product$\langle\cdot,\cdot\rangle$ such that: (1) κ has the reproducing property $\langle f,\kappa(\mathbf{x},\cdot)\rangle = f(\mathbf{x})$ for $\mathbf{x}\in\mathbb{R}^d$; (2) $\mathcal{H}$ is the closure of the span of all $\kappa(\mathbf{x},\cdot)$ with $\mathbf{x}\in\mathbb{R}^d$, that is, $\kappa(\mathbf{x},\cdot)\in\mathcal{H}\ \forall\mathbf{x}\in\mathcal{X}$. The inner product $\langle\cdot,\cdot\rangle$ induces a norm on $f\in\mathcal{H}$ in the usual way: $\|f\|_{\mathcal{H}} := \langle f,f\rangle^{\frac{1}{2}}$. To make it clear, we denote by $\mathcal{H}_\kappa$ an RKHS with explicit dependence on kernel κ. Throughout the analysis, we assume $\kappa(\mathbf{x},\mathbf{x})\le X^2\ \forall\mathbf{x}\in\mathbb{R}^d$.

PA algorithms [10] are a family of margin-based online learning algorithms, which can achieve a bound on the cumulative loss comparable with the smallest loss that can be attained by any fixed hypothesis. Specifically, consider an online binary classification task with label set $\mathcal{Y}=\{-1,+1\}$, and the widely used hinge loss function:

$$\ell(f;(\mathbf{x},y)) = [1-yf(\mathbf{x})]_+,$$

where $[z]_+=\max(0,z)$. We denote $\ell_t(f)=\ell(f;(\mathbf{x}_t,y_t))$ for simplicity. The PA algorithm sequentially updates the online classifier. At the tth iteration, the online hypothesis will be updated by the following strategy:

$$f_{t+1} = \arg\min_{f\in\mathcal{H}_\kappa}\frac{1}{2}\|f-f_t\|^2_{\mathcal{H}_\kappa}+\eta\ell_t(f),$$

where $\eta>0$ is the learning rate parameter. This optimization trades off between two desires: (i) keeping the new function close to the existing one, (ii) minimizing the loss of the new function on the current example.

This optimization problem enjoys a simple closed-form solution:

$$f_{t+1}(\cdot) = f_t(\cdot)+\tau_t y_t\kappa(\mathbf{x}_t,\cdot), \tau_t = \min(\eta, \frac{\ell_t(f_t)}{\kappa(\mathbf{x}_t,\mathbf{x}_t)}).$$

We further rewrite the classifier as

$$f_{t+1}(\cdot) = \sum_{s=1}^{t}\tau_s y_s\kappa(\mathbf{x}_s,\cdot),$$

and whenever $\ell_s\neq 0$, we have $\tau_s\neq 0$.

Obviously, like conventional online kernel learning methods, the critical limitation of PA is the unbounded number of SVs. This limitation makes online kernel learning extremely computationally expensive in both time and memory cost for large-scale applications.

Some existing works have attempted to learn a bounded classifier at each iteration using some budget maintenance strategy (e.g., replacing an old SV when adding a new one). Think about the toy example below, where the budget size is set to $B=3$. At time t, we have the classifier,

$$f_t = \tau_1y_1\kappa(\mathbf{x}_1,\cdot)+\tau_2y_2\kappa(\mathbf{x}_2,\cdot)+\tau_3y_3\kappa(\mathbf{x}_3,\cdot).$$

After the update, a new support $\mathbf{x}_4$ is added to the classifier. To maintain the budget size $B=3$, an existing SV should be discarded, and we get a new classifier:

$$f_{t+1} = \tau_2y_2\kappa(\mathbf{x}_2,\cdot)+\tau_3y_3\kappa(\mathbf{x}_3,\cdot)+\tau_4y_4\kappa(\mathbf{x}_4,\cdot).$$

In the iteration $t+1$, we add a new support vector $\mathbf{x}_5$ and discard another existing SV and get

$$f_{t+2} = \tau_3y_3\kappa(\mathbf{x}_3,\cdot)+\tau_4y_4\kappa(\mathbf{x}_4,\cdot)+\tau_5y_5\kappa(\mathbf{x}_5,\cdot).$$

Note that in many online learning tasks, especially in Online-to-Batch Conversion, we need an averaged classifier over all iterations, which has been proven to have higher accuracy than the classifier in the last iteration f_T. Now, we try to average the classifiers in the toy example above and get

$$\begin{aligned} f_{average} &= \frac{1}{3}(f_t + f_{t+1} + f_{t+2}) \\ &= \frac{\tau_1 y_1}{3}\kappa(\mathbf{x}_1, \cdot) + \frac{2\tau_2 y_2}{3}\kappa(\mathbf{x}_2, \cdot) + \tau_3 y_3 \kappa(\mathbf{x}_3, \cdot) + \frac{2\tau_4 y_4}{3}\kappa(\mathbf{x}_4, \cdot) + \frac{\tau_5 y_5}{3}\kappa(\mathbf{x}_5, \cdot). \end{aligned}$$

Here, we find the problem of most of the existing online kernel learning algorithms. Although the number of SVs in each iteration is bounded, the number of SVs for the averaged classifier over all the learning iterations is often not bounded. The averaged classifier contains the union set of SV sets in all iterations, which is exactly all the SVs appeared in the classifier, including the discarded ones.

This drawback limits the application of existing budget online kernel learning algorithms for online-to-batch conversion tasks where the output final classifier is typically obtained by averaging the classifiers obtained at every learning step. Furthermore, even in a pure online setting, where the aim is not to achieve a good generalization accuracy on test set but to achieve accurate predictions along the online learning process, the prediction using the averaging classifier, i.e., $\frac{1}{t}\sum_{i=1}^{t} f_i(\mathbf{x}_t)$ usually outperforms a single online classifier $f_t(\mathbf{x}_t)$.

Our motivation is to build an algorithm whose averaged classifier is also bounded, so that we can use the averaged classifier instead of the f_T to get higher accuracy.

3.2 Sparse Passive-Aggressive Algorithm (SPA)

To overcome the above limitation, we propose a novel SPA learning algorithm, which not only guarantees that the kernel classifier is bounded at each iteration but also ensures the SV size of the final averaging classifier is always bounded in expectation, a salient property that is crucial for online-to-batch conversion to obtain bounded kernel classifiers.

Unlike conventional approaches using budget maintenance to bound the SV size at each iteration, which fails to bound the averaging classifier, we propose a stochastic sampling strategy that sequentially constructs the set of support vectors by sampling from the sequence of incoming instances. The key challenge is how to design an appropriate sampling strategy so that the support vector size of the kernel classifiers is always bounded while the learning accuracy of the classifier could be maximized. To tackle this challenge, we propose a simple yet effective sampling rule, which decides if an incoming instance should be a support vector by performing a Bernoulli trial as follows:

$$\Pr(Z_t = 1) = \rho_t, \quad \rho_t = \frac{\min(\alpha, \ell_t(f_t))}{\beta},$$

where $Z_t \in \{0, 1\}$ is a random variable such that $Z_t = 1$ indicates a new support vector should be added to update the classifier at the tth step, and $\beta \geq \alpha > 0$ are parameters to adjust the ratio of support vectors with respect to some given budget. The above sampling rule has two key concerns:

(i) The probability of the tth step update is always bounded by α/β, which avoids assigning too large probability on noisy instances, meanwhile bounding the number of SVs of the final classifier by $\frac{\alpha}{\beta}T$ in expectation (detailed discussions are given in Theorem 2 in Section 5).

(ii) An example suffering a higher loss will be assigned to the support vector set with a higher probability. This aims to update the classifier by adding only very informative support vectors or equivalently avoid making unnecessary updates when the example is not difficult to be classified.

ALGORITHM 1: Sparse PA Learning with Kernels (**SPA**)

```
**Input:** aggressiveness parameter $\eta > 0$, and parameters $\beta \geq \alpha > 0$
**Initialize:** $f_1(\mathbf{x}) = 0$
**for** $t = 1, 2, \ldots, T$ **do**
    Receive example: $(\mathbf{x}_t, y_t)$
    Suffer loss: $\ell_t(f_t) = \ell(f_t; (\mathbf{x}_t, y_t))$
    Compute $\rho_t = \frac{\min(\alpha, \ell_t(f_t))}{\beta}$
    Sample a Bernoulli random variable $Z_t \in \{0, 1\}$ by: $\Pr(Z_t = 1) = \rho_t$
    Update the classifier (using Proposition 1):
    $f_{t+1} = \min_{f \in \mathcal{H}_\kappa} \frac{1}{2}\|f - f_t\|^2_{\mathcal{H}_\kappa} + \frac{Z_t}{\rho_t}\eta\ell_t(f)$
**end for**
**Output:** $\bar{f}_T(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$
```

Following the PA learning principle, we propose the following updating method:

$$f_{t+1} = \arg\min_{f \in \mathcal{H}_\kappa} P_t(f) := \frac{1}{2}\|f - f_t\|^2_{\mathcal{H}_\kappa} + \frac{Z_t}{\rho_t}\eta\ell_t(f). \tag{1}$$

Note when $\ell_t(f_t) = 0$, then $\rho_t = 0$ and $Z_t = 0$, we set $Z_t/\rho_t = 0$ so that no update is needed. We adopt the above update, because its objective is an unbiased estimate of that of PA update, that is,

$$\mathbb{E}(P_t(f)) = \frac{1}{2}\|f - f_t\|^2_{\mathcal{H}_\kappa} + \eta\ell_t(f).$$

We can derive a closed-form solution for the optimization Equation (1) as follows.

Proposition 1. *When $\ell_t(f) = [1 - y_t f(\mathbf{x}_t)]_+$, the optimization Equation (1) enjoys the following closed-form solution:*

$$f_{t+1}(\cdot) = f_t(\cdot) + \tau_t y_t \kappa(\mathbf{x}_t, \cdot), \quad \tau_t = \min\left(\frac{\eta Z_t}{\rho_t}, \frac{\ell_t(f_t)}{\kappa(\mathbf{x}_t, \mathbf{x}_t)}\right).$$

The above proposition can be easily verified. We omit the detailed derivations. Finally, we summarize the proposed SPA algorithm in Algorithm 1.

4 BOUNDED ONLINE MULTIPLE-KERNEL LEARNING

In this section, we first introduce the problem setting for OMKL [20, 22], and then we extend the proposed SPA algorithm to address the Bounded Online Multiple-Kernel Classification (BOMKC) problem. Note that our discussions will be focused on online classification tasks, but they can be easily extended to other online learning tasks, such as online regression [33], similarity learning [48], and beyond.

4.1 Problem Setting and Preliminaries

Consider a binary classification task, our goal is to learn a hypothesis $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathcal{X} \subset \mathbb{R}^d$ and its class label $y_t \in \mathcal{Y} = \{+1, -1\}$. We denote by $\hat{y} = \text{sign}(f(\mathbf{x}))$ the predicted class label, and $|f(\mathbf{x})|$ the classification confidence.

Consider a collection of m kernel functions $\mathcal{K} = \{\kappa_i : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, i = 1, \ldots, m\}$. Each kernel can be a predefined parametric or nonparametric function. MKL aims to learn a kernel-based prediction model by identifying the best linear combination of the m kernels whose weights are denoted by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$. The learning task can be cast into the following optimization:

$$\min_{\boldsymbol{\theta} \in \Delta} \min_{f \in \mathcal{H}_{K(\boldsymbol{\theta})}} \frac{1}{2} \|f\|^2_{\mathcal{H}_{K(\boldsymbol{\theta})}} + C \sum_{t=1}^{T} \ell(f(\mathbf{x}_t), y_t),$$

where $\Delta = \{\boldsymbol{\theta} \in \mathbb{R}^m_+ \,|\boldsymbol{\theta}^T \mathbf{1}_m = 1\}$, $K(\boldsymbol{\theta})(\cdot,\cdot) = \sum_{i=1}^m \theta_i \kappa_i(\cdot,\cdot)$, and $\ell(f(\mathbf{x}_t), y_t)$ is a convex loss function that penalizes the deviation of estimating $f(\mathbf{x}_t)$ from observed labels y_t. For simplicity, we denote $\ell_t(f) = \ell(f(\mathbf{x}_t), y_t)$.

The above convex optimization of regular batch MKL has been resolved using various optimization schemes [18]. Despite the extensive studies in literature, they suffer some common drawbacks of batch learning methods, i.e., poor scalability for large-scale applications, expensive retraining cost for increasing data, and cannot adapt to fast-changing patterns.

To address the challenges faced by batch MKL methods, OMKL techniques have been proposed to resolve the multiple kernel classification tasks in an online learning manner [20]. In particular, the Online Multiple-Kernel Classification (OMKC) method in Reference [20] consists of two major steps. First, it learns a set of single-kernel classifiers $f_t^i \in \mathcal{H}_{\kappa_i}, i = 1, \ldots, m$ using some existing online kernel learning algorithms (such as kernel Perceptron or kernel Passive-Aggressive). Second, it learns to find the optimal linear combination of these single-kernel classifiers to yield an effective final classifier $f_t(\mathbf{x})$:

$$f_t(\mathbf{x}) = \sum_{i=1}^{m} \theta_t^i f_t^i(\mathbf{x}), \tag{2}$$

where $\theta_t^i \in [0, 1]$ is the combination weight of the classifier with respect to κ_i at time t. The combination weights can be updated during the online learning process by adopting the Hedge algorithm [16].

Compared with batch learning methods, these OMKC algorithms are more scalable for large-scale applications and more natural for learning from sequential data. Despite these merits, one major deficiency of the existing OMKC algorithms is their unbounded support vector size. In particular, whenever a new instance is misclassified, it will be always added into the SV set. The unbounded support vector size will eventually lead to increasing cost for both computational time and memory space when dealing with very large-scale applications.

4.2 Bounded Online Multiple-Kernel Learning Using the SPA Algorithm

Similar to budget online kernel learning, the goal of bounded online multiple-kernel learning is to ensure the total number of support vectors in the final multi-kernel classifier is bounded by a given budget. To achieve this, the basic idea is to apply an existing bounded online kernel learning algorithm to bound the support vector size of each individual single-kernel classifier, which in turn can bound the total SV size given the fixed number of kernels. However, given the number of kernels can be potentially large, the challenge of bounded online multiple-kernel learning is not only to bound the number of SVs for each kernel classifier, but also to minimize the overall computational cost and maximize the learning accuracy.

In the following, we propose a novel BOMKC approach by extending the proposed SPA algorithm for multiple-kernel settings, which not only ensures the SV sizes are always bounded during the online learning process but also attempts to minimize the unnecessary updates for the poor quality kernels, which thus can improve both the overall efficiency and learning effectiveness.

Specifically, similar to the single-kernel SPA algorithm, we adopt a stochastic sampling strategy to decide if an incoming training instance should be added to the SV set. In particular, the sampling probability has two concerns:

ALGORITHM 2: Bounded OMKC Algorithm using SPA "**BOMKC(SPA)**"

```
INPUT:
Kernels: $k_i(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, i = 1, \ldots, m$; Aggressiveness parameter $\eta > 0$, and parameters $\beta \geq \alpha > 0$;
Discount parameter $\gamma \in (0,1)$ and smoothing parameters $\delta \in (0,1)$.
Initialization: $f_1^i = 0, \theta_1^i = \frac{1}{m}, i = 1, \ldots m$
for $t = 1, 2, \ldots, T$ do
    Receive an instance $\mathbf{x}_t$;
    Predict $\hat{y}_t = \text{sign}\Big(\sum_{i=1}^m \theta_t^i \cdot \text{sign}[f_t^i(\mathbf{x}_t)]\Big)$;
    Receive the true class label $y_t$;
    for $i = 1, 2, \ldots, m$ do
        Compute $p_t^i = (1-\delta)\frac{\theta_t^i}{\max_j \theta_t^j} + \delta$, $\rho_t^i = \frac{\min(\alpha, \ell_t(f_t^i))}{\beta}$;
        Sample a Bernoulli random variable $Z_t^i \in \{0,1\}$ by $\Pr(Z_t^i = 1) = \rho_t^i * p_t^i$
        if $Z_t^i = 1$ then
            Update the $i$th kernel classifier using Proposition 1
        end if
        Update weight $\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i}$, where $M_t^i = \mathbb{I}(y_t f_t^i(\mathbf{x}_t) < 0)$
    end for
    Scale the weights $\theta_{t+1}^i = \frac{\theta_{t+1}^i}{\sum_{j=1}^m \theta_{t+1}^j}, i = 1, \ldots, m$
end for
```

(i) The probability of updating the ith kernel classifier in the tth iteration should be related to the loss suffered by the current instance, as discussed in the single-kernel case (Section 3),

$$\rho_t^i = \frac{\min(\alpha, \ell_t(f_t^i))}{\beta}.$$

(ii) To make the algorithm efficient and avoid wasting time in learning with poor quality kernels, a kernel classifier with higher accumulated learning accuracy will be assigned with a higher sampling probability,

$$p_t^i = (1-\delta)\frac{\theta_t^i}{\max_j \theta_t^j} + \delta,$$

where θ_t^i is the importance weight variable for kernel combination, which reflects the historical classification performance of the ith kernel classifier, and $\delta \in (0,1)$ is a small constant for smoothing purpose (which ensures every kernel has a certain small probability to be sampled). This sampling strategy was originally proposed in the SD algorithm of OMKC using the stochastic updating strategy in Reference [20].

By combining the above strategies, we can now form the final updating strategy for BOMKC by performing a Bernoulli trial (for each kernel i at each step t) as follows:

$$\Pr(Z_t^i = 1) = \rho_t^i * p_t^i,$$

where $Z_t^i \in \{0,1\}$ is a random variable such that $Z_t^i = 1$ indicates if the incoming instance should be added as a new support vector to update the ith kernel classifier at the tth step. After a specific kernel classifier is selected, we then apply the proposed SPA algorithm for bounded online kernel learning with the individual kernel by Algorithm 1.

The remaining problem is how to learn the appropriate kernel combination weights θ_t^i for the set of single-kernel classifiers according to their classification accuracy at each online learning iteration. Specifically, we initialize their weights by a uniform distribution $\theta_1^i = \frac{1}{m}$. Then, the Hedge algorithm [16] is adopted to update the combination weights of the kernel classifiers, during the online learning process, i.e.,

$$\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i},$$

where $M_t^i = \mathbb{I}(y_t f_t^i(\mathbf{x}_t) < 0)$ indicates if there is a mistake, and $\gamma \in (0,1)$ is a discount weight parameter. Finally, we summarize the proposed BOMKC using the SPA algorithm in Algorithm 2.

5 THEORETICAL ANALYSIS

In this section, we give detailed theoretical analysis on the mistake bounds of the proposed SPA algorithm for both online single-kernel and multiple-kernel classification tasks. We denote by $f_* = \arg\min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^T \ell_t(f)$ the optimal classifier in $\mathcal{H}_\kappa$ space with the assumption of the foresight to all the instances.

Theorem 5.1. *Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of examples where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ for all t. If we assume $\kappa(\mathbf{x}, \mathbf{x}) = 1$ and $\ell_t(\cdot)$ is the hinge loss function, then for any $\beta \geq \alpha > 0$, $\alpha \leq 1$ and $\eta > 0$, when using the proposed SPA update strategy to generate a sequence of single-kernel classifiers in space $\mathcal{H}_\kappa$, we have the bound for the expected number of mistakes,*

$$\mathbb{E}\left[\sum_{t=1}^T M_t\right] \leq \max\left(\frac{1}{\eta}, \frac{\beta}{\alpha}\right)\left[2\eta \sum_{t=1}^T \ell_t(f_*) + ||f_*||^2_{\mathcal{H}_\kappa}\right],$$

where $M_t = \mathbb{I}(y_t \neq \hat{y}_t)$.

Proof. We first generalize the Lemma 1 in Reference [10] to space $\mathcal{H}_\kappa$ and get the following inequality:

$$\sum_{t=1}^T \tau_t \left(2\ell_t(f_t) - \tau_t \kappa(\mathbf{x}_t, \mathbf{x}_t) - 2\ell_t(f_*)\right) \leq ||f_*||^2_{\mathcal{H}_\kappa},$$

where $\tau_t = \min(\frac{\eta Z_t}{\rho_t}, \frac{\ell_t(f_t)}{\kappa(\mathbf{x}_t, \mathbf{x}_t)})$. Plugging $\tau_t \kappa(\mathbf{x}_t, \mathbf{x}_t) \leq \ell_t(f_t)$ into the above inequality gives

$$\sum_{t=1}^T \tau_t \left(\ell_t(f_t) - 2\ell_t(f_*)\right) \leq ||f_*||^2_{\mathcal{H}_\kappa}.$$

Re-arranging the above inequality gives

$$\sum_{t=1}^T \tau_t \ell_t(f_t) \leq \sum_{t=1}^T 2\frac{\eta Z_t}{\rho_t} \ell_t(f_*) + ||f_*||^2_{\mathcal{H}_\kappa}.$$

Combining the above inequality with the fact $\ell_t(f_t) > 1$ when $y_t \neq \hat{y}_t$ gives

$$\sum_{t=1}^T \min\left(\frac{\eta Z_t}{\rho_t}, 1\right) M_t \leq \sum_{t=1}^T \tau_t M_t \leq \sum_{t=1}^T \tau_t \ell_t(f_t) \leq \sum_{t=1}^T 2\frac{\eta Z_t}{\rho_t} \ell_t(f_*) + ||f_*||^2_{\mathcal{H}_\kappa}, \tag{3}$$

where $M_t = \mathbb{I}(y_t \neq \hat{y}_t)$. Now, we can take conditional expectation with respect to Z_t (given all random variables $Z_1, \ldots, Z_{t-1}$) to get

$$\mathbb{E}\left[\min\left(\frac{\eta Z_t}{\rho_t}, 1\right) M_t | Z_1, \ldots Z_{t-1}\right] = \min(\eta, \rho_t) M_t,$$

$$\mathbb{E}\left[\frac{Z_t}{\rho_t} | Z_1, \ldots, Z_{t-1}\right] = 1.$$

Note that when $M_t = 1$, i.e., there is a mistake at the tth iteration, we have $\ell_t(f_t) > 1$. Due to the assumption that $\alpha \leq 1$, we have $\rho_t = \frac{\alpha}{\beta}$,

$$\mathbb{E}\left[\min\left(\frac{\eta Z_t}{\rho_t}, 1\right) M_t | Z_1, \ldots Z_{t-1}\right] = \min\left(\eta, \frac{\alpha}{\beta}\right) M_t.$$

Plugging the above two equalities into the inequality Equation (3) gives

$$\min\left(\eta, \frac{\alpha}{\beta}\right) \mathbb{E}\left[\sum_{t=1}^{T} M_t\right] \leq 2\eta \sum_{t=1}^{T} \ell_t(f_*) + ||f_*||^2_{\mathcal{H}_\kappa}.$$

Re-arranging the above inequality concludes this proof. □

Remark. This lemma indicates that the accuracy of the proposed SPA algorithm decreases when $\frac{\beta}{\alpha}$ increases, i.e., when less support vectors are sampled. In addition, it is easy to justify that the minimum of the mistake bound is achieved when setting $\frac{1}{\eta} = \frac{\beta}{\alpha}$, which is identical to the mistake bound proposed for PA-I (Theorem 4, [10]). Namely, although only a small fraction of the SVs are sampled, the proposed algorithm achieves a strong mistake bound in expectation.

Next, we will bound the number of SVs of the single-kernel classifier f_T in expectation.

Theorem 5.2. *Let* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *be a sequence of examples where* $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ *for all* t. *If we assume* $\kappa(\mathbf{x}, \mathbf{x}) = 1$ *and the hinge loss function* $\ell_t(\cdot)$ *is* 1*-Lipschitz,* $\beta \geq \alpha > 0$, $\alpha \leq 1$, *and* $\eta > 0$, *for any* $i = 1, \ldots m$, *then the expected number of support vectors yielded by the proposed SPA algorithm satisfies*

$$\mathbb{E}\left[\sum_{t=1}^{T} Z_t\right] \leq \frac{\alpha}{\beta} T.$$

Proof. Since $\mathbb{E}_t[Z_t] = \rho_t$, where $\mathbb{E}_t$ is the conditional expectation, we have

$$\mathbb{E}\left[\sum_{t=1}^{T} Z_t\right] = \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}_t Z_t\right] = \mathbb{E}\left[\sum_{t=1}^{T} \rho_t\right] = \mathbb{E}\left[\sum_{t=1}^{T} \min\left(\frac{\alpha}{\beta}, \frac{\ell_t(f_t)}{\beta}\right)\right] \leq \frac{\alpha}{\beta} T. \quad □$$

Remark. First, this theorem indicates the expected number of support vectors is less than $\frac{\alpha T}{\beta}$. Thus, by setting $\beta \geq \frac{\alpha T}{B}$ $(1 < B \leq T)$, we guarantee the expected number of support vectors of the final classifier is bounded by a budget B. In practice, as the online multiple-kernel classifier using Hedge algorithm trends to converge to the single best kernel classifier, the total number of support vectors of all classifiers is only slightly larger than that used by the best classifier.

We have demonstrated that the number of mistakes made by a single-kernel online classifier is bounded by some constant factor of its batch counterpart in Theorem 5.1. While in the multiple-kernel classification task, the performance of single-kernel classifiers varies significantly, and we have no foresight to the winner. In the following, we will show that the final multiple-kernel classifier in our proposed algorithm achieves almost the same performance as that of the best online single-kernel classifier. We denote $f_*^i = \arg\min_{f \in \mathcal{H}_\kappa^i} \sum_{t=1}^{T} \ell_t(f)$ as the optimal classifier in $\mathcal{H}_\kappa^i$ space with the assumption of the foresight to all the instances.

Theorem 5.3. *Let* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *be a sequence of examples where* $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ *for all* t. *If we assume* $\kappa_i(\mathbf{x}, \mathbf{x}) = 1$ *and* $\ell_t(\cdot)$ *is the hinge loss function, then for any* $\beta \geq \alpha > 0$, $\alpha \leq 1$, *and* $\eta > 0$, $\gamma \in (0, 1)$, $\delta \in (0, 1)$, *the expected number of mistakes made by the multiple-kernel* *classifier generated by our proposed SPA algorithm satisfies the following inequality:*

$$\mathbb{E}\left[\sum_{t=1}^{T} M_t\right] \leq \min_{i\in[m]} \frac{2\ln m - 2\ln\gamma \max(\frac{1}{\eta\delta}, \frac{\beta}{\alpha\delta})\left[2\eta\sum_{t=1}^{T}\ell_t(f_*^i) + ||f_*^i||^2_{\mathcal{H}_\kappa^i}\right]}{1-\gamma},$$

where $M_t = \mathbb{I}(y_t \neq \hat{y}_t)$.

Proof. In the following proof, we first generalize the loss bound of the Hedge algorithm [16] to a different situation, where stochastic update and stochastic combination are adopted. Using the analysis in Theorem 2 [16] and the initialization w_1^i with $1/m$, we have

$$\sum_{t=1}^{T}\sum_{i=1}^{m}\theta_t^i M_t^i \leq \frac{\ln m - \ln\gamma\sum_{t=1}^{T} M_t^i}{1-\gamma}, \tag{4}$$

where $M_t^i = 1$ indicates that classifier f^i makes a mistake in time t. Since

$$\sum_{t=1}^{T} M_t = \sum_{t=1}^{T} I\left(\sum_{i=1}^{m}\theta_t^i M_t^i > 0.5\right) \leq 2\sum_{t=1}^{T}\sum_{i=1}^{m}\theta_t^i M_t^i,$$

we have

$$\sum_{t=1}^{T} M_t \leq \frac{2\ln m - 2\ln\gamma\sum_{t=1}^{T} M_t^i}{1-\gamma}. \tag{5}$$

Now, we need to bound the number of mistakes of a single classifier, $\sum_{t=1}^{T} M_t^i$.

In Theorem 1, we give the mistake bound of one single-kernel classifier learnt by SPA assuming $\mathbb{E}[Z_t] = \rho_t$ in Equation (3). Similarly, we have the following inequality for each single classifier in multi-kernel SPA algorithm:

$$\sum_{t=1}^{T}\min\left(\frac{\eta Z_t^i}{\rho_t^i}, 1\right) M_t^i \leq \sum_{t=1}^{T} 2\frac{\eta Z_t^i}{\rho_t^i}\ell_t(f_*^i) + ||f_*^i||^2_{\mathcal{H}_\kappa^i}. \tag{6}$$

Here, according to Algorithm 2, $\mathbb{E}[Z_t^i] = \rho_t^i * p_t^i$, where $\delta < p_t^i < 1$. Now, we can take conditional expectation with respect to Z_t^i (given all random variables $Z_1^i, \ldots, Z_{t-1}^i$) to get

$$\mathbb{E}\left[\min\left(\frac{\eta Z_t^i}{\rho_t^i}, 1\right) M_t^i | Z_1^i, \ldots Z_{t-1}^i\right] \geq \delta\min(\eta, \rho_t^i) M_t^i = \delta\min\left(\eta, \frac{\alpha}{\beta}\right) M_t^i,$$

$$\mathbb{E}\left[\frac{Z_t^i}{\rho_t^i} | Z_1^i, \ldots, Z_{t-1}^i\right] \leq 1,$$

where the fact $\alpha \leq 1$ is used for the first equality. Plugging the above two equalities into the inequality Equation (6) gives

$$\mathbb{E}\left[\sum_{t=1}^{T} M_t^i\right] \leq \max\left(\frac{1}{\eta\delta}, \frac{\beta}{\alpha\delta}\right)\left[2\eta\sum_{t=1}^{T}\ell_t(f_*^i) + ||f_*^i||^2_{\mathcal{H}_\kappa^i}\right].$$

Combining the above inequality with inequality Equation (5) concludes this proof. □

Remark. The key difference between the single-kernel bound and multiple-kernel bound is the factor $-2\frac{\ln\gamma}{1-\gamma}$, which is a decreasing function of γ and $\lim_{\gamma\mapsto 1} -2\frac{\ln\gamma}{1-\gamma} = 2$. For example, we set $\gamma = 0.99$ in our experiments and $-2\frac{\ln\gamma}{1-\gamma} = 2.01$. This theorem implies that even without any foresight of which kernel would achieve the best accuracy, the mistake rate of our proposed bounded online

Table 1. Summary of Binary Classification Datasets in Our Experiments

Dataset	# instances	#features
KDD08	102,294	117
A9a	48,842	123
W7a	49,749	300
Codrna	59,535	8
Covtype	581,012	54
SUSY	1,000,000	18

multiple-kernel classifier is near to the best single-kernel classifier. This algorithm is powerful when there is a large set of kernels to choose from and their performance varies significantly.

6 EXPERIMENTS

In this section, we conduct an extensive set of experiments to evaluate the empirical performance of the proposed SPA algorithms for online binary classification tasks in both single-kernel and multiple-kernel settings.

6.1 Experiments for Single-Kernel Classification

We first evaluate the empirical performance of the proposed SPA algorithm on single-kernel classification tasks.

6.1.1 Experimental Testbed. Table 1 summarizes details of some binary classification datasets in our experiments. The first five can be downloaded from LIBSVM[1] or KDDCUP competition site.[2] The original SUSY dataset contains 5,000,000 instances, we randomly sampled a subset.

6.1.2 Compared Algorithms and Experimental Setup. We evaluate the proposed SPA algorithm by comparing with many state-of-the-art online kernel learning algorithms. First, we implement the following non-budget kernel learning algorithms as a yardstick for evaluation:

- "Perceptron": the kernelized Perceptron [17];
- "OGD": the kernelized Online Gradient Descent algorithm [23];
- "PA-I": the kernelized Passive-Aggressive algorithm with soft margin [10].

Further, we compare a variety of budget online kernel learning algorithms:

- "RBP": the Randomized Budget Perceptron by random removal [4];
- "Forgetron": the Forgetron by discarding oldest support vectors [14];
- "Projectron": the Projectron algorithm using the projection strategy [30];
- "Projectron++": the aggressive version of Projectron algorithm [30];
- "BPA-S": the Budget Passive-Aggressive algorithm [47], for which we only adopt the BPA-Simple algorithm, since the other two variants are too computationally expensive for large datasets;
- "BOGD": the Bounded Online Gradient Descent algorithm [45, 53];
- "OSKL": the Online Sparse Kernel Learning algorithm [51].

[1]http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/.
[2]http://www.sigkdd.org/kddcup/.

As the SPA and OSKL algorithm are designed for online-to-batch conversion problem, we report the online prediction results of the averaged classifier $\frac{1}{t}\sum_{\tau=1}^{t} f_\tau$. While for the other budget algorithms, we report the online prediction results of f_t. There are two reasons for not reporting their performance of averaged classifier $\frac{1}{t}\sum_{\tau=1}^{t} f_\tau$. First, these budget algorithms were designed for pure online learning, not online-to-batch conversion. Second, for some of the budget algorithms that discard SVs along the learning process (RBP, Forgetron, BPA-S, and BOGD), the averaged classifier $\frac{1}{t}\sum_{\tau=1}^{t} f_\tau$ is unbounded; i.e., they contain almost the same SVs as that of non-budget Perceptron.

To make a fair comparison, we adopt the same experimental setup for all the algorithms. We use the hinge loss for gradient-based algorithms (OGD and BOGD). For all the algorithms, we adopt a Gaussian kernel $\exp(-\gamma||x_1 - x_2||^2)$ with parameter γ set to 0.4 for all the datasets. The learning rate parameter η for OGD, BOGD, OSKL, and SPA is automatically chosen by searching from $\{10^3, \ldots, 10^{-3}\}$ based on a random permutation of each dataset. We adopt the PA-I algorithm for comparison, since it was proved to be robust to noise and achieved better performance than original PA without soft margin. The soft margin parameter C is optimized by searching from range {0.25, 0.5, 1, 2}. The support vector size of the proposed SPA algorithm depends on the choices of parameters α and β. In our experiments, the parameter $\alpha = 1$ is chosen for all other datasets. Then, we choose a proper β parameter so that the resulting support vector size is roughly a proper fraction of that of the non-budget OGD algorithm (specifically, we set $\beta = 20$ for three small datasets, and $\beta = 200$ for three large datasets). Note that the OSKL algorithm also adopts a stochastic approach to sample the support vectors, which may generate varied number of support vectors when choosing different parameters. This raises some practical challenge in choosing the budget on the size of support vectors when comparing with our algorithm. To achieve the most fair comparison, we tune the sampling probability parameter G in the OSKL algorithm so that the average number of resulting support vectors is closest to that of our proposed algorithm. Finally, to ensure that all budget algorithms adopt the same budget size, we choose the budget size B of all the other compared algorithms according to the average number of support vectors generated by the proposed SPA algorithm.

For each dataset, all the experiments were repeated for 20 times on different random permutations of instances in the dataset and all the results were obtained by averaging over these 20 runs. For performance metrics, we evaluate the online classification performance by mistake rates and running time. Finally, all the algorithms were implemented in C++, and all experiments were conducted in a PC with 3.2GHz CPU.

6.1.3 Evaluation of Online Single-Kernel Learning Performance. The first experiment is to evaluate the performance of kernel-based online learning algorithms for online classification tasks. Table 2 shows the experimental results. To examine the superiority of averaging classifier in online setting, the accuracies of both our SPA and OSKL algorithms were obtained using the averaged classifier $\frac{1}{t}\sum_{i=1}^{t} f_i$. Since most of the existing algorithms don't have a bounded averaged classifier, we use f_t for prediction. The aim of this experiment is to show the advantages of our averaged classifier in accuracy and time cost in comparison to existing algorithms under the same SV setting. We can draw some observations below.

First, by examining the time cost comparisons, all the budget algorithms are significantly more efficient than the non-budget algorithms (Perceptron, OGD, and PA-I) for most cases, especially on large-scale datasets. This validates the importance of studying bounded online kernel learning algorithms. Second, by comparing different budget algorithms, Projectron++ is the least efficient due to its expensive projection strategy, and the proposed SPA algorithm is the most efficient. The other budget algorithms (RBP, Forgetron, BPAS, BOGD) are in general quite efficient as they

Table 2. Evaluation of Online Kernel Classification on Six Datasets

	KDD08, $\beta = 20$			a9a, $\beta = 20$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	99.04±0.01	11.58	0.9k	79.40± 0.11	19.93	9.9k
OGD	99.44±0.01	14.54	1.2k	83.41 ± 0.05	54.20	23.5k
PA-I	99.45±0.01	39.53	3.2k	84.07±0.08	57.91	22.8k
RBP	98.96±0.03	3.24	149	78.83±0.38	5.34	1,350
Forgetron	98.93±0.03	3.28	149	78.08±0.22	6.45	1,350
Projectron	99.02±0.01	4.19	149	79.25±0.13	32.47	1,350
Projcetron++	99.32±0.01	4.86	149	79.35±0.13	150.63	1,350
BPAS	**99.41±0.01**	3.93	149	80.44±0.14	7.75	1,350
BOGD	99.39±0.01	3.40	149	80.97±0.04	6.17	1,350
OSKL	**99.41±0.01**	**2.63**	149	82.01±0.32	4.08	1,344
SPA	**99.41±0.01**	**2.60**	149	**82.04± 0.25**	**3.84**	1,350

	w7a, $\beta = 200$			codrna, $\beta = 20$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	97.64 ±0.04	2.50	1.1k	90.79 ±0.09	6.21	5.4k
OGD	98.06±0.02	38.16	10.1k	93.31±0.05	10.41	8.8k
PA-I	98.16±0.02	63.47	19.6k	93.65±0.05	15.82	12.4k
RBP	96.36±0.06	0.72	175	86.59±0.22	1.68	822
Forgetron	95.19±0.39	0.87	175	86.62±0.15	1.91	822
Projectron	95.19±0.39	0.87	175	83.35±0.23	19.38	822
Projectron++	95.90±0.38	3.01	175	84.71±0.19	32.99	822
BPAS	96.83±0.23	1.77	175	91.23±0.05	2.21	822
BOGD	96.75±0.03	1.05	175	85.62±0.06	1.92	822
OSKL	96.98±0.43	0.71	177	**92.07±0.37**	1.37	825
SPA	**97.05±0.13**	**0.56**	175	91.59±0.35	**1.31**	822

	covtype, $\beta = 200$			SUSY, $\beta = 200$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	73.08±0.03	2861	156.0k	71.69±0.04	7298	283.2k
OGD	78.34±0.03	5113	296.7k	78.97±0.01	13648	491.7k
PA-I	78.18±0.05	4402	298.2k	78.96± 0.01	14715	507.2k
RBP	65.63±0.23	50.27	1,510	66.44±0.26	96.7	1,933
Forgetron	65.33±0.22	67.60	1,510	66.31±0.33	131.5	1,933
Projectron	57.82±6.09	316.05	1,510	54.46±0.10	4523.0	1,933
Projcetron++	59.33±5.85	470.35	1,510	60.01±1.62	3738.6	1,933
BPAS	**72.37±0.13**	73.48	1,510	74.85±0.07	157.4	1,933
BOGD	68.75±0.03	54.44	1,510	68.75±0.03	105.0	1,933
OSKL	72.11±0.67	29.77	1,508	73.57± 0.47	74.3	1,957
SPA	71.34±0.59	**27.53**	1,510	**80.04± 0.34**	**49.2**	1,933

Time in seconds.

all are based on simple SV removal strategy for budget maintenance. The reason that our SPA algorithm is even more efficient than these algorithms is because our algorithm adds the support vectors incrementally, while the other algorithms perform the budget maintenance only when the support vector size reaches the budget. This encouraging result validates the high-efficiency advantage of our stochastic support vector sampling strategy.

Next, by comparing online classification accuracy results of different algorithms, we found that the non-budget algorithms generally achieve better accuracy results than their budget variants. This is not surprising as the non-budget algorithms use a much larger SV size. Furthermore, by comparing different budget algorithms, we found that PA- and OGD-based algorithms (BPAS, BOGD, and SPA) generally outperform Perceptron-based algorithms (RBP, Forgetron, Projectron) due to more aggressive and precise updating strategies. Finally, the proposed SPA algorithm achieves the best accuracy among all the budget algorithms for most cases, and is at least comparable to, if not better than, the OSKL algorithm. These results again validate the effectiveness of the proposed bounded learning strategy.

6.1.4 Evaluation of Output Single-Kernel Classifiers on Test Data. A key advantage of SPA is that it assures the final output averaged classifier is sparse, which is very important when applying the output classifier in real applications. Our last experiment thus is to examine if the final output classifier of SPA is effective for batch classification. We evaluate two SPA classifiers: "SPA-last" that simply outputs the classifier at the last iteration f_t as the final classifier, and "SPA-avg" that outputs the average of classifiers at every iteration, i.e., $\frac{1}{T}\sum_{t=1}^{T} f_t$.

We compare our algorithms with two state-of-the-art batch algorithms for SVM classifications: (1) LIBSVM, a widely used and the most accurate solution of kernel SVM for batch classification [6]; (2) Pegasos[3] [37], an efficient stochastic gradient descent solver for SVM, for which we adapt it for kernel learning. Moreover, we compare our solutions with the output classifiers by two bounded kernel-based online algorithms: BOGD and BPAS, as they achieve the highest accuracy among all the existing algorithms in previous experiments of Table 2. For these two algorithms, as their average classifier is not sparse, we use their last classifiers for comparisons.

To enable fair comparisons, all the algorithms follow the same setup for batch classification. We conduct the experiments on 4 medium-scale datasets as used in previous online experiments: "a9a," "codrna," "w7a," and "KDDCUP08" (the other two large datasets were excluded due to too long training time by LIBSVM). We use the original splits of training and test sets on the LIBSVM website. We adopt the same Gaussian kernel with the same kernel parameter γ for all the algorithms. We perform cross validation on the training set to search for the best parameters of different algorithms. In particular, we search for the best kernel parameter γ in the range of $\{2^5, 2^4, \ldots, 2^{-5}\}$, the parameter C of SVM in the range of $\{2^5, \ldots 2^{-5}\}$, both the regularization parameter λ in Pegasos and BOGD, and the learning rate parameter η in BOGD and SPA in the range of $\{10^3, 10^2, \ldots, 10^{-3}\}$. For the proposed SPA-last and SPA-avg, we set $\alpha = 1$ and $\beta = 5$ for all the datasets.

Table 3 shows the results, where we only report the test set accuracy and training time (we exclude the test time as it is proportional to SV sizes). We can draw some observations. First, in terms of training time, the budget algorithms run much faster than LIBSVM and Pegasos, in which our SPA algorithm achieves the lowest training time among all. Specifically, compared with batch algorithms, SPA achieves the speedup of training time for about 20 to 30 times over LIBSVM, and about 5 times over Pegasos.

Second, by examining the test set accuracy, we found that LIBSVM always achieves the best. The proposed SPA-avg algorithm achieves the best accuracy among all the budget algorithms, which is slightly lower but fairly comparable to LIBSVM, and even beats the accuracy of Pegasos that has almost 4 times more SVs than our SPA algorithm. This promising result validates the efficacy of our SPA algorithm for producing sparse and effective average classifier.

Third, we notice that BOGD, BPAS, and SPA-last achieve similar test set accuracy, but their standard deviations are in general much larger than that of SPA-avg for most cases. This shows

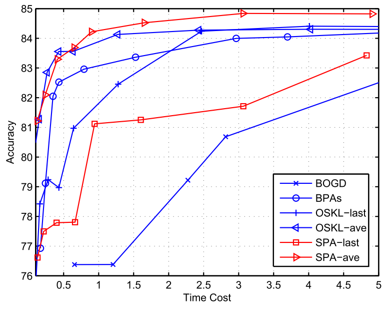
[3] http://www.cs.huji.ac.il/shais/code/.

Table 3. Evaluation of Final Classifiers for Test Data

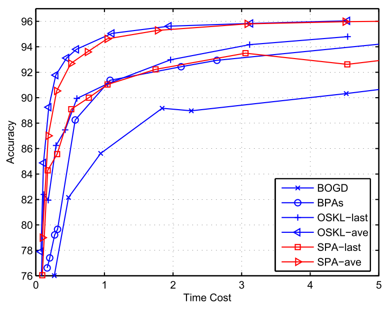
	KDD08			w7a		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
LIBSVM	99.39	577.3	14.6k	98.31	92.61	8.0k
Pegasos	99.50±0.01	976.8	81.8k	97.56±0.01	29.21	24.7k
BOGD	99.37±0.01	31.84	1,760	97.05±0.01	8.33	3,710
BPAS	99.22±0.29	37.48	1,760	97.75±0.40	13.95	3,710
OSKL-last	99.20±0.01	22.79	1,752	97.92±0.20	5.08	3,726
OSKL-avg	99.20±0.01	22.79	1,752	97.94±0.07	5.08	3,726
SPA-last	99.41±0.07	**19.42**	1,760	97.49±2.05	**5.04**	3,710
SPA-avg	**99.46±0.01**	**19.42**	1,760	**97.97±0.04**	**5.04**	3,710

	a9a			codrna		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
LIBSVM	85.04	97.8	11.5k	96.67	80.2	8.0k
Pegasos	84.66±0.21	15.8	11.0k	95.63±0.43	14.9	12.2k
BOGD	82.49±1.22	5.85	2,079	92.10±1.32	5.34	2,386
BPAS	82.11±0.83	7.59	2,079	93.12±2.01	6.98	2,386
OSKL-last	80.17±3.91	3.19	2,073	94.15±1.66	3.42	2,410
OSKL-avg	84.28±0.21	3.19	2,073	95.67±0.13	3.42	2,410
SPA-last	82.97±2.59	**3.16**	2,079	91.23±3.40	**3.03**	2,386
SPA-avg	**84.88±0.10**	**3.16**	2,079	**96.05±0.09**	**3.03**	2,386

Time in seconds.



(a) a9a



(b) codrna

Fig. 1. Evaluation of different budget single-kernel algorithms on a9a and codrna datasets. The curves of online mistake rates vs. time costs were obtained by choosing varied budget values.

that employing the averaged classifier with SPA for test data classification leads to more accurate and more stable performance than many budget learning algorithms that output the last classifier, which again validates the advantage of our technique.

6.1.5 Experiments on Various Budget Sizes. In the previous experiments, we compared many state-of-the-art online budget learning algorithms under fixed budget size. To be more convincing, in this subsection, we show the "time versus accuracy" of all compared algorithms under varied SV size setting in Figure 1.

The train and test splitting and parameter setting are identical to that used in Table 3, except the β and B. We adopt various β and B values to show the different performance of all algorithms

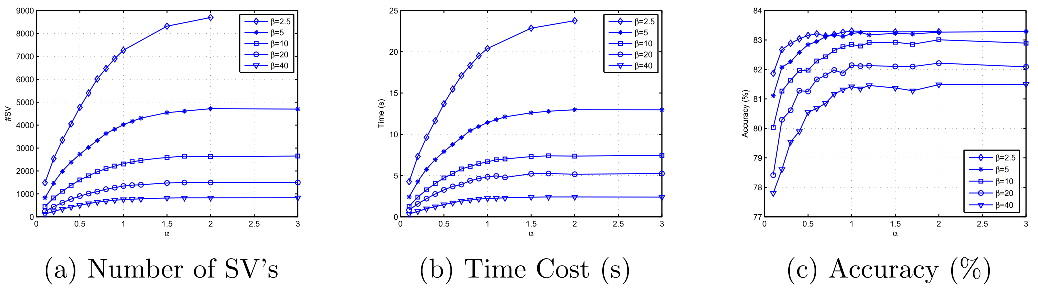


Fig. 2. Impacts of parameters α and β for #SV, time cost and accuracy by SPA on "a9a."

under various SV's set size. Finally, we plot the accuracy on the same time axis for fair comparison. Several observation can be drawn as follows:

First, for all algorithms, more support vectors (larger B, smaller β) result in higher accuracy. Consequently, it is important for the learner to choose a proper trade-off between efficiency and accuracy. Second, when comparing the "averaged classifiers" (SPA-ave and OSKL-ave) and their corresponding "last classifiers," we find that the averaged classifiers always achieve higher accuracy, which validates our main motivation. Third, by comparing the accuracy between different algorithms, we find our proposed algorithm, SPA-ave, gets better accuracy than most of the compared algorithms and sometimes comparable to that of OSKL. These observations consist with that in Table 3.

6.1.6 Parameter Sensitivity of α and β. The proposed SPA algorithm has two critical parameters, α and β, which could considerably affect the accuracy, support vector size, and time cost. Our second experiment is to examine how different values of α and β affect the learning performance to give insights for how to choose them in practice. Figure 2 shows the performance (support vector size, time, accuracy) of the SPA algorithm on the "a9a" dataset with varied α values ranging from 0.1 to 3, and varied β in the range of $\{2.5, 5, 10, 20, 40\}$. Several observations can be drawn from the experimental results.

First, when β is fixed, increasing α generally results in (i) larger support vector size, (ii) higher time cost, but (iii) better classification accuracy, especially when α is small. However, when α is large enough (e.g., $\alpha > 1.5$), increasing α has very minor impact to the performance. This is because the number of instances whose hinge loss above α is relatively small. We also note that the accuracy decreases slightly when α is too large, e.g., $\alpha >= 3$. This might be because some (potentially noisy) instances with large loss are given a high chance of being assigned as SVs, which may harm the classifier due to noise. Thus, on this dataset ("a9a"), it is easy to find a good α in the range of $[1, 2]$.

Second, when α is fixed, increasing β will result in (i) smaller support vector size, (ii) smaller time cost, but (iii) worse classification accuracy. On one hand, β cannot be too small as it will lead to too many support vectors and thus suffer very high time cost. On the other hand, β cannot be too large as it will considerably decrease the classification accuracy. We shall choose β that yields a sufficiently accurate classifier while minimizing the support vector size and training time cost. For example, for this particular dataset, choosing β in the range of $[5, 10]$ achieves a good trade-off between accuracy and efficiency/sparsity.

Third, we would like to discuss about the impact of the number of SVs on the accuracy. As the proposed algorithm is an approximation of the kernel PA algorithm by sampling a subset of SVs, increasing the number of SVs usually leads to higher accuracy. However, when the number of SVs is already large, increasing it does not have much influence on the accuracy. In other words, the

Table 4. Summary of Binary Classification Datasets in the OMKC Experiments

Datasets	# instances	#features
German	1,000	24
Svmguide3	1,243	21
Madelon	2,000	500
Magic04	19,020	10
A9a	48,842	123
Ijcnn1	49,990	22
Kdd08	102,294	117
Codrna	271,617	8
Susy	1,000,000	18

marginal utility of increasing SVs decreases. This can be easily observed from Figure 1. When the number of SVs increases from 5,000 to 9,000, the accuracy nearly stays unchanged.

6.2 Experiments for Online Multiple-Kernel Classification

We now evaluate the empirical performance of the proposed SPA technique for bounded online multiple-kernel classification tasks.

6.2.1 Experimental Testbed. All datasets used in our experiments are commonly used benchmark datasets and are publicly available from LIBSVM, UCI,[4] and KDDCUP competition site. These datasets are chosen fairly randomly to cover a variety of different dataset scales. We summarize the details of the datasets in Table 4.

6.2.2 Kernels. In our experiments, we examine BOMKC by exploring a set of 16 predefined kernels, including 3 polynomial kernels $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j)^p$ with the degree parameter $p = 1, 2, 3$; 13 Gaussian kernels $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2_{\mathcal{H}_\kappa}}{2\sigma^2})$ with the kernel width parameter $\sigma = [2^{-6}, 2^{-5}, \ldots, 2^6]$.

6.2.3 Compared Algorithms. First, we include an "ideal" baseline algorithm, which assumes the best kernel among the pool of kernels can be disclosed prior to the arrival of training data at the beginning of online learning. Specifically, we search for the best single-kernel classifier from the set of 16 predefined kernels using one random permutation of all the training examples. Using this best kernel, we then construct an "ideal" kernel classifier using the Perceptron algorithm [32], denoted as "Per(*)" in later discussion.

The second group of compared algorithms are the existing OMKC algorithms in Reference [20], including three variants of OMKC:

- "U": the OMKC algorithm with a naive **U**niform combination;
- "DD": OMKC with **D**eterministic combination and **D**eterministic update;
- "SD": OMKC with **S**tochastic update and **D**eterministic combination.

Since all single-kernel component classifiers in the above OMKC algorithms are updated by Perceptron, we amend the two existing OMKC algorithms by adopting the Passive-Aggressive [10] update strategy to obtain two stronger baselines for comparison, including:

[4]http://archive.ics.uci.edu/ml/.

ALGORITHM 3: The Bounded OMKC-DD Using the Deterministic Update

```
for $t = 1, 2, \ldots, T$ do
    Receive an instance $\mathbf{x}_t$;
    Predict $\hat{y}_t = \text{sign}\left( \sum_{i=1}^{m} \theta_t^i \cdot \text{sign}[f_t^i(\mathbf{x}_t)] \right)$
    Receive the true class label $y_t$
    for $i = 1, 2, \ldots, m$ do
        Update weight $\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i}$, where $M_t^i = \mathbb{I}(y_t f_t^i(\mathbf{x}_t) < 0)$
        if $\#SV_i < \frac{B}{m}$ then
            Normal update;
        else
            Budget maintenance;
        end if
        Scale the weights $\theta_{t+1}^i = \frac{\theta_{t+1}^i}{\sum_{j=1}^{m} \theta_{t+1}^j}, i = 1, \ldots, m$
    end for
end for
```

- "DDPA": we replace Perceptron with the PA algorithm in OMKC-DD;
- "SDPA": we replace Perceptron with the PA algorithm in OMKC-SD.

Finally, to test the efficiency and effectiveness of our budget strategy, we should also compare with budget OMKC algorithms. Since very few existing work has attempted to address this issue, we then construct a few baselines by turning the above OMKC algorithms ("DD" and "SD") into budget OMKC by replacing its unbounded single-kernel classifiers with some existing budget online (single) kernel learning algorithms. These result in the following BOMKC algorithms for comparisons:

- "RBP": the Random Budget Perceptron algorithm [4];
- "Forgetron": budget Perceptron by discarding the oldest SV [14];
- "BOGD": the Budget OGD algorithm [53];
- "BPAS": the Budget PA algorithm [47].

Due to the highly intensive computational costs, we exclude the comparisons with other insufficient budget online kernel learning algorithms such as Projectron and its variants. Although the procedure of both DD and SD algorithms was clearly presented in Reference [20], we still describe its budget variants used in our comparison in Algorithms 3 and 4 for easier understanding, where B is the total budget size of all component classifiers and the "budget maintenance" step denotes any of the four budget algorithms.

6.2.4 Parameter Settings. To make a fair comparison, we adopt the same experimental setup for all the algorithms. The weight discount parameter γ is fixed to 0.99 for all multiple-kernel algorithms on all datasets. The smoothing parameter δ for all stochastic update algorithms is fixed to 0.001. The learning rate parameters in PA-based algorithms (SPA, BPAS, DDPA, SDPA) are all fixed to 0.1. The regularization parameter λ and learning rate parameter η in BOGD is searched in the range of $\{1, 0.1, \ldots, 0.0001\}$. For the proposed SPA algorithm, we set $\alpha = 1$ for all the datasets. In addition, as discussed in the theoretical analysis, the number of SVs is bounded by $\alpha T/\beta$, which indicates that β should vary according to the number of instances T in a dataset. We thus set $\beta = 3$ for smaller datasets ($T < 10^5$) and $\beta = 10$ for other datasets. For fair comparison, we choose the

ALGORITHM 4: The Bounded OMKC-SD Using the Stochastic update

```
for $t = 1, 2, \ldots, T$ do
    Receive an instance $\mathbf{x}_t$,
    Predict $\hat{y}_t = \text{sign}\Big(\sum_{i=1}^m \theta_t^i \cdot \text{sign}[f_t^i(\mathbf{x}_t)]\Big)$
    Receive the true class label $y_t$
    for $i = 1, 2, \ldots, m$ do
        Compute $p_t^i = (1-\delta)\frac{\theta_t^i}{\max_j \theta_t^j} + \delta$;
        Sample a Bernoulli random variable $Z_t^i \in \{0, 1\}$ by $\Pr(Z_t^i = 1) = p_t^i$
        if $Z_t^i = 1$ then
            Update weight $\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i}$, where $M_t^i = \mathbb{I}(y_t f_t^i(\mathbf{x}_t) < 0)$
            if $\sum_{i=1}^m \#SV_i < B$ then
                Normal update;
            else
                Budget maintenance;
            end if
        end if
        Scale the weights $\theta_{t+1}^i = \frac{\theta_{t+1}^i}{\sum_{j=1}^m \theta_{t+1}^j}, i = 1, \ldots, m$
    end for
end for
```

budget size B for all the other compared budget algorithms the same as the total number of SVs yielded by our proposed SPA algorithms.

All experiments were repeated 10 times on different random permutations of instances and all the results were obtained by averaging over 10 runs. The algorithms were implemented in C++ on a PC with 3.2GHz CPU. We report the online mistake rates along the online learning process, the total number of SVs used by all single-kernel classifiers and the running time.

6.2.5 Evaluation by Comparing BOMKC(SPA) with Unbounded OMKC. The first experiment is to evaluate the performance of BOMKC(SPA) for binary classification tasks by comparing it with the existing unbounded OMKC algorithms. Note that we did not report the results on three large-scale datasets in this experiment, since it is simply computationally prohibited to run the non-budget algorithms on such large datasets due to time and memory limits. Table 5 summarizes the results. We can draw some observations below.

First, we compare the three unbounded OMKC algorithms using deterministic updates (Perceptron(*), OMKC(U), OMKC(DD)). Obviously, the mistake rate of OMKC(DD) is much lower than that of OMKC(U), which indicates that OMKC(DD) can build more effective multiple-kernel classifiers through learning the best combination. It is a bit surprised that, even with the unrealistic assumption of choosing the best kernel prior to online learning, the Perceptron algorithm using the best kernel did not achieve the lowest error rate. We conjecture that there might be two reasons. First, the optimal kernel is searched only on one random permutation, which might not be the best kernel for other permutations, while OMKC(DD) always learns the best combination of the kernel functions. Second, on some datasets, no single-kernel function has a significant advantage over the others, while an optimal weighted combination might outperform any single-kernel classifier. This further validates the significance of exploring multiple-kernel learning algorithms.

Second, we found that despite generating fewer SVs, the OMKC(SD) algorithms using stochastic updates sometimes achieve even lower mistake rates compared with OMKC(DD) using

Table 5. Evaluation of Online Classification on Small-scale and Medium-scale Datasets by Comparing BOMKC(SPA) with Unbounded OMKC Algorithms

	german			madelon		
Algorithm	Error(%)	#SVs	Time	Error(%)	#SVs	Time
Per(*)	31.87±1.61	318.8±16.1	0.04	48.65±1.57	973.0±31.4	0.81
U	35.75±1.52	6904.1±101.3	0.38	50.00±0.00	26498.2±92.6	35.27
DD	30.71±1.25	6904.1±101.3	0.38	41.15±0.50	26498.2±92.6	35.46
SD	34.83±1.54	5723.3±95.1	0.34	50.00±0.00	13872.3±150.6	20.67
DDPA	28.92±0.90	12481.5±60.9	0.70	**37.55±1.00**	28918.0±95.7	39.39
SDPA	28.88±0.63	6794.6±93.5	0.45	50.00±0.00	23837.2±29.7	33.93
SPA	**28.57±0.51**	2388.2±95.2	**0.16**	39.39±1.12	2285.2±68.8	**3.58**

	a9a			magic04		
Algorithm	Error(%)	#SVs	Time	Error(%)	#SVs	Time
Per(*)	20.43±0.13	9980.1±67.4	22.5	24.29±0.13	4620.9±25.9	1.94
U	20.60±0.10	234008.7±373.8	1178.5	28.70±0.12	157922.7±164.4	199.5
DD	19.20±0.12	234008.7±373.8	1178.6	22.58±0.46	157922.7±164.4	199.0
SD	18.93±0.10	155393.9±560.9	778.9	22.39±0.11	73433.4±320.1	69.4
DDPA	**15.82±0.07**	491137.2±237.2	3061.3	**18.91±0.10**	243950.0±106.9	352.8
SDPA	20.70±0.48	214751.2±474.8	1051.6	34.57±0.01	92635.9±560.1	94.6
SPA	16.31±0.11	14540±1779.4	**58.0**	19.62±0.23	7452.3±869.2	**5.10**

	KDD08			svmguide3		
Algorithm	Error(%)	#SVs	Time	Error(%)	#SVs	Time
Per(*)	0.94±0.01	963.5±11.0	14.7	25.98±0.51	323.0±6.3	0.03
U	0.66±0.01	32665.1±199.7	829.7	28.25±0.80	6166.9±100.1	0.36
DD	0.72±0.01	32665.1±199.7	829.0	25.41±0.95	6166.9±100.1	0.37
SD	0.63±0.01	17218.5±95.3	450.8	24.65±0.46	5550.8±95.8	0.36
DDPA	0.60±0.01	473444.2±124.6	12371	21.81±0.21	13734.9±61.6	0.87
SDPA	**0.59±0.01**	409441.0±7293.0	10834	**21.64±0.11**	12596.2±217.9	0.80
SPA	**0.59±0.01**	13749.1±858.2	**331.5**	22.21±0.24	2777.1±92.50	**0.20**

Time in seconds.

deterministic updates, which is consistent with the previous observations in Reference [20]. This indicates that not all SVs are essentially useful for constructing an accurate final classifier. This supports our main claim that when the added SVs are wisely selected, the accuracy may not be degraded much while the efficiency can be significantly improved.

Finally, we found that most PA-based algorithms significantly outperform the Perceptron-based ones in terms of accuracy, but paid by requiring much higher time and space costs due to more aggressive updates. We then make a comparison among the three PA-based algorithms. Although only a small fraction of SVs are adopted, the proposed SPA algorithm still achieves comparable or sometimes even better accuracy compared with its unbounded counterparts. Moreover, the time cost reduction by the proposed SV sampling strategy is especially more significant on the

Table 6. Evaluation of BOMKC Using Different Budget Learning Algorithms

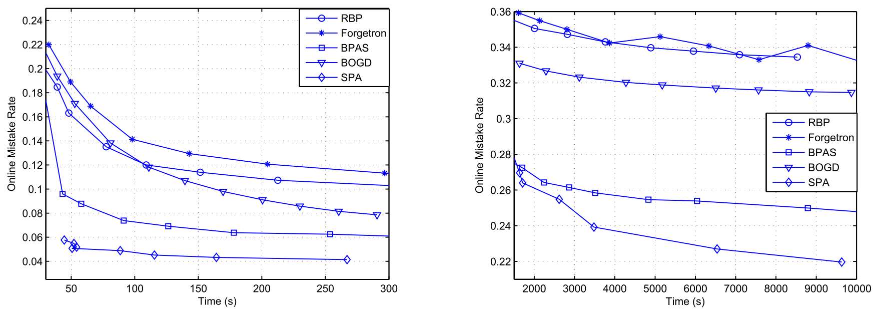
	KDD08			ijcnn1		
Algorithm	Error(%)	#SVs	Time	Error(%)	#SVs	Time
DD						
RBP	0.72±0.01	13744.0±0.0	520.6	9.23±0.54	4384.0±0.0	14.5
Forgetron	0.74±0.01	13744.0±0.0	537.0	9.27±0.49	4384.0±0.0	17.1
BOGD	0.61±0.00	13744.0±0.0	792.4	9.71±0.01	4384.0±0.0	17.0
BPAS	0.61±0.00	13744.0±0.0	718.0	9.70±0.01	4384.0±0.0	16.3
SD						
RBP	0.64±0.01	4384.0±0.0	385.2	7.82±0.18	4384.0±0.0	14.6
Forgetron	0.64±0.01	4384.0±0.0	380.2	8.21±0.05	4384.0±0.0	16.4
BOGD	0.61±0.00	13744.0±0.0	773.9	9.24±0.04	4384.0±0.0	16.7
BPAS	0.61±0.00	13744.0±0.0	758.8	9.71±0.01	4384.0±0.0	16.6
SPA	**0.59±0.01**	13749.1±858.2	**331.5**	**7.06±0.32**	4391.1±1164.7	**8.8**
	codrna			a9a		
Algorithm	Error(%)	#SVs	Time	Error(%)	#SVs	Time
DD						
RBP	12.44±0.27	5744.0±0.0	85.5	19.90±0.49	14544.0±0.0	81.5
Forgetron	13.39±0.28	5744.0±0.0	109.1	20.03±0.22	14544.0±0.0	118.4
BOGD	12.59±0.04	5744.0±0.0	98.9	17.12±0.08	14544.0±0.0	99.4
BPAS	7.06±0.15	5744.0±0.0	100.2	16.66±0.07	14544.0±0.0	95.1
SD						
RBP	9.06±0.29	5744.0±0.0	84.8	18.15±0.13	14544.0±0.0	78.2
Forgetron	10.24±0.26	5744.0±0.0	97.2	18.83±0.14	14544.0±0.0	96.5
BOGD	13.38±0.17	5744.0±0.0	98.0	17.46±0.11	14544.0±0.0	89.7
BPAS	10.59±0.20	5744.0±0.0	93.2	17.01±0.11	14544.0±0.0	93.4
SPA	**5.94±0.17**	5745.1±647.2	**46.2**	**16.31±0.11**	14540.0±1779.4	**58.0**

Time in seconds.

large datasets (e.g., KDD08 and a9a). In fact, when the datasets are very large (such as "Codrna" and "Susy," as shown in later experiments), it is even impossible for non-budget algorithms to complete the whole online learning process due extremely huge time cost and memory space needed for storing the unbounded SVs, which grows continuously in the online learning process for large-scale datasets.

6.2.6 Evaluation of Different Bounded OMKC Algorithms. The last experiment is to test the accuracy and efficiency of our proposed SPA algorithm in comparison to the other variants of Bounded OMKC algorithms using traditional budget maintenance strategies. Table 6 summarizes the experimental results on several large-scale datasets.

First, we compare the accuracy of the two groups of traditional budget algorithms, bounded OMKC(DD) using the Deterministic updating strategy (denoted as "DD" for short) and bounded OMKC(SD) using Stochastic updating strategy (denoted as "SD" for short). For the two



(a) codrna (b) susy

Fig. 3. Evaluation of different Bounded OMKC algorithms on two large-scale datasets. The curves of online mistake rates vs. time costs were obtained by choosing varied budget values. As bounded SD algorithms have no significant advantages over Bounded DD (see Table 6), we thus only include Bounded DD algorithms in these two figures to simplify the presentation.

Perceptron-based algorithms (RBP and Forgetron), the SD updating strategy demonstrates significant advantage in terms of accuracy over the DD strategy. This is due to the fact that SD focuses on the best kernels and spends only a small fraction of budget on poor kernels. While for the two aggressive algorithms (BOGD and BPAS), SD achieves comparable or even worse performance compared with DD. We conjecture that this may be because too aggressive update strategy results in sub-optimal budget allocation.

Second, it is clear that SPA not only achieves the best accuracy among all bounded OMKC algorithms, but also the lowest time cost. There are two major reasons for explaining the promising results: (i) Different from the budget DD that treats each kernel equally, SPA concentrates more effort on updating the classifiers of good kernels and thus the poor kernels will yield fewer SVs, which can greatly reduce the prediction time cost; (ii) Different from the budget SD where the classifier weights θ are also updated in a stochastic manner, the weights in the proposed SPA algorithm are updated always whenever a classification mistake appears, which leads to better SV allocation and more precise prediction combination.

To further examine the trade-off between classification accuracy and computational efficiency, Figure 3 shows the evaluation of different Bounded OMKC algorithms on two large-scale datasets, where the curves of online mistake rates versus time costs were obtained by choosing varied budget values (via proper parameter settings). As observed from the results, for the largest dataset "SUSY" with one million instances, we found that when following the previous setting where all the algorithms adopt the same number of SVs, the compared budget algorithms cannot finish processing the dataset in the fixed amount of time. Thus, for a fair comparison, we plot the online mistake rate of different algorithms on the same time axis according to different SV sparsity setting ($\beta = 2, \ldots, 10$). Obviously, the proposed SPA algorithm always achieves the lowest mistake rate with the given same amount of time cost, which validates the effectiveness and robustness of the proposed SPA algorithm.

7 CONCLUSIONS

To make large-scale kernel methods practical, we proposed a novel framework of SPA learning for bounded online kernel methods. Unlike traditional budget online kernel learning methods that rely on some budget maintenance strategies, the proposed algorithm adopts a simple yet effective stochastic sampling strategy to ensure the number of support vectors is bounded. In contrast to traditional budget online kernel algorithms that only bound the number of support vectors at each iteration, our new algorithm outputs a sparse final averaged classifier with bounded support vector size, making it not only suitable for online learning tasks but also applicable to batch classification. We also extended the proposed SPA algorithm for BOMKC tasks by learning with multiple kernels. Our promising experimental results showed that our SPA algorithms outperform a variety of state-of-the-art budget online learning algorithms in both single-kernel and multiple-kernel settings, validating the efficacy, efficiency and scalability of the proposed SPA technique.

REFERENCES

[1] Francis R. Bach. 2008. Consistency of the group lasso and multiple-kernel learning. *J. Mach. Learn. Res.* 9 (2008), 1179–1225.

[2] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. 2004. Multiple-kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the 21st International Conference on Machine Learning*. ACM, 6.

[3] Colin Campbell. 2002. Kernel methods: A survey of current techniques. *Neurocomputing* 48, 1 (2002), 63–84.

[4] Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. 2007. Tracking the best hyperplane with a simple budget perceptron. *Mach. Learn.* 69, 2–3 (2007), 143–167.

[5] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. 2004. On the generalization ability of on-line learning algorithms. *IEEE Trans. Info. Theory* 50, 9 (2004), 2050–2057.

[6] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. (TIST)* 2, 3 (2011), 27.

[7] Ning Chen, Steven C. H. Hoi, Shaohua Li, and Xiaokui Xiao. 2015. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*. ACM, 305–314.

[8] Ning Chen, Steven C. H. Hoi, Shaohua Li, and Xiaokui Xiao. 2016. Mobile app tagging. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*. ACM, 63–72.

[9] Corinna Cortes and Vladimir Vapnik. 1995. Support vector machine. *Mach. Learn.* 20, 3 (1995), 273–297.

[10] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *J. Mach. Learn. Res.* 7 (2006), 551–585.

[11] Koby Crammer, Jaz S. Kandola, and Yoram Singer. 2003. Online classification on a budget. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'03)*, Vol. 2. 5.

[12] Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.* 3 (2003), 951–991.

[13] Ofer Dekel. 2009. From online to batch learning with cutoff-averaging. In *Adv. Neural Info. Process. Syst.* 377–384.

[14] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2008. The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput.* 37, 5 (2008), 1342–1372. DOI: http://dx.doi.org/10.1137/060666998

[15] Ofer Dekel and Yoram Singer. 2005. Data-driven online to batch conversions. In *Adv. Neural Info. Process. Syst.* 267–274.

[16] Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.

[17] Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Mach. Learn.* 37, 3 (1999), 277–296.

[18] Mehmet Gönen and Ethem Alpaydın. 2011. Multiple-kernel learning algorithms. *J. Mach. Learn. Res.* 12 (2011), 2211–2268.

[19] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. 2008. Kernel methods in machine learning. *Ann. Stat.* (2008), 1171–1220.

[20] Steven C. H. Hoi, Rong Jin, Peilin Zhao, and Tianbao Yang. 2013. Online multiple-kernel classification. *Mach. Learn.* 90, 2 (2013), 289–316.

[21] Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. 2014. LIBOL: A library for online learning algorithms. *J. Mach. Learn. Res.* 15, 1 (2014), 495–499.

[22] Rong Jin, Steven C. H. Hoi, and Tianbao Yang. 2010. Online multiple-kernel learning: Algorithms and mistake bounds. In *Algorithmic Learning Theory*. Springer, 390–404.

[23] Jyrki Kivinen, Alex J. Smola, and Robert C. Williamson. 2001. Online learning with kernels. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'01)*. 785–792.

[24] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. 2004. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.* 5 (2004), 27–72.

[25] Jing Lu, Peilin Zhao, and Steven C. H. Hoi. 2016. Online sparse passive-aggressive learning with kernels. In *Proceedings of the 2016 SIAM International Conference on Data Mining (SDM'16).*

[26] Jie Luo, Francesco Orabona, Marco Fornoni, Barbara Caputo, and Nicolo Cesa-Bianchi. 2010. OM-2: An online multi-class multi-kernel learning algorithm. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'10), Online Learning for Computer Vision Workshop.*

[27] André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mario A. T. Figeuiredo. 2011. Online learning of structured predictors with multiple kernels. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics.* 507–515.

[28] Francesco Orabona, Luo Jie, and Barbara Caputo. 2010. Online-batch strongly convex multi kernel learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10).* IEEE, 787–794.

[29] Francesco Orabona, Luo Jie, and Barbara Caputo. 2012. Multi kernel learning with online-batch optimization. *J. Mach. Learn. Res.* 13, 1 (2012), 227–253.

[30] Francesco Orabona, Joseph Keshet, and Barbara Caputo. 2009. Bounded kernel-based online learning. *J. Mach. Learn. Res.* 10 (2009), 2643–2666.

[31] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. 2007. More efficiency in multiple-kernel learning. In *Proceedings of the 24th International Conference on Machine Learning.* ACM, 775–782.

[32] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* 65 (1958), 386–407.

[33] Doyen Sahoo, Steven C. H. Hoi, and Bin Li. 2014. Online multiple-kernel regression. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 293–302.

[34] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1997. Kernel principal component analysis. In *Proceedings of the Conference on Artificial Neural Networks (ICANN'97).* Springer, 583–588.

[35] Bernhard Scholkopf and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press.

[36] Shai Shalev-Shwartz. 2011. Online learning and online convex optimization. *Found. Trends Mach. Learn.* 4, 2 (2011), 107–194.

[37] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. 2007. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the International Conference on Machine Learning (ICML'07).* 807–814.

[38] Ohad Shamir and Tong Zhang. 2012. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *arXiv:1212.1824.*

[39] John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis.* Cambridge University Press.

[40] Alex J. Smola and Bernhard Schölkopf. 1998. *Learning with Kernels.* Citeseer.

[41] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. 2006. Large-scale multiple-kernel learning. *J. Mach. Learn. Res.* 7 (2006), 1531–1565.

[42] Niranjan Subrahmanya and Yung C. Shin. 2010. Sparse multiple-kernel learning for signal processing applications. *IEEE Trans. Pattern Anal. Mach. Intell.* 32, 5 (2010), 788–798.

[43] Vladimir Naumovich Vapnik and Vlamimir Vapnik. 1998. Statistical learning theory. Vol. 1. Wiley New York.

[44] Manik Varma and Bodla Rakesh Babu. 2009. More generality in efficient multiple-kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09).* ACM, 1065–1072.

[45] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. 2012. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale SVM training. *J. Mach. Learn. Res.* 13, 1 (2012), 3103–3131.

[46] Zhuang Wang and Slobodan Vucetic. 2009. Twin vector machines for online learning on a budget. In *Proceedings of the SIAM International Conference on Data Mining (SDM'09).* SIAM, 906–917.

[47] Zhuang Wang and Slobodan Vucetic. 2010. Online passive-aggressive algorithms on a budget. In *Proceedings of the International Conference on Artificial Intelligence and Statistics.* 908–915.

[48] Hao Xia, Steven C. H. Hoi, Rong Jin, and Peilin Zhao. 2014. Online multiple-kernel similarity learning for visual search. *IEEE Trans. Patt. Anal. Mach. Intell.* 36, 3 (2014), 536–549.

[49] Shi Yu, Tillmann Falck, Anneleen Daemen, Leon-Charles Tranchevent, Johan A. K. Suykens, Bart De Moor, and Yves Moreau. 2010. L 2-norm multiple-kernel learning and its application to biomedical data fusion. *BMC Bioinformat.* 11, 1 (2010), 1.

[50] Lijun Zhang, Rong Jin, Chun Chen, Jiajun Bu, and Xiaofei He. 2012. Efficient online learning for large-scale sparse kernel logistic regression. In *Proceedings of the Conference on Artificial Intelligence, the Innovative Application of Artificial Intelligence (AAAI'12).*

[51] Lijun Zhang, Jinfeng Yi, Rong Jin, Ming Lin, and Xiaofei He. 2013. Online kernel learning with a near optimal sparsity bound. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13).* 621–629.

[52] Peilin Zhao, Steven C. H. Hoi, and Rong Jin. 2011. Double updating online learning. *J. Mach. Learn. Res.* 12 (2011), 1587–1615.
[53] Peilin Zhao, Jialei Wang, Pengcheng Wu, Rong Jin, and Steven C. H. Hoi. 2012. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. In *Proceedings of the International Conference on Machine Learning (ICML'12)*.

Received February 2017; revised September 2017; accepted October 2017