MKBoost: A Framework of Multiple Kernel Boosting

Hao Xia, Steven C. H. Hoi

Abstract—Multiple kernel learning (MKL) is a promising family of machine learning algorithms using multiple kernel functions for various challenging data mining tasks. Conventional MKL methods often formulate the problem as an optimization task of learning the optimal combinations of both kernels and classifiers, which usually results in some forms of challenging optimization tasks that are often difficult to be solved. Different from the existing MKL methods, in this paper, we investigate a boosting framework of multiple kernel learning for classification tasks, i.e., we adopt boosting to solve a variant of MKL problem, which avoids solving the complicated optimization tasks. Specifically, we present a novel framework of Multiple Kernel Boosting (MKBoost), which applies the idea of boosting techniques to learn kernel-based classifiers with multiple kernels for classification problems. Based on the proposed framework, we propose several variants of MKBoost algorithms and extensively examine their empirical performance on a number of benchmark datasets in comparisons to various state-of-the-art MKL algorithms on classification tasks. Experimental results show that the proposed method is more effective and efficient than the existing MKL techniques.

Index Terms—Multiple kernel learning, boosting, kernel methods, classification

1 Introduction

Kernel methods have been extensively studied in data mining and machine learning for their proven state-of-the-art performance in many real data analysis applications [26], [27]. For any kernel method, a core element is the kernel function $\kappa(x, x')$, which measures the similarity between two data examples x and x'. Many kernel methods usually adopt a single predefined kernel function, for example, a linear kernel or a gaussian kernel. However, in many real-world scenarios, it is usually not enough for employing a single predefined kernel function since real data may come from multiple diverse sources or could be given in terms of different kinds of representations. This has motivated the need of studying machine learning techniques for combining multiple kernels to achieve better capability and higher flexibility in solving real-world challenges. Such a learning problem is often known as "Multiple Kernel Learning" (MKL) in machine learning and data mining areas [28].

Recent years have witnessed a surge of studies that have attempted to explore a variety of techniques to resolve the Multiple Kernel Learning problems [25], [28], [34]. Similar to some well-known kernel methods, e.g., Support Vector Machines (SVM) [32], MKL has been demonstrated as a promising technique with state-of-the-art performance for solving many real-world applications, especially for its power of exploiting multiple kernels in fusing diverse information from multiple sources. For instance, MKL techniques have been applied to resolve the challenges of protein-protein interaction extraction in Bioinformatics [23], speech recognition in signal processing [22], and anomaly detection in data mining [9].

School of Computer Engineering, Nanyang Technological University, Singapore 639798, E-mail: xiah0002@ntu.edu.sg
School of Computer Engineering, Nanyang Technological University, Singapore 639798, E-mail: chhoi@ntu.edu.sg

Despite being studied actively, the regular MKL methods have some critical limitations. Specifically, most existing MKL methods are often formulated as a complicated optimization task, typically convex optimization, e.g., a Semi-Definite Program (SDP) [20], which is then resolved by applying some existing optimization techniques. Despite their nice convex formulation, it is often a great challenge for solving such complicated optimization tasks. Recently, a surge of efforts have attempted to improve the efficiency of the optimization task by various techniques [28], [34]. For instance, the study in [28] formulated the MKL problem as a min-max optimization task, and proposed to find the saddle-point solution by solving a Semi-Infinite Linear Program (SILP). Although some encouraging progress has been made in this research direction, low efficiency and scalability issues remain the key limitations of the existing regular MKL methods towards a real large application.

In addition, despite engaging a complicate optimization process, the final classifier learned by the regular MKL methods is a single kernel-based classifier based on a kernel that is a linear combination of multiple kernels. Such a single kernel-based classifier might not be effective enough to handle diverse patterns and complex decision boundaries in real applications.

To address the above limitations of the conventional MKL methods, in this paper, we investigate a new framework of multiple kernel learning, termed "Multiple Kernel Boosting" (MKBoost), which adapts the idea of boosting for learning classifiers with multiple kernels for classification tasks. The intuitive idea of multiple kernel boosting is to employ the boosting framework to learn an ensemble of multiple base kernel classifiers, each of them is learned from a single kernel. The combination weights for both the kernels and classifiers can be efficiently determined through

the learning process of boosting. Consequently, we can efficiently learn a classifier with multiple kernels without resolving a complicated optimization task as required in a regular MKL approach. In addition, unlike the regular MKL methods that only learn a single kernel-based classifier, MKBoost learns an ensemble of multiple kernel-based classifiers, in which each individual classifier is trained on a single kernel and the final classifier is obtained by a weighted combination of all classifiers according to their performance.

As a summary, the main contributions of our work include the following: (i) we present a novel framework of Multiple Kernel Boosting (MKBoost) for MKL, which offers a new approach to learning kernel based classifiers with multiple kernels efficiently; (ii) we propose two kinds of MKBoost approaches: deterministic and stochastic algorithms, which attempt to resolve the MKBoost problem to trade off between accuracy and efficiency; (iii) we conduct extensive experiments for empirically validating the performance of the proposed MKBoost algorithms by comparing with various state-of-the-art MKL algorithms. We note that a short version of this journal had been presented in the SDM'11 conference [33].

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the proposed framework of multiple kernel boosting, gives two kinds of different approaches for solving the MKBoost problem, and briefly analyzes the time and space complexity. Section 4 discusses our empirical study. Section 5 concludes this paper.

2 Related Work

Kernel methods have been extensively studied in literature [27], [8]. Most kernel methods usually assume some predefined parametric kernel, e.g., a gaussian kernel, is given a priori, where the parameters of the kernel function are usually determined empirically by cross validation. Several studies have been proposed to learn parametric, semi-parametric, or nonparametric kernel functions/matrices from labeled and/or unlabeled data. Example techniques include diffusion kernels [19], marginalized kernels [17], spectral kernel learning [36], [13], [4], nonparametric kernel learning [12], [37], and so on.

Recently, Multiple Kernel Learning (MKL) [20], [28], [34], [16], [39], [38] has been actively studied, aiming to learn kernel based models by finding the optimal combination of multiple predefined kernels for classification tasks. Example algorithms include MKL by SDP [20], MKL by SILP [28], MKL by sub-gradient descent [25], MKL by the level based optimization [34], and MKL by the second-order Newton method [6]. Recently, some emerging works also attempt to improve the efficacy of regular MKL. For example, the ℓ_p-norm MKL method extended the regular ℓ_1-norm MKL for arbitrary ℓ_p-norm MKL with $p > 1$. Besides, some recent studies [40], [15], [30] also attempt to address other issues of MKL, such as multi-class and multi-labeled classification. Unlike the existing MKL methods that have to solve complicated optimization tasks, our method employs a boosting approach, which is more efficient and scalable.

Moreover, our work is related to some existing works that adopt the idea of boosting for learning mixture-of-kernels models [1], [2], [3]. In [1], the authors addressed a kernel-based regression problem, and applied the idea of gradient boosting/column generation to optimize a heterogeneous kernel using a gradient descent algorithm in function space, which was named as the Multiple Additive Regression Kernels (MARK) algorithm. Unlike the MARK algorithm that were designed to mainly address a regression task, the authors in [2] proposed the column generation (CG) boosting algorithms from a family of kernels for both classification and regression tasks, which boost on kernel columns by employing the previous column generation (CG) technique in [10]. The similar idea was also extended to a semi-supervised learning setting in [3]. Our work differs from these studies in that they all are based on the column generation idea, which usually engages a huge number of columns and is thus computationally very intensive.

Our work is also related to some existing work using the idea of boosting for kernel design [7]. The authors in [7] formulated the kernel design problem as the construction of an accurate kernel from simple base kernels, and employ the boosting idea to perform the kernel construction process. Similar to the regular MKL method, the kernel design approach using boosting [7] also learns a linear combination of kernels. However, their approach is limited in that it learns the single base kernel from empirical data in a transductive learning setting, while regular MKL and our approach learn from a set of predefined kernels in an inductive learning setting, which can be more general and flexible to handle multiple sources of data in real applications.

Finally, there are some works which adopt boosting technique to improve kernel methods, such as, Boost-SMO [24], Boost-SVM [35], AdaBoostSVM [29], [21], AdaBoost with SVM [31], etc. But they are all unable to deal with multiple kernels that may encode information originates from multiple resources.

3 Multiple Kernel Boosting

In this section, we present a novel framework of multiple kernel boosting (MKBoost), which adapts boosting techniques for multiple kernel learning. Before presenting our MKBoost algorithms, we will first introduce the problem and review the regular multiple kernel learning (MKL). To ease our presentation, we will restrict our discussion on a typical binary classification task. Similar algorithms in this work can be easily extended to multi-class classification tasks.

3.1 Problem Formulation

Consider a given set of training examples $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, N\}$ where $y_i \in \{-1, +1\}, i = 1, \ldots, N$, and a collection of M kernel functions $\mathcal{K} = \{\kappa_j : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, j = 1, \ldots, M\}$.

The goal of our multiple kernel boosting task is to learn a kernel based classifier f, which is an ensemble of kernel classifiers using the collection of M kernels trained from the given training data examples. Typically, we express such a kernel classifier as:

$$f(x_i) = \sum_{t=1}^{T} \alpha_t f_t(x_i) \tag{1}$$

where f_t is a hypothesis learned from a boosting trial, α_t is its associated weight in the final classifier, and T is the total number of boosting trials. The main challenge of multiple kernel boosting is to develop an effective boosting scheme to learn the optimal hypothesis f_t and its combination weight α_t at each boosting trial.

For instance, if we consider only a single kernel κ, f_t can be learned by applying any regular kernel classifier such as SVM from a subset of n training examples in a boosting trial, i.e.,

$$f_t(\mathbf{x}) = \sum_{i=1}^{n} \beta_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) \tag{2}$$

where β_i's are the coefficients of an SVM model. Such an approach reduces to a regular boosting approach using a single kernel SVM as the base classifier.

In this paper, we present two kinds of approaches to solving the MKBoost problem. Before presenting our algorithms, we first review some basics of regular MKL to better understand the motivation and difference of our work.

3.2 Multiple Kernel Learning

The goal of a regular MKL task is to identify the optimal combination of M kernels, denoted by $\theta = (\theta_1, \ldots, \theta_M)$ by following maximum margin learning principle, which can be cast into the following optimization problem:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_{K(\theta)}} \frac{1}{2}\|f\|^2_{\mathcal{H}_{K(\theta)}} + C \sum_{i=1}^{N} \ell(f(x_i), y_i) \tag{3}$$

where $\Delta = \{\theta \in \mathbb{R}^M_+ | \theta^\top e_M = 1\}$, $K(\theta)(\cdot, \cdot) = \sum_{j=1}^{M} \theta_j \kappa_j(\cdot, \cdot)$, $\ell(f(x_i), y_i) = \max(0, 1 - y_i f(x_i))$.

In the above, we use notation e_M to represent a vector of M dimensions with all its elements being 1. The above formulation can also be turned into the following min-max optimization task:

$$\min_{\theta \in \Delta} \max_{\alpha \in \Xi} \left\{ \alpha^\top e_N - \tfrac{1}{2}(\alpha \circ \mathbf{y})^\top \left(\sum_{j=1}^{M} \theta_j K^j \right) (\alpha \circ \mathbf{y}) \right\}$$

where $K^j \in \mathbb{R}^{N \times N}$ with $K^j_{p,q} = \kappa_i(x_p, x_q)$, $\Xi = \{\alpha | \alpha \in [0, C]^N\}$, and $\circ$ defines the element-wise product between two vectors. Despite some encouraging results achieved recently [25], [34], developing an efficient and scalable algorithm for solving the regular MKL problem remains an open challenge.

3.3 Deterministic MKBoost Algorithms

The general idea of MKBoost is to apply boosting techniques to learn a classifier using multiple kernels. To this purpose, we follow the typical procedure of a popular and successful boosting algorithm, i.e., Adaptive Boosting known as "Adaboost" [11].

In particular, we repeatedly learn some kernel classifiers with multiple kernels f_t through a series of boosting trials $t = 1, \ldots, T$, where T denotes the total number of boosting trials. At each boosting trial, a distribution of weights D_t is engaged to indicate the importance of the training examples for learning. At each trial, we increase the weights of the wrongly classified examples and/or decrease the weights of those correctly classified examples in order to focus on those examples that are hard to be correctly classified.

During each boosting trial, we first sample a subset of n training examples according to distribution D_t, where n is specified by a predefined *boosting sampling ratio* that determines the proportion of training data to be sampled at each boosting trial. Once obtaining the subset of training data, the next key issue is how to learn the kernel based classifier f_t from these training data. The first approach is to learn one classifier f^j_t with each kernel κ_j from the set of M kernels using a regular kernel method, e.g., SVM used in our study. Based on the set of M base classifiers, we can further measure the misclassification performance of each classifier f^j_t with kernel κ_j over distribution D_t of the whole collection of training data:

$$\epsilon^j_t = \epsilon(f^j_t) = \sum_{i=1}^{N} D_t(i)(f^j_t(x_i) \neq y_i) \tag{4}$$

As a result, we can build the classifier f_t for the t-th boosting trial by choosing the best classifier with the smallest misclassification rate, i.e.,

$$f_t = \arg \min_{f^j_t, j \in \{1, \ldots, M\}} \epsilon(f^j_t) \tag{5}$$

The next step follows the similar procedure of Adaboost by computing the misclassification rate ϵ_t for the combined classifier f_t over the distribution D_t on the whole collection of training data:

$$\epsilon_t = \sum_{i=1}^{N} D_t(i)(f_t(x_i) \neq y_i) \tag{6}$$

The last step of each boosting trial is to update the weight of each training example $D_{t+1}(i)$ as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & if\ f_t(x_i) = y_i \\ e^{\alpha_t} & if\ f_t(x_i) \neq y_i \end{cases}$$

Algorithm 1 The MKBoost algorithm (MKBoost-D1)

INPUT:
- training data: $(x_1, y_1), \ldots, (x_N, y_N)$
- kernel functions: $\kappa_j(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, j = 1, \ldots, M$
- initial distribution $D_1(i) = 1/N, i = 1, \ldots, N$

for $t = 1, \ldots, T$ **do**
  sample n examples using distribution D_t
  for $j = 1, \ldots, M$ **do**
    train weak classifier with kernel κ_j: $f_t^j : \mathcal{X} \to \{-1, +1\}$
    compute the training error over D_t: $\epsilon_t^j = \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i)$
  end for
  select the best classifier with the minimal error rate

$$f_t = \arg\min_{f_t^j} \epsilon_t^j = \arg\min_{f_t^j} \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i),$$

  choose $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$, where $\epsilon_t = \min_{j \in \{1,\ldots,M\}} \epsilon_t^j$
  Update $D_{t+1}(i)$:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & if\ f_t(x_i) = y_i \\ e^{\alpha_t} & if\ f_t(x_i) \neq y_i \end{cases}$$

  Z_t is a normalization factor to make D_t a distribution.
end for
OUTPUT: $f(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t f_t(x)\big)$

Algorithm 2 The MKBoost Algorithm (MKBoost-D2)

INPUT:
- training data: $(x_1, y_1), \ldots, (x_N, y_N)$
- kernel functions: $\kappa_j(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, j = 1, \ldots, M$
- initial distribution $D_1(i) = 1/N, i = 1, \ldots, N$

for $t = 1, \ldots, T$ **do**
  sample a set of n examples using distribution D_t
  for $j = 1, \ldots, M$ **do**
    train weak classifier with kernel κ_j: $f_t^j : \mathcal{X} \to \{-1, +1\}$
    compute the training error over D_t: $\epsilon_t^j = \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i)$
    choose $\alpha_t^j = \frac{1}{2}\ln(\frac{1-\epsilon_t^j}{\epsilon_t^j})$
  end for
  combine the M classifiers: $f_t(x) = \text{sign}\big(\sum_{j=1}^{M} \alpha_t^j f_t^j(x)\big)$
  compute the training error over D_t: $\epsilon_t = \sum_{i=1}^{N} D_t(i)(f_t(x_i) \neq y_i)$
  choose $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
  Update $D_{t+1}(i)$:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & if\ f_t(x_i) = y_i \\ e^{\alpha_t} & if\ f_t(x_i) \neq y_i \end{cases}$$

  Z_t is a normalization factor to make D_t a distribution.
end for
OUTPUT: $f(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t f_t(x)\big)$

where $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ and Z_t is a normalization factor to make D_{t+1} a distribution.

Finally, after finishing all T boosting trials, the algorithm outputs the final classifier as follows:

$$f(x) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t f_t(x)\Big) \tag{7}$$

We refer to the above MKBoost algorithm as "MKBoost-D1" for short. The details of the proposed algorithm are shown in Algorithm 1.

In the above algorithm, at each boosting trial, we simply choose the best classifier among the M kernel classifiers as the classifier and simply discard the other $M-1$ classifiers. In some situation, it is possible that other kernel classifiers may make complementary contribution in improving the performance. Thus, we propose another way to build the classifier by combining all these M classifiers, each of which is assigned with a weight. Specifically, we suggest to build the classifier at the t-th boosting trial:

$$f_t(x) = \text{sign}\Big(\sum_{j=1}^{M} \alpha_t^j f_t^j(x)\Big) \tag{8}$$

where the weight α_t^j is computed based on the misclassification rate ϵ_t^j, i.e.,

$$\alpha_t^j = \frac{1}{2}\ln\Big(\frac{1-\epsilon_t^j}{\epsilon_t^j}\Big) \tag{9}$$

We refer to this MKBoost algorithm as "MKBoost-D2" for short. The other part of this algorithm is same to MKBoost-D1. Algorithm 2 gives the details of the proposed MKBoost-D2 algorithm.

3.4 Stochastic MKBoost Algorithms

One important limitation for the above two MKBoost algorithms is that they need to repeatedly train a set of M kernel classifiers at each boosting trial. This could be computationally intensive if the number of kernels M is large in some applications. To address this limitation, in this section, we propose a *stochastic* learning approach for MKBoost, which aims to avoid the need of training all the M kernel classifiers. To ease our presentation, we refer to the new algorithms as *stochastic* MKBoost algorithms, and the previous two algorithms as *deterministic* MKBoost algorithms.

The intuitive idea of a stochastic MKBoost approach is that we could try to avoid unnecessary costs of training classifiers with some kernels that have relatively poor classification performance for the classification task. To this purpose, we introduce a variable $S_t(j)$ as the kernel sampling probability, which indicates how likely a kernel κ_j will be sampled at the t-th boosting trial. At the beginning of the MKBoost algorithm, all $S_1(j)$ values are set to 1, which means that all M kernels will be definitely selected at the first boosting trial.

Algorithm 3 The MKBoost algorithm (MKBoost-S1)

INPUT:
- training data: $(x_1, y_1), \ldots, (x_N, y_N)$
- kernel functions: $\kappa_j(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, j = 1, \ldots, M$
- init. data distribution $D_1(i) = 1/N, i = 1, \ldots, N$
- init. kernel sampling prob. $S_1(j) = 1, j = 1, \ldots, M$
- sampling decay rate $0 < \beta < 1$

for $t = 1, \ldots, T$ **do**
  Sample n examples using distribution D_t
  Sample a set of kernels $\mathcal{K}_t$ by sampling probability S_t
  for $\kappa_j \in \mathcal{K}_t$ **do**
    Train weak classifier with kernel κ_j: $f_t^j : \mathcal{X} \to \{-1, +1\}$
    Compute the training error over D_t: $\epsilon_t^j = \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i)$
    Update $S_{t+1}(j) \leftarrow S_t(j)\beta^{\epsilon_t^j}$
  end for
  Update $S_{t+1}(j) \leftarrow S_{t+1}(j)/Z^S$, $Z^S = \max(S_{t+1})$
  $f_t = \arg\min_{\kappa_j \in \mathcal{K}_t} \epsilon_t^j = \arg\min_{\kappa_j \in \mathcal{K}_t} \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i)$
  Compute the training error over D_t: $\epsilon_t = \sum_{i=1}^{N} D_t(i)(f_t(x_i) \neq y_i)$
  $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
  Update $D_{t+1}(i) \leftarrow \frac{D_t(i)}{Z_t}\exp(-\alpha_t y_i f_t(x_i))$
end for
OUTPUT: $f(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t f_t(x)\big)$

Algorithm 4 The MKBoost algorithm (MKBoost-S2)

INPUT:
- training data: $(x_1, y_1), \ldots, (x_N, y_N)$
- kernel functions: $\kappa_j(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, j = 1, \ldots, M$
- init. data distribution $D_1(i) = 1/N, i = 1, \ldots, N$
- init. kernel sampling prob. $S_1(j) = 1, j = 1, \ldots, M$
- sampling decay rate $0 < \beta < 1$

for $t = 1, \ldots, T$ **do**
  Sample n examples using distribution D_t
  Sample a set of kernels $\mathcal{K}_t$ by sampling probability S_t
  for $\kappa_j \in \mathcal{K}_t$ **do**
    Train weak classifier with kernel κ_j: $f_t^j : \mathcal{X} \to \{-1, +1\}$
    Compute the training error over D_t: $\epsilon_t^j = \sum_{i=1}^{N} D_t(i)(f_t^j(x_i) \neq y_i)$
    Choose $\alpha_t^j = \frac{1}{2}\ln(\frac{1-\epsilon_t^j}{\epsilon_t^j})$
    Update $S_{t+1}(j) \leftarrow S_t(j)\beta^{\epsilon_t^j}$
  end for
  Update $S_{t+1}(j) \leftarrow S_{t+1}(j)/Z^S$, $Z^S = \max(S_{t+1})$
  $h_t(x) = \text{sign}\big(\sum_{\kappa_j \in \mathcal{K}_t} \alpha_t^j f_t^j(x)\big)$
  Compute the training error over D_t: $\epsilon_t = \sum_{i=1}^{N} D_t(i)(f_t(x_i) \neq y_i)$
  Choose $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
  Update $D_{t+1}(i) \leftarrow \frac{D_t(i)}{Z_t}\exp(-\alpha_t y_i f_t(x_i))$
end for
OUTPUT: $f(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t f_t(x)\big)$

For each boosting trial, we sample a subset of kernels according to the kernel sampling probability S_t. The proposed MKBoost algorithm will train kernel classifiers only for those selected kernels. At the end of each boosting trial, we update the kernel sampling probability according to its classification performance:

$$S_{t+1}(j) \leftarrow S_t(j)\beta^{\epsilon_t^j} \tag{10}$$

where $\beta \in (0, 1)$ is a constant parameter introduced as a sampling decay factor for updating the kernel sampling probability, and ϵ_t^j is the misclassification rate of the kernel classifier with a selected kernel κ_j. The above formula indicates the larger the misclassification rate, the more decay penalty will be applied to the kernel to reduce the chance of being sampled in the next trial. Finally, at the end of a boosting trial, we conduct a normalization step by ensuring all kernel sampling weights are in $[0, 1]$. This normalization step could affect the sampling weights of those kernels that are not selected.

By incorporating the above kernel sampling scheme, we suggest two stochastic MKBoost algorithms, MKBoost-S1 and MKBoost-S2, which are corresponding to the previous two deterministic MKBoost algorithms, respectively. The details of these two algorithms are shown in Algorithm 3 and 4.

3.5 Time and Space Complexity

Compared with the regular MKL methods, the MKBoost algorithms enjoy significant advantages in their high efficiency and scalability performance.

Time Complexity. We first denote by $\mathcal{C}(n)$ the time complexity of training a base kernel classifier from a collection of n examples, where n is specified by the boosting sampling ratio. As we adopt SVM based on the SMO solver in the LIBSVM library [5], the empirical complexity of $\mathcal{C}(n)$ is about $O(n^{1.4})$ to $O(n^{2.3})$, depending on different datasets and parameter settings. Since all the proposed MKBoost algorithms follow the similar framework, they in general share the same worst case time complexity, i.e., $O(T \times M \times (N + \mathcal{C}(n)))$, where T is the total number of boosting trials, M is the total number of kernels, and N is total number of training examples. For a large-scale application, we could choose a small value of n ($n \ll N$), making the algorithms efficient and scalable to large classification tasks. Finally, comparing the deterministic and stochastic variants of the MKBoost algorithms, despite sharing the same worst case time complexity, the stochastic algorithms are empirically more efficient than the deterministic ones because the number of sampled kernels is usually smaller than M especially in the later boosting stages.

Space Complexity. We denote by $\mathcal{S}(n)$ the space complexity of a base kernel classifier trained from a set of n examples, which is typically equal to or

smaller than $O(n)$. For the proposed MKBoost algorithms, MKBoost-D1 and MKBoost-S1 share the same worst case space complexity, i.e., $O(T \times \mathcal{S}(n))$, because both of them choose the best kernel classifier at each boosting trial. In contrast, MKBoost-D2 and MKBoost-S2 have higher space complexity. Specifically, both of them have the worst case space complexity of $O(T \times M \times S(n))$, but the empirical space complexity of MKBoost-S2 is lower than that of MKBoost-D2 because MKBoost-S2 usually samples a number of kernels that is smaller than M at each boosting trial.

4 Experiments

In this section, we conduct an extensive set of experiments to examine the classification performance of the proposed MKBoost algorithms by comparing with a number of state-of-the-art techniques in literature. All the source code and datasets are available at http://www.cais.ntu.edu.sg/~chhoi/mkboost/.

4.1 Experimental Testbed

We evaluate the performance of MKBoost algorithms for both binary and multiclass classification tasks. We randomly choose a number of publicly available datasets from web machine learning repositories, which can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. TABLE 1 shows the summary of the datasets in our experiments. Note that the sizes of these datasets are not very large because we want to compare our algorithms with other existing MKL algorithms, most of them are usually not scalable and only applicable to relatively small-sized datasets. In general, our algorithms are efficient and scalable for large applications.

TABLE 1: Summary of the datasets (where C, N, D denote # classes, #examples, #features, respectively).

Abbr.	name	source	C	N	D
D1	a1a	UCI	2	1605	123
D2	a3a	UCI	2	3185	123
D3	a4a	UCI	2	4781	123
D4	balance-scale	UCI	2	576	4
D5	german.numer	Statlog	2	1000	24
D6	glass	UCI	2	214	10
D7	ionosphere	UCI	2	351	34
D8	monks1	UCI	2	556	6
D9	monks2	UCI	2	1033	6
D10	satimage	Statlog	6	4435	36
D11	segment	Statlog	7	2310	19
D12	sonar	UCI	2	208	60
D13	svmguide3	CWH03a	2	1243	21
D14	svmguide4	CWH03a	6	300	10
D15	vehicle	Statlog	4	846	18
D16	wdbc	UCI	2	569	30

4.2 Comparison Schemes

In our experiments, we compare the proposed MKBoost algorithms against several state-of-the-art MKL algorithms using different optimization techniques. Besides, we also compare with some boosting based kernel methods. The regular MKL algorithms in our comparisons include:

- MKL-SD: An MKL algorithm solved by a Subgradient Descent (SD) optimization method [25].
- MKL-SILP: An MKL algorithm solved by a Semi-Infinite Linear Program (SILP) [28].
- MKL-Level: An MKL algorithm solved by an extended level optimization method [34].
- MKL-Hessian: An MKL algorithm solved by a second order Newton update optimization method [6].
- Lp-MKL: An extended MKL algorithm that generalizes the regular ℓ_1-norm MKL method to arbitrary ℓ_p-norm MKL [18].

The existing boosting based algorithms include:

- Boost-Exp: An algorithm uses boosting paradigm to perform the kernel construction process [7], with ExpLoss: $exp(-yf(x))$.
- Boost-Log: An algorithm uses boosting paradigm to perform the kernel construction process [7], with Logloss: $log(1 + exp(-yf(x)))$.
- Boost-CG-L1: An algorithm uses column-generation boosting methods to learn mixture of kernels [2], with L1 norm.
- Boost-CG-L2: An algorithm uses column-generation boosting methods to learn mixture of kernels [2], with L2 norm.
- Boost-SVM: An algorithm uses Adaboost to improve SVM learning accuracy [35].

4.3 Experimental Setup

For the setup of our experiments, we follow the typical approach used in the previous MKL studies in literature. In particular, for each dataset, we create a set of 17 base kernels, i.e.,

- Gaussian kernels with 14 different widths $(2^{-6}, 2^{-5}, \ldots, 2^{7})$ on all features.
- Polynomial kernels of degree 1 to 3 on all features.

To implement the existing MKL algorithms, we adopt the SimpleMKL toolbox [25] that includes the implementations of both SILP and SD algorithms, the LevelMKL toolbox [34] that implements MKL-Level, and the MKL-Hessian toolbox provided by the author of [6]. For running the above existing MKL algorithms, we adopt their default settings suggested by the two toolboxes, which employ the duality gap as the stopping criterion, and stop the optimization process when the duality gap is less than a threshold (0.01) or the max number (500) of optimization iterations is reached. For Lp-MKL, as no code available,

we implemented it by ourselves, and set $p = 2$ for the best tradeoff of accuracy and efficiency.

For Boost-Exp and Boost-Log [7], we implemented them by ourselves as no code is available. We use linear kernel as base kernel, adopt Matlab built-in function "eig" to compute the generalized eigenvector, and set the number of boosting rounds to 30 as suggested in [7] (the algorithm involves the computation of generalized eigenvector [14] which is very time-consuming, this parameter setting is empirically a good tradeoff between efficacy and efficiency), . For Boost-CG-L1 and Boost-CG-L2 [2], as no code is available too, we implemented them by ourselves, where we adopt MOSEK http://www.mosek.com/ to solve the restricted optimization problems instead of the commercial package ILOG CPLEX 8.0 used by the authors [2]. For Boost-SVM [35], 10-fold cross validation is adopted to select the best kernel, other settings are the same as in our MKBoost algorithms.

For the implementation of our MKBoost algorithms, by default, we set the total number of boosting trials T to 100, the boosting sampling ratio to 0.2, and the sampling decay factor β to 2^{-5} for stochastic MKBoost algorithms. For SVM, we adopt the popular LIBSVM toolbox [5] as the SVM solver. For the multiclass classification tasks, we adopt a one-to-one approach to training a set of binary classifiers. Finally, all the experiments were running in Matlab on a Linux machine with 3GHz Intel CPU and 16GB RAM.

For all the experiments, we repeat each algorithm 20 times on every dataset. Similar to the previous studies, for each run, 50% of the examples were randomly sampled as training data and the remaining were used for test. The training data were normalized to have zero mean and unit variance, and the test data were then normalized using the mean and variance of the training data. The penalty cost parameter C was fixed to 50 for SVM and the compared algorithms. To avoid unstable results for stochastic MKBoost algorithms, we run 10 times under each setting and report average performances. Finally, we report both classification accuracy and time cost for performance evaluation.

4.4 Comparison Results

We first compare the performances between two deterministic MKBoost algorithms (MKBoost-D1 and MKBoost-D2) and the five existing MKL algorithms (MKL-SD, MKL-SILP, MKL-Level, MKL-Hessian and Lp-MKL). Then we compare two deterministic MKBoost algorithms with boosting based algorithms (Boost-Exp, Boost-Log, Boost-CG-L1, Boost-CG-L2, Boost-SVM). Further, we will compare both the accuracy and efficiency performances between the deterministic and stochastic MKBoost algorithms. Finally, we will evaluate the effects of various parameters that may influence the performance of the MKBoost algorithms. To examine statistical significance of the comparisons, for the experimental results reported below, we highlight the best result in each group in bold font by conducting student t-tests with the significance level $\alpha = 0.05$.

4.4.1 MKBoost vs. Regular MKL

TABLE 2 shows the results of comparisons between the two deterministic MKBoost algorithms and the five existing MKL algorithms.

Several observations can be drawn from the results. First of all, in terms of classification accuracy performance, among the four ℓ_1-norm MKL algorithms (MKL-SD, MKL-SILP, MKL-Level, and MKL-Hessian), we found that their accuracy performances are generally comparable, in which no one algorithm significantly performs better than the others. This is consistent to the previous MKL studies as all these MKL algorithms are essentially based on the same formulation, but adopt different optimization techniques to speed up the optimization process. Further, comparing the four regular MKL algorithms and the two proposed MKBoost algorithms, we found the MKBoost algorithms considerably outperform the four regular MKL algorithms in most cases. For example, for dataset D6("glass"), the four regular MKL algorithms achieved the accuracy of no more than 95%, while the proposed deterministic MKBoost algorithms are able to attain a much higher accuracy of over 98% for both MKBoost-D1 and MKBoost-D2 algorithms. Moreover, by examining the performance of the state-of-the-art Lp-MKL algorithm, we found that it often performs better than the four regular ℓ_1-norm MKL algorithms, but still performs considerably worse than our MKBoost algorithms. Finally, by comparing the two deterministic MKBoost algorithms themselves, we found that their classification accuracy performances are generally comparable, in which MKBoost-D1 outperformed MKBoost-D2 in some datasets (D1, D2, D3, D5, D8, D9, D10, D12, D13, D14, D15) but failed to beat MKBoost-D2 in the other datasets.

In addition to the clear gain of classification accuracy, we also found that the MKBoost algorithms enjoy a significant advantage in higher learning efficiency over the regular MKL algorithms. In particular, by examining the time costs of learning the classifiers, we found that the two proposed MKBoost algorithms are often significantly more efficient than the five existing MKL algorithms. Typically, both MKBoost algorithms are several times (about 2$\sim$10 times depending on datasets) faster than the existing MKL algorithms. For example, for the D5("german.number") dataset, the time costs of both MKBoost algorithms are only about one tenth of the costs by MKL-SD and MKL-SILP, and about half, one-sixth and one-third of the costs by MKL-Level, MKL-Hessian and Lp-MKL, respectively. Finally, for the two proposed MKBoost algorithms, their time costs are comparable as they essentially share the same time complexity in theory.

TABLE 2: Comparison between two deterministic MKBoost algorithms and five different MKL algorithms.

Dataset	Metric	MKBoost-D1	MKBoost-D2	MKL-SD	MKL-SILP	MKL-Level	MKL-Hessian	Lp-MKL
D1	accuracy	**0.8122**	0.8093	0.7547	0.7547	0.7544	0.7544	0.7607
	std	±**0.0077**	±0.0086	±0.0009	±0.0009	±0.0000	±0.0000	±0.0066
	time	**4.8513**	4.9239	34.5300	15.6220	30.8875	15.169	19.865
	std	±**0.0750**	±0.0592	±3.5283	±1.9550	±3.6841	±1.6931	±6.3624
D2	accuracy	**0.8137**	0.8122	0.7576	0.7576	0.7575	0.7575	0.7597
	std	±**0.0051**	±0.0048	±0.0003	±0.0003	±0.0000	±0.0000	±0.0040
	time	**21.5289**	21.9411	113.2365	62.0065	101.0525	60.216	85.2905
	std	±**0.4417**	±0.5168	±8.0158	±4.7652	±7.2299	±4.0650	±29.6062
D3	accuracy	**0.8110**	0.8103	0.7515	0.7515	0.7515	0.7515	0.7561
	std	±**0.0052**	±0.0037	±0.0001	±0.0001	±0.0000	±0.0000	±0.0033
	time	**60.6319**	62.5635	258.1065	157.279	238.038	152.84	268.5895
	std	±**2.4952**	±3.6807	±16.9583	±9.4822	±17.7089	±3.7433	±88.5061
D4	accuracy	0.9858	**0.9907**	0.9464	0.946	0.9726	0.9642	0.9809
	std	±0.0079	±**0.0057**	±0.0099	±0.0132	±0.0102	±0.0115	±0.0078
	time	**0.7658**	0.8683	2.952	1.3615	5.175	1.5255	4.1735
	std	±**0.0262**	±0.0240	±0.6294	±0.5663	±0.4705	±0.2693	±0.8449
D5	accuracy	**0.7361**	0.7263	0.7001	0.7003	0.7	0.7	0.7065
	std	±**0.0117**	±0.0094	±0.0004	±0.0025	±0.0000	±0.0000	±0.0055
	time	4.6168	**4.5845**	51.8245	46.996	8.4915	24.1665	12.4300
	std	±0.1372	±**0.1533**	±19.2827	±25.6543	±1.7317	±4.9634	±4.3739
D6	accuracy	0.9885	**0.9890**	0.9462	0.9448	0.9458	0.9491	0.9646
	std	±0.0140	±**0.0121**	±0.0181	±0.0166	±0.0181	±0.0180	±0.0170
	time	0.6058	**0.4497**	1.812	2.4105	3.57	0.9425	1.4880
	std	±0.0964	±**0.0067**	±1.2291	±0.8126	±4.6531	±0.1290	±0.5612
D7	accuracy	0.9426	**0.9453**	0.9306	0.9249	0.93	0.9329	0.9337
	std	±0.0139	±**0.0103**	±0.0155	±0.0255	±0.0183	±0.0177	±0.0200
	time	0.697	**0.6048**	4.376	3.922	4.0235	1.486	2.683
	std	±0.0849	±**0.0170**	±1.3413	±1.5076	±0.8049	±0.5212	±0.8026
D8	accuracy	**0.9522**	0.8829	0.791	0.7932	0.7835	0.8259	0.8556
	std	±**0.0154**	±0.0199	±0.0251	±0.0311	±0.0232	±0.0266	±0.0221
	time	1.163	**1.0960**	8.6205	5.051	6.527	2.9815	5.0955
	std	±0.1033	±**0.0142**	±2.5738	±1.2678	±3.4252	±0.6179	±0.8415
D9	accuracy	**0.9590**	0.9132	0.6628	0.6628	0.6628	0.6628	0.7421
	std	±**0.0143**	±0.0167	±0.0000	±0.0000	±0.0000	±0.0000	±0.0147
	time	**3.1505**	3.2468	73.3835	17.5325	27.1455	18.228	13.125
	std	±**0.0469**	±0.0383	±10.4151	±11.0115	±7.3326	±1.6739	±5.6322
D10	accuracy	**0.9090**	0.9032	0.8646	0.8579	0.8648	0.8321	0.8793
	std	±**0.0070**	±0.0050	±0.0048	±0.0143	±0.0050	±0.0045	±0.0061
	time	**36.0281**	39.8875	361.6830	119.3535	303.527	205.3645	239.864
	std	±**0.4904**	±0.6097	±19.8077	±7.1526	±8.4926	±6.8991	±13.8356
D11	accuracy	0.9663	**0.9675**	0.9096	0.9076	0.9101	0.9065	0.9406
	std	±0.0030	±**0.0026**	±0.0075	±0.0115	±0.0087	±0.0101	±0.0064
	time	**16.7547**	17.5432	171.386	91.966	177.194	62.3781	134.704
	std	±**0.3175**	±0.3060	±14.4990	±6.5391	±15.6756	±4.5846	±6.9403
D12	accuracy	**0.8183**	0.8021	0.7752	0.7791	0.7684	0.7587	0.8024
	std	±**0.0396**	±0.0338	±0.0423	±0.0427	±0.0407	±0.0365	±0.0406
	time	0.5722	**0.4547**	2.135	1.9425	4.4125	0.7485	1.0120
	std	±0.0891	±**0.0045**	±0.9856	±0.6026	±4.3765	±0.2909	±0.2790
D13	accuracy	**0.8158**	0.7923	0.7744	0.7746	0.7733	0.7617	0.7816
	std	±**0.0067**	±0.0079	±0.0038	±0.0040	±0.0036	±0.0000	±0.0057
	time	5.2625	**5.2193**	62.118	25.4405	33.147	52.0675	16.7775
	std	±0.1788	±**0.1737**	±20.8911	±7.4684	±3.8329	±5.0793	±4.2652
D14	accuracy	**0.6493**	0.6329	0.5513	0.553	0.5587	0.554	0.5856
	std	±**0.0268**	±0.0273	±0.0285	±0.0288	±0.0272	±0.0249	±0.0337
	time	**5.2695**	5.4672	12.245	15.4545	12.955	6.291	9.4140
	std	±**0.0314**	±0.0741	±3.1804	±2.1441	±2.3191	±0.1907	±1.0471
D15	accuracy	**0.7877**	0.7848	0.6737	0.6733	0.673	0.6374	0.7371
	std	±**0.0085**	±0.0115	±0.0191	±0.0199	±0.0186	±0.0150	±0.0140
	time	**4.1932**	4.5257	28.208	21.383	25.9065	16.559	20.429
	std	±**0.0654**	±0.0524	±4.6233	±2.0378	±1.9193	±2.7623	±2.6937
D16	accuracy	0.9663	**0.9741**	0.9595	0.9595	0.9597	0.9482	0.9692
	std	±0.0107	±**0.0074**	±0.0094	±0.0094	±0.0097	±0.0091	±0.0079
	time	0.8688	**0.8285**	2.514	1.576	4.9735	1.3365	5.7715
	std	±0.0663	±**0.0361**	±0.5658	±0.7922	±1.0263	±0.3829	±1.4157

The above encouraging results show that the two MKBoost algorithms are not only more accurate than the regular MKL algorithms, but also are able to learn the models considerably more efficiently.

4.4.2 MKBoost vs. Other Boosting Algorithms

TABLE 3 shows the results of comparisons between the two deterministic MKBoost algorithms and the existing boosting based algorithms. Several observations can be drawn from the results.

First of all, in terms of classification accuracy, among all the algorithms, the proposed two MKBoost algorithms achieve the best for most cases, and Boost-SVM tends to perform better than the other existing algorithms. In particular, comparing with Boost-Exp and Boost-Log, MKBoost-D1 and MKBoost-D2 obtain significantly better results on all the datasets; further, comparing with Boost-CG-L1 and Boost-CG-L2, MKBoost-D1 and MKBoost-D2 achieve much better results on most datasets, except a few datasets (e.g., D6, D9, D13) where Boost-CG-L2 is comparable; Boost-SVM usually performs better than the other 4 boosting based algorithms, but worse than MKBoost-D1 and MKBoost-D2 for most datasets.

Second, in terms of time efficiency, the proposed MKBoost algorithms are clearly more efficient than the former four boosting based algorithms, and Boost-CG-L2 is the most inefficient algorithm among those four compared algorithms. Specifically, the time costs by Boost-Exp and Boost-Log are often a number of times higher than those by the proposed MKBoost algorithms. For example, on several datasets (D1, D2, D3 and D10), the time costs by Boost-Exp and Boost-Log are almost 100 times of MKBoost-D1 and MKBoost-D2. By examining the other two boosting algorithms, Boost-CG-L1 is in general comparable to Boost-Exp and Boost-Log, while Boost-CG-L2 is the most computationally intensive one. For example, for some large dataset such as D3, the proposed MKBoost-D1 and MKBoost-D2 algorithms took about 1 minute, in contrast, Boost-CG-L2 took almost 9 hours. When compared with Boost-SVM, MKBoost tends to be more efficient on larger datasets (which take more running time), but less efficient on those smaller scale datasets (D4, D6, D7, D8, D12 and D16). This is because SVM is solved by LIBSVM, whose empirical time complexity is about $O(n^{1.4}) \sim O(n^{2.3})$, cross validation in Boost-SVM is time-consuming on larger datasets. Thus, compared with Boost-SVM, MKBoost is more scalable for large dataset.

In a nutshell, the above observations clearly indicate the proposed MKBoost algorithms are empirically more effective, efficient and scalable than the existing boosting based algorithms.

4.4.3 Deterministic vs. Stochastic MKBoost

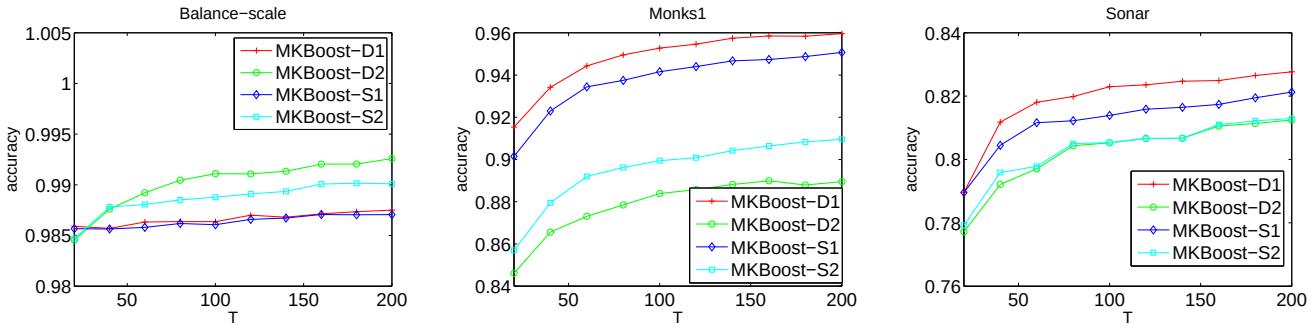
TABLE 4 shows the results of comparisons between the deterministic MKBoost algorithms (MKBoost-D1

TABLE 4: Comparison between the deterministic and stochastic MKBoost algorithms. Here "DS" denotes datasets, "MKB" denotes MKBoost, and "acc." denotes accuracy.

DS	Metric	MKB-D1	MKB-D2	MKB-S1	MKB-S2
D1	acc.	0.8122	0.8093	0.8126	**0.8134**
	std	±0.0077	±0.0086	±0.0072	±**0.0069**
	time	4.8513	4.9239	**2.2384**	2.2776
	std	±0.0750	±0.0592	±**0.0396**	±0.0384
D2	acc.	0.8137	0.8122	0.8132	**0.8164**
	std	±0.0051	±0.0048	±0.0052	±**0.0041**
	time	21.5289	21.9411	9.4543	**9.4108**
	std	±0.4417	±0.5168	±0.3263	±**0.3138**
D3	acc.	0.811	0.8103	0.8118	**0.8156**
	std	±0.0052	±0.0037	±0.0055	±**0.0046**
	time	60.6319	62.5635	26.0512	**25.8940**
	std	±2.4952	±3.6807	±1.4618	±**1.4806**
D4	acc.	0.9858	**0.9907**	0.9878	0.9886
	std	±0.0079	±**0.0057**	±0.0067	±0.0061
	time	0.7658	0.8683	0.435	**0.3975**
	std	±0.0262	±0.0240	±0.0119	±**0.0192**
D5	acc.	**0.7361**	0.7263	0.7355	0.7359
	std	±**0.0117**	±0.0094	±0.0106	±0.0079
	time	4.6168	4.5845	2.0623	**2.0320**
	std	±0.1372	±0.1533	±0.0746	±**0.0563**
D6	acc.	0.9885	**0.9890**	0.989	0.9879
	std	±0.0140	±**0.0121**	±0.0138	±0.0137
	time	0.6058	0.4497	0.2732	**0.2727**
	std	±0.0964	±0.0067	±0.0088	±**0.0078**
D7	acc.	0.9426	0.9453	**0.9469**	0.943
	std	±0.0139	±0.0103	±**0.0130**	±0.0152
	time	0.697	0.6048	0.3547	**0.3527**
	std	±0.0849	±0.0170	±0.0085	±**0.0106**
D8	acc.	**0.9522**	0.8829	0.943	0.8998
	std	±**0.0154**	±0.0199	±0.0153	±0.0221
	time	1.163	1.096	**0.5347**	0.5423
	std	±0.1033	±0.0142	±**0.0122**	±0.0132
D9	acc.	**0.9590**	0.9132	0.9519	0.9308
	std	±**0.0143**	±0.0167	±0.0156	±0.0189
	time	3.1505	3.2468	**1.6099**	1.6198
	std	±0.0469	±0.0383	±**0.0233**	±0.0183
D10	acc.	**0.9090**	0.9032	0.9085	0.9070
	std	±**0.0070**	±0.0050	±0.0070	±0.0056
	time	36.0281	39.8875	20.6196	**20.4699**
	std	±0.4904	±0.6097	±0.3701	±**0.3793**
D11	acc.	0.9663	0.9675	0.9665	**0.9677**
	std	±0.0030	±0.0026	±0.0034	±**0.0028**
	time	16.7547	17.5432	9.366	**9.2138**
	std	±0.3175	±0.3060	±0.1516	±**0.1477**
D12	acc.	0.8183	0.8021	**0.8186**	0.8076
	std	±0.0396	±0.0338	±**0.0356**	±0.0338
	time	0.5722	0.4547	0.2388	**0.2370**
	std	±0.0891	±0.0045	±0.0051	±**0.0039**
D13	acc.	**0.8158**	0.7923	0.8132	0.8065
	std	±**0.0067**	±0.0079	±0.0066	±0.0080
	time	5.2625	5.2193	2.4548	**2.4192**
	std	±0.1788	±0.1737	±0.0892	±**0.0663**
D14	acc.	**0.6493**	0.6329	0.6416	0.6335
	std	±**0.0268**	±0.0273	±0.0270	±0.0274
	time	5.2695	5.4672	**2.8047**	2.8174
	std	±0.0314	±0.0741	±**0.0382**	±0.0257
D15	acc.	**0.7877**	0.7848	0.786	0.7853
	std	±**0.0085**	±0.0115	±0.0114	±0.0110
	time	4.1932	4.5257	2.6175	**2.6172**
	std	±0.0654	±0.0524	±0.0357	±**0.0333**
D16	acc.	0.9663	**0.9741**	0.9673	0.9701
	std	±0.0107	±**0.0074**	±0.0102	±0.0087
	time	0.8688	0.8285	0.554	**0.5508**
	std	±0.0663	±0.0361	±0.0251	±**0.0242**

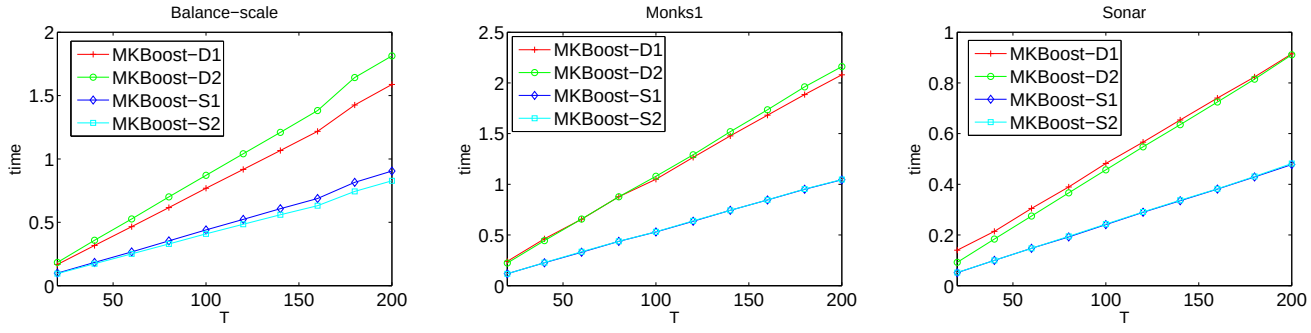
TABLE 3: Comparison between two deterministic MKBoost algorithms and Boosting based algorithms.

Dataset	Metric	MKBoost-D1	MKBoost-D2	Boost-Exp	Boost-Log	Boost-CG-L1	Boost-CG-L2	Boost-SVM
D1	accuracy	**0.8122**	0.8093	0.7855	0.7830	0.7095	0.7942	0.8008
	std	**±0.0077**	±0.0086	±0.0052	±0.0055	±0.0043	±0.0054	±0.0085
	time	**4.8513**	4.9239	265.6700	276.3300	361.04	1271.32	7.7455
	std	**±0.0750**	±0.0592	±15.7931	±18.4387	±26.2137	±134.2567	±0.5961
D2	accuracy	**0.8137**	0.8122	0.7594	0.7368	0.7682	0.7922	0.8031
	std	**±0.0051**	±0.0048	±0.0049	±0.0048	±0.0050	±0.0054	±0.0100
	time	**21.5289**	21.9411	2487.6000	2152.6600	815.91	6734.44	32.2918
	std	**±0.4417**	±0.5168	±56.7813	±63.4569	±61.2565	±531.2546	±3.3735
D3	accuracy	**0.8110**	0.8103	0.7251	0.769	0.7931	0.7912	0.8065
	std	**±0.0052**	±0.0037	±0.0037	±0.0041	±0.0047	±0.0043	±0.0107
	time	**60.6319**	62.5635	7830.15	7968.52	1801.96	32456.2655	71.8097
	std	**±2.4952**	±3.6807	±153.1465	±187.3217	±126.1425	±3013.2650	±5.0836
D4	accuracy	0.9858	**0.9907**	0.941	0.941	0.8472	0.9811	0.9817
	std	±0.0079	**±0.0057**	±0.0097	±0.0135	±0.0096	±0.0157	±0.0099
	time	0.7658	0.8683	7.38	7.85	1.73	256.5200	**0.6973**
	std	±0.0262	±0.0240	±1.2564	±1.3466	±0.2977	±34.6522	**±0.0253**
D5	accuracy	**0.7361**	0.7263	0.672	0.648	0.692	0.7180	0.7141
	std	**±0.0117**	±0.0094	±0.0050	±0.0050	±0.0051	±0.0053	±0.0149
	time	4.6168	**4.5845**	48.91	47.21	53.33	467.0900	5.4105
	std	±0.1372	**±0.1533**	±5.1546	±6.1354	±6.1365	±45.3166	±0.3827
D6	accuracy	0.9885	**0.9890**	0.9657	0.9611	0.9623	0.9806	0.9729
	std	±0.0140	**±0.0121**	±0.0168	±0.0146	±0.0140	±0.0121	±0.0237
	time	0.6058	0.4497	1.51	1.51	0.92	120.4900	**0.1725**
	std	±0.0964	±0.0067	±0.3456	±0.3689	±0.1366	±21.2364	**±0.0035**
D7	accuracy	0.9426	**0.9453**	0.859	0.8419	0.8914	0.9371	0.9365
	std	±0.0139	**±0.0103**	±0.0136	±0.0157	±0.0204	±0.0200	±0.0185
	time	0.697	0.6048	1.87	1.89	1.96	154.62	**0.3699**
	std	±0.0849	±0.0170	±0.6431	±0.7613	±0.7865	±30.0132	**±0.0033**
D8	accuracy	**0.9522**	0.8829	0.6703	0.6835	0.5899	0.9101	0.9251
	std	**±0.0154**	±0.0199	±0.0244	±0.0244	±0.0214	±0.0135	±0.0277
	time	1.163	1.096	6.63	6.8	8.95	204.67	**0.7738**
	std	±0.1033	±0.0142	±0.9736	±0.9513	±2.6354	±25.3643	**±0.0202**
D9	accuracy	**0.9590**	0.9132	0.6628	0.6628	0.6705	0.9496	0.9469
	std	**±0.0143**	±0.0167	±0.0113	±0.0138	±0.0127	±0.0143	±0.0220
	time	**3.1505**	3.2468	59.5300	61.5500	47.91	399.3	3.5175
	std	**±0.0469**	±0.0383	±9.1466	±12.3685	±8.1666	±15.3147	±0.0484
D10	accuracy	**0.9090**	0.9032	0.8597	0.8592	0.7333	0.8798	0.901
	std	**±0.0070**	±0.0050	±0.0057	±0.0051	±0.0052	±0.0054	±0.0056
	time	**36.0281**	39.8875	3551.8400	3607.8900	440.61	16901.86	87.1784
	std	**±0.4904**	±0.6097	±156.4560	±179.1655	±13.5655	±435.2456	±4.0017
D11	accuracy	0.9663	**0.9675**	0.9255	0.9152	0.5905	0.9455	0.9597
	std	±0.0030	**±0.0026**	±0.0097	±0.0091	±0.0097	±0.0026	±0.0065
	time	**16.7547**	17.5432	235.66	226.17	58.46	7299.5	17.7602
	std	**±0.3175**	±0.3060	±12.3646	±11.9764	±4.1655	±268.1564	±0.2537
D12	accuracy	**0.8183**	0.8021	0.7443	0.7799	0.7961	0.7961	0.8049
	std	**±0.0396**	±0.0338	±0.0313	±0.0325	±0.0403	±0.0343	±0.0363
	time	0.5722	0.4547	1.56	1.56	2.2	38.15	**0.1887**
	std	±0.0891	±0.0045	±0.3146	±0.3568	±1.0012	±7.3247	**±0.0009**
D13	accuracy	**0.8158**	0.7923	0.7681	0.7375	0.5539	0.8148	0.7844
	std	**±0.0067**	±0.0079	±0.0047	±0.0043	±0.0032	±0.0060	±0.0141
	time	5.2625	**5.2193**	103.86	110.96	159.9	574.4	5.6725
	std	±0.1788	**±0.1737**	±10.3654	±12.1467	±16.6554	±49.8135	±0.1725
D14	accuracy	**0.6493**	0.6329	0.6121	0.6174	0.2819	0.5772	0.6266
	std	**±0.0268**	±0.0273	±0.0370	±0.0356	±0.0179	±0.0219	±0.0318
	time	**5.2695**	5.4672	11.29	11.22	16.27	1209.91	5.983
	std	**±0.0314**	±0.0741	±1.4366	±1.2547	±2.3665	±156.7213	±0.0287
D15	accuracy	**0.7877**	0.7848	0.7214	0.7275	0.6256	0.7299	0.7779
	std	**±0.0085**	±0.0115	±0.0144	±0.0144	±0.0152	±0.0047	±0.0208
	time	**4.1932**	4.5257	20.46	20.62	27.01	905.01	4.8038
	std	**±0.0654**	±0.0524	±2.6888	±2.7832	±4.3654	±101.3590	±0.0412
D16	accuracy	0.9663	**0.9741**	0.8838	0.9331	0.9542	0.9554	0.9611
	std	±0.0107	**±0.0074**	±0.0097	±0.0095	±0.0077	±0.0067	±0.0095
	time	0.8688	0.8285	8.07	7.63	3.57	146.77	**0.7467**
	std	±0.0663	±0.0361	±1.9875	±1.7825	±0.8765	±21.5791	**±0.0248**



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 1: Evaluation of MKBoost classification accuracy with respect to boosting parameter T.



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 2: Evaluation of MKBoost learning time cost with respect to boosting parameter T.

and MKBoost-D2) and the stochastic MKBoost algorithms (MKBoost-S1 and MKBoost-S2). Several observations can be drawn from the results.

First of all, it is clear to see that the stochastic MKBoost algorithms are more efficient than the deterministic MKBoost algorithms. The exact value of time efficiency speedup achieved by the stochastic algorithms depends on different datasets. Overall, the time cost of the two stochastic MKBoost algorithms is about half of the cost taken by the two deterministic algorithms as observed from this set of experiments.

In addition to the time efficiency issue, another concern is how accurate can the stochastic MKBoost algorithms perform in comparison to the deterministic algorithms. By further examining the classification accuracy results in details, it is a bit surprising to find that the two stochastic algorithms perform very well, which are in general fairly comparable to the deterministic algorithms. The difference of the accuracy values between a deterministic algorithm and a stochastic algorithm is often no more than 1%. Besides, for a few cases, we even observed that a stochastic algorithm slightly outperformed a deterministic algorithm. Finally, by comparing the two different stochastic algorithms, we found that both of their accuracy and efficiency performances are quite comparable, which is similar to the previous comparison of the two deterministic algorithms.

The above observations show that the proposed stochastic MKBoost algorithms can effectively speed up the deterministic algorithms by maintaining comparable classification accuracy.

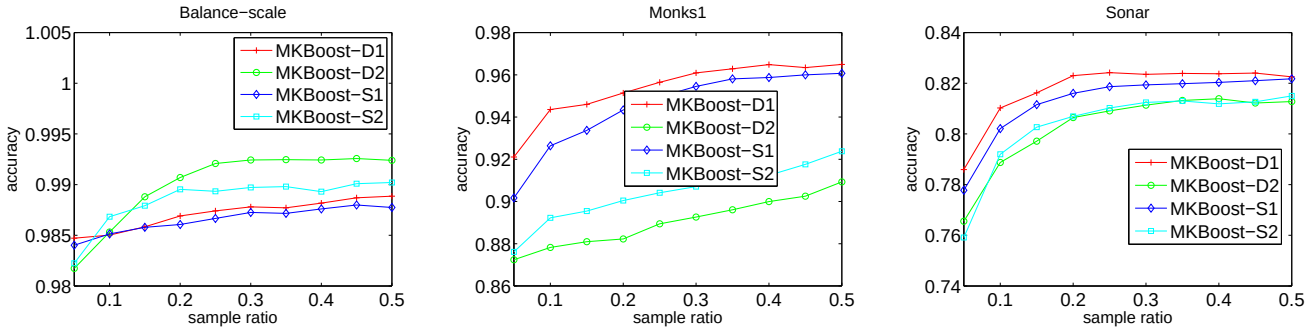
4.5 Parameter Evaluation

We notice that there are a few parameters that could affect the performance (both accuracy and efficiency) of the proposed MKBoost algorithms. These parameters include the total number of boosting trials T, the sampling ratio of training data at each boosting trial, and the sampling decay factor β in the stochastic MKBoost algorithms. To examine the influence of these parameters, we conduct a set of experiments to evaluate their impacts on both accuracy and efficiency performance in the classification tasks.

4.5.1 Evaluation of Boosting Parameter T

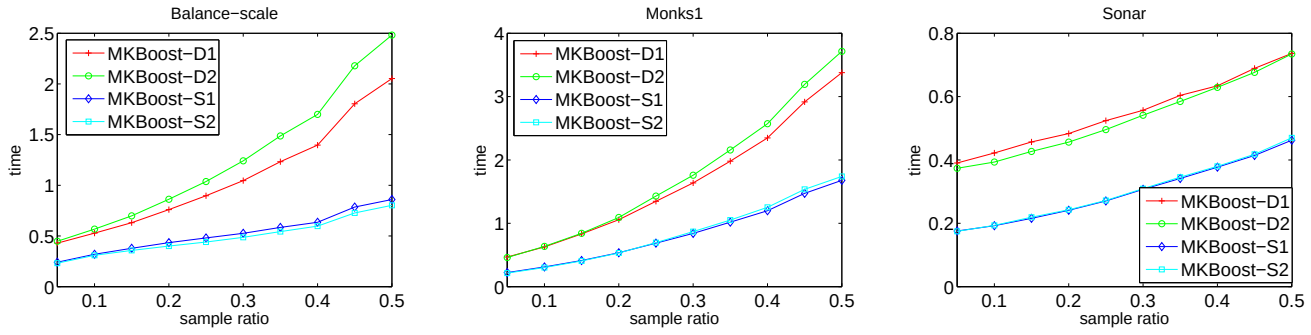
The first set of experiments is to examine the influence of the total number of boosting trials T for the MKBoost algorithms. In our previous experiments, we simply fix T to 100. In this set of experiments, we examine the experimental results by varying the parameter T from 20 to 200. Fig. 1 and Fig. 2 show the evaluation results for the impact of the boosting parameter T on the classification accuracy and learning time cost, respectively. Some observations can be drawn from the evaluation results.

First of all, we observed that, in terms of classification accuracy performance, increasing the total number of boosting trials T in general is able to



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 3: Evaluation of MKBoost classification accuracy with respect to boosting sampling ratio.



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 4: Evaluation of MKBoost learning time cost with respect to boosting sampling ratio.

boost the accuracy performance of all the proposed MKBoost algorithms consistently. Such observation is particularly more evident when the total number of boosting trials T is small (e.g. $T < 50$) at the beginning. The improvements of classification accuracy performance usually become very small when the parameter T is large (e.g. $T > 200$).

On the other hand, we found that increasing the T value leads to a linear increase of the time cost for training the models by all the proposed MKBoost algorithms. This is not surprising since all the MKBoost algorithms have a linear time complexity with respect to the total number of boosting trials T.

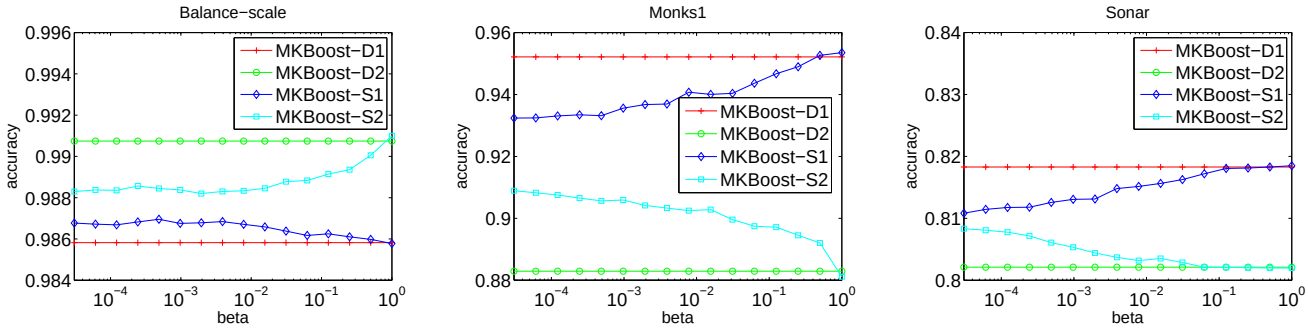
The above empirical observations indicate that choosing an appropriate boosting parameter T is essentially a tradeoff between classification accuracy and efficiency performances. In practice, it is not difficult to choose an appropriate T value that usually falls in between 50 and 200, which sometimes also depends on the empirical requirements of efficiency and accuracy in a real-life application.

4.5.2 *Evaluation of Boosting Sampling Ratio*

Another boosting parameter that may affect the MKBoost algorithms is the boosting sampling ratio, which determines the proportion of training data examples sampled from the whole collection of training data at each boosting trial. Fig. 3 and Fig. 4 respectively show the evaluations of accuracy and efficiency performance with respect to the boosting sampling ratio by varying its value from 0.05 to 0.5.

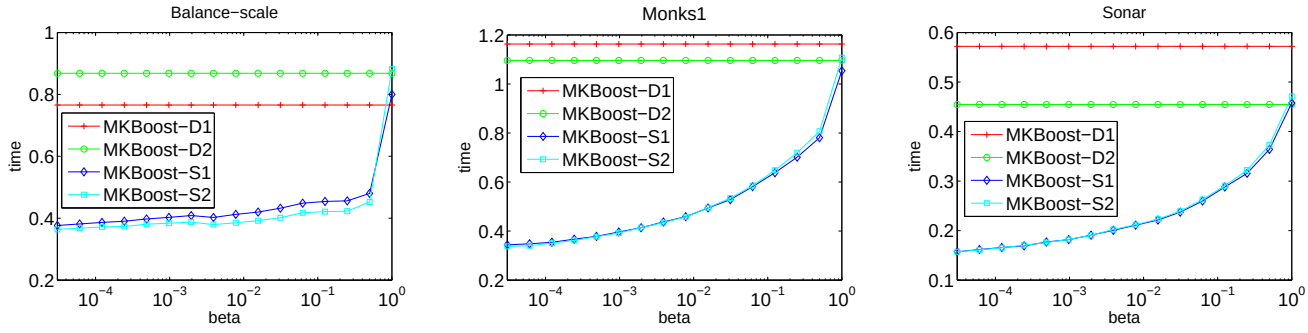
From the experimental results, we found that the MKBoost algorithms with a large boosting sampling ratio value usually produced better classification accuracy performance. This is especially more evident when the boosting sampling ratio is small. For example, on the dataset "Monks1", the MKBoost-S1 algorithm has the accuracy of less than 93%, which was boosted to above 95% when the sampling ratio is increased to 0.3. Despite the improvement when increasing the boosting sampling ratio, we found that the classification accuracy tends to saturate when the value is large enough (e.g. larger than 0.4). All these observations are not difficult to interpret because when the sampling ratio is too small, the base kernel classifiers trained at the boosting process may suffer from insufficient training examples. On the other hand, employing a too large sampling ratio may lead to sample too many training data examples for some large dataset, which may be somewhat redundant for building the base classifiers at the boosting trials.

In addition to the impact on the accuracy, the boosting sampling ratio parameter also affects the efficiency of the MKBoost algorithms. Similar to the observation from the evaluation of parameter T, increasing the boosting sampling ratio also leads to the increase of the time cost needed by the MKBoost algorithms. In particular, the relationship between the sampling ratio and the learning time cost of the MKBoost algorithms fully depends on the underlying algorithms used in training the kernel classifiers. As we adopt SVM for training the base kernel classifiers, the time cost of



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 5: Evaluation of accuracy of stochastic MKBoost algorithms with respect to the sampling decay factor β.



(a) "Balance-scale" (b) "Monks1" (c) "Sonar"

Fig. 6: Evaluation of time cost of stochastic MKBoost algorithms with respect to the sampling decay factor β.

an MKBoost algorithm thus depends on the time cost of the SVM algorithm (e.g., the empirical time complexity of the SMO algorithm implemented in the LIBSVM package [5] is about $O(n^{1.4})$ to $O(n^{2.3})$).

4.5.3 *Evaluation of the Sampling Decay Factor β for Stochastic MKBoost Algorithms*

The last set of experiments is to examine the effect of the sampling decay factor β for the two stochastic MKBoost algorithms (MKBoost-S1 and MKBoost-S2).

Fig. 5 and Fig. 6 show the impact of sampling decay factor on both accuracy and time cost, respectively. From the results, we found that when β is set to 1 in the extreme case, the stochastic algorithms converge to the deterministic algorithms, respectively. This is not surprising as $\beta = 1$ indicates no decay at all; as a result the stochastic algorithm reduces to the deterministic one. Further, we observed that when decreasing the value of β, the time cost can be consistently decreased since only some kernels will be sampled for training due to the decay effect.

However, there is not a consistent trend for the impact on the accuracy by varying the β value. But we empirically found that the performance of MKBoost-D1 is in general better than MKBoost-D2 for those datasets where there is a single best kernel classifier that significantly dominates among all the kernels (e.g., Monks1, Sonar); for such datasets, MKBoost-S1 usually favors larger β, while MKBoost-S2 prefers smaller β. However, for those datasets where no such a single best kernel classifier exists, the situation is reversed. In a real-world application, it is an open question to determine which algorithm is more preferred. To address this challenge, we could try to check if there exists a single best kernel classifier for the application; or if such prior knowledge is not available, we could adopt cross validation to address the algorithm selection issue.

5 Conclusions

This paper presented a framework of Multiple Kernel Boosting (MKBoost) for classification, which applies boosting techniques to learn classification models with multiple kernels. As a trade-off between accuracy and efficiency, we first proposed two deterministic MKBoost algorithms using all kernels at each boosting trial; we then presented two stochastic MKBoost algorithms that randomly sample a subset of kernels at each boosting trial. We found MKBoost empirically achieved better accuracy than the regular MKL methods, and performed much more efficiently than the state-of-the-art MKL methods. Besides, we also found the stochastic MKBoost algorithms considerably improved the efficiency of the deterministic algorithms by maintaining comparable accuracy. An open problem of future work is to theoretically analyze the generalization performance of MKBoost.

Acknowledgements

This work was supported in part by Singapore MOE tier-1 grant (M4010978), MOE ARC tier-2 grant (T208B2203), and Microsoft grant (PO90161723).

REFERENCES

[1] Kristin P. Bennett, Michinari Momma, and Mark J. Embrechts. Mark: A boosting algorithm for heterogeneous kernel models. In *KDD*, pages 24–31, 2002.

[2] J. Bi, T. Zhang, and K.P. Bennett. Column-generation boosting methods for mixture of kernels. In *KDD*, page 526, 2004.

[3] Jinbo Bi, Glenn Fung, Murat Dundar, and R. Bharat Rao. Semi-supervised mixture of kernels via lpboost methods. In *ICDM*, pages 569–572, 2005.

[4] Olivier Bousquet and Daniel J. L. Herrmann. On the complexity of learning the kernel matrix. In *NIPS*, pages 399–406, 2002.

[5] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[6] Olivier Chapelle and Alain Rakotomamonjy. Second order optimization of kernel parameters. *NIPS Workshop*, 2008.

[7] K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In *NIPS*, pages 537–544, 2002.

[8] Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz S. Kandola. On kernel-target alignment. In *NIPS*, pages 367–373, 2001.

[9] Santanu Das, Bryan L. Matthews, Ashok N. Srivastava, and Nikunj C. Oza. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In *KDD'10*, pages 47–56, Washington, DC, USA, 2010.

[10] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

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

[12] Steven C. H. Hoi, Rong Jin, and Michael R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *ICML*, pages 361–368, 2007.

[13] Steven C. H. Hoi, Michael R. Lyu, and Edward Y. Chang. Learning the unified kernel machines for classification. In *KDD*, pages 187–196, 2006.

[14] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, Cambridge, New York, 1985.

[15] Shuiwang Ji, Liang Sun, Rong Jin, and Jieping Ye. Multi-label multiple kernel learning. In *NIPS*, 2008.

[16] Rong Jin, Steven C. H. Hoi, and Tianbao Yang. Online multiple kernel learning: Algorithms and mistake bounds. In *ALT*, pages 390–404, 2010.

[17] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, pages 321–328, 2003.

[18] Marius Kloft, Ulf Brefeld, Soeren Sonnenburg, Pavel Laskov, Klaus-Robert Müller, and Alexander Zien. Efficient and accurate ℓ_p-norm multiple kernel learning. In *NIPS*, pages 997–1005, 2009.

[19] Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, pages 315–322, 2002.

[20] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *JMLR*, 5:27–72, 2004.

[21] Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Eng. Appl. of AI*, 21(5):785–795, 2008.

[22] Chris Longworth and Mark J. F. Gales. Combining derivative and parametric kernels for speaker verification. *IEEE Trans. on Audio, Speech & Language Processing*, 17(4):748–757, 2009.

[23] Makoto Miwa, Rune Sætre, Yusuke Miyao, and Jun ichi Tsujii. Protein-protein interaction extraction by leveraging multiple kernels and parsers. *I. J. Medical Informatics*, 78(12):39–46, 2009.

[24] Dmitry Pavlov, Jianchang Mao, and Byron Dom. Scaling-up support vector machines using boosting algorithm. In *ICPR*, pages 2219–2222, 2000.

[25] Alain Rakotomamonjy, Francis R. Bach, Stephane Canu, and Yves Grandvalet. Simplemkl. *JMLR*, 11:2491–2521, 2008.

[26] B. Schölkopf and Alex Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

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

[28] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *JMLR*, 7:1531–1565, 2006.

[29] E Sung, Lei Wang, and Eric Sung. A study of adaboost with svm based weak learners. *Proceedings 2005 IEEE International Joint Conference on Neural Networks 2005*, 1:196–201, 2005.

[30] Lei Tang, Jianhui Chen, and Jieping Ye. On multiple kernel learning with multiple labels. In *IJCAI*, pages 1255–1260, 2009.

[31] Seyyed Majid Valiollahzadeh, Abolghasem Sayadiyan, and Mohammad Nazari. Face detection using adaboosted svm-based component classifier. *CoRR*, abs/0812.2575, 2008.

[32] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[33] Hao Xia and Steven C. H. Hoi. Mkboost: A framework of multiple kernel boosting. In *SDM*, pages 199–210, 2011.

[34] Zenglin Xu, Rong Jin, Irwin King, and Michael R. Lyu. An extended level method for efficient multiple kernel learning. In *NIPS*, 2008.

[35] Xiaolong Zhang and Fang Ren. Improving svm learning accuracy with adaboost. *International Conference on Natural Computation*, 3:221–225, 2008.

[36] Xiaojin Zhu, Jaz S. Kandola, Zoubin Ghahramani, and John D. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *NIPS*, 2004.

[37] Jinfeng Zhuang, Ivor W. Tsang, and Steven C. H. Hoi. A family of simple non-parametric kernel learning algorithms. *Journal of Machine Learning Research*, 12:1313–1347, 2011.

[38] Jinfeng Zhuang, Ivor W. Tsang, and Steven C. H. Hoi. Two-layer multiple kernel learning. *Journal of Machine Learning Research - Proceedings Track*, 15:909–917, 2011.

[39] Jinfeng Zhuang, Jialei Wang, Steven C. H. Hoi, and Xiangyang Lan. Unsupervised multiple kernel learning. *Journal of Machine Learning Research - Proceedings Track*, 20:129–144, 2011.

[40] Alexander Zien and Cheng Soon Ong. Multiclass multiple kernel learning. In *ICML*, pages 1191–1198, 2007.

Hao Xia is currently a PhD candidate in the School of Computer Engineering at the Nanyang Technological University, Singapore. He received his bachelor degree from Tsinghua University, Beijing, P.R. China, in 2008. His research interests are statistical machine learning, data mining, and multimedia information retrieval.

Steven C. H. Hoi is an Assistant Professor of the School of Computer Engineering at Nanyang Technological University, Singapore. He received his Bachelor degree from Tsinghua University, P.R. China, in 2002, and his Ph.D degree in computer science and engineering from The Chinese University of Hong Kong, in 2006. His research interests are machine learning and data mining and their applications to multimedia information retrieval (image and video retrieval), social media and web mining, and computational finance, etc. He has published over 100 referred papers in top conferences and journals in related areas. He has served as general co-chair for ACM SIGMM Workshops on Social Media (WSM'09, WSM'10, WSM'11), program co-chair for the fourth Asian Conference on Machine Learning (ACML'12), book editor for "Social Media Modeling and Computing", guest editor for ACM Transactions on Intelligent Systems and Technology (ACM TIST), technical PC member for many international conferences, and external reviewer for many top journals and worldwide funding agencies, including NSF in US and RGC in Hong Kong. He is a member of IEEE and ACM.