Journal of Machine Learning Research 17 (2016) 1-43 Submitted 4/14; Revised 4/15; Published 4/16

# Large Scale Online Kernel Learning

**Jing Lu** JING.LU.2014@PHDIS.SMU.EDU.SG
**Steven C.H. Hoi** * CHHOI@SMU.EDU.SG
*School of Information Systems, Singapore Management University*
*80 Stamford Road, Singapore, 178902*

**Jialei Wang** JIALEI@UCHICAGO.EDU
*Department of Computer Science, University of Chicago*
*5050 S Lake Shore Drive Apt S2009 Chicago IL, USA, 60637*

**Peilin Zhao** ZHAOP@I2R.A-STAR.EDU.SG
*Institute for Infocomm Research, A*STAR*
*1 Fusionopolis Way, 21-01 Connexis, Singapore, 138632*

**Zhi-Yong Liu** ZHIYONG.LIU@IA.AC.CN
*State Key Lab of Management and Control for Complex System, Chinese Academy of Sciences*
*No. 95 Zhongguancun East Road, Haidian District, Beijing, China, 100190*

**Editor:** John Shawe-Taylor

## Abstract

In this paper, we present a new framework for large scale online kernel learning, making kernel methods efficient and scalable for large-scale online learning applications. Unlike the regular budget online kernel learning scheme that usually uses some budget maintenance strategies to bound the number of support vectors, our framework explores a completely different approach of kernel functional approximation techniques to make the subsequent online learning task efficient and scalable. Specifically, we present two different online kernel machine learning algorithms: (i) Fourier Online Gradient Descent (FOGD) algorithm that applies the random Fourier features for approximating kernel functions; and (ii) Nyström Online Gradient Descent (NOGD) algorithm that applies the Nyström method to approximate large kernel matrices. We explore these two approaches to tackle three online learning tasks: binary classification, multi-class classification, and regression. The encouraging results of our experiments on large-scale datasets validate the effectiveness and efficiency of the proposed algorithms, making them potentially more practical than the family of existing budget online kernel learning approaches.

**Keywords:** online learning, kernel approximation, large scale machine learning

## 1. Introduction

In machine learning, *online learning* represents a family of efficient and scalable learning algorithms for building a predictive model incrementally from a sequence of data examples (Rosenblatt, 1958). Unlike regular batch machine learning methods (Shawe-Taylor and Cristianini, 2004; Vapnik, 1995) which usually suffer from a high re-training cost whenever new training data arrive, online learning algorithms are often very efficient and highly

∗. Corresponding Author

scalable, making them more suitable for large-scale online applications where data usually arrive sequentially and evolve dynamically and rapidly. Online learning techniques can be applied to many real-world applications, such as online spam detection (Ma et al., 2009), online advertising, multimedia retrieval (Xia et al., 2013), and computational finance (Li et al., 2012). In this paper, we first present an online learning methodology to tackle *online binary classification* tasks and then extend the technique to solve the tasks of *online multi-class classification* and *online regression* in the following sections.

Recently, a wide variety of online learning algorithms have been proposed to tackle online classification tasks. One popular family of online learning algorithms, which are referred to as the "linear online learning" (Rosenblatt, 1958; Crammer et al., 2006; Dredze et al., 2008), learn a linear predictive model on the input feature space. The key limitation of these algorithms lies in that the linear model sometimes is restricted to make effective classification if training data are linearly separable in the input feature space, which is not a common scenario for many real-world classification tasks especially when dealing with noisy training data in relatively low dimensional space. This has motivated the studies of "kernel based online learning" or referred to as "online kernel learning" (Kivinen et al., 2001; Freund and Schapire, 1999), which aims to learn kernel-based predictive models for resolving the challenging tasks of classifying instances that are non-separable in the input space.

One key challenge of conventional online kernel learning methods is that an online learner usually has to maintain a set of support vectors (SV's) in memory for representing the kernel-based predictive model. During the online learning process, whenever a new incoming training instance is misclassified, it typically will be added to the SV set, making the size of support vector set unbounded and potentially causing memory overflow for a large-scale online learning task. To address this challenge, a promising research direction is to explore "budget online kernel learning" (Crammer et al., 2003), which attempts to bound the number of SV's with a fixed budget size using different budget maintenance strategies whenever the budget overflows. Despite being studied actively, the existing budget online kernel methods have some limitations. Some efficient algorithms are too simple to achieve satisfactory approximation accuracy; some other algorithms are, despite more effective, too computationally intensive to run for large datasets, making them harm the crucial merit of high efficiency of online learning techniques for large-scale applications. In addition, when dealing with extremely large-scale databases in distributed machine learning environments (Low et al., 2012; Dean and Ghemawat, 2008), the growing large size of SV's would be a significant overhead for communication between different nodes. It is thus very important to investigate effective budget online kernel learning techniques to reduce the size of SV's so as to minimize the overall communication cost.

Unlike the existing budget online kernel learning methods, in this paper, we present a novel framework of large scale online kernel learning by exploring a completely different strategy. In particular, the key idea of our framework is to explore functional approximation techniques to approximate a kernel by transforming data from the input space to a new feature space, and then apply existing linear online learning algorithms on the new feature space. This allows to inherit the power of kernel learning while being able to take advantages of existing efficient linear online learning algorithms for large-scale online learning tasks. Specifically, we propose two different new algorithms: (i) Fourier Online Gradient

Descent (FOGD) algorithm which adopts the random Fourier features for approximating shift-invariant kernels and learns the subsequent model by online gradient descent; and (ii) Nyström Online Gradient Descent (NOGD) algorithm which employs the Nyström method for large kernel matrix approximation followed by online gradient descent learning. We explore the applications of the proposed algorithms for three different online learning tasks: binary classification, multi-class classification, and regression. We give theoretical analysis of our proposed algorithms, and conduct an extensive set of empirical studies to examine their efficacy.

The rest of the paper is organized as follows. Section 2 reviews the background and related work. Section 3 proposes the FOGD and NOGD algorithms for binary classification task and Section 4 analyze their theoretical properties. Section 5 and Section 6 further extends the two techniques for tackling online multi-class classification and online regression tasks, respectively. Section 7 presents our experimental results for three different tasks and Section 8 concludes our work.

## 2. Related Work

Our work is related to two major categories of machine learning research work: *online learning* and *kernel methods*. Below we briefly review some representative related work in each category.

First of all, our work is closely related to online learning methods for classification (Rosenblatt, 1958; Freund and Schapire, 1999; Crammer et al., 2006; Zhao and Hoi, 2010; Zhao et al., 2011; Wang et al., 2012a; Hoi et al., 2013), particularly for budget online kernel learning where various algorithms have been proposed to address the critical drawback of unbounded SV size and computational cost in online kernel learning. Most existing budget online kernel learning algorithms attempt to achieve a bounded number of SV's through the following major ways:

- *SV Removal.* Some well-known examples include Randomized Budget Perceptron (RBP) (Cavallanti et al., 2007) that randomly removes one existing SV when the number of SV's overflows the budget, Forgetron (Dekel et al., 2005) that simply discards the oldest SV, Budget Online Gradient Descent (BOGD) that basically also discards some old SV, and Budget Perceptron (Crammer et al., 2003) and Budgeted Passive Aggressive (BPA-S) algorithm (Wang and Vucetic, 2010) which attempt to discard the most redundant SV.

- *SV Projection.* By projecting the discarded SV's onto the remaining ones, these algorithms attempt to bound the SV size while reducing the loss due to budget maintenance. Examples include Projectron (Orabona et al., 2008), Budgeted Passive Aggressive Projectron (BPA-P), and Budgeted Passive Aggressive Nearest Neighbor (BPA-NN) (Wang and Vucetic, 2010). Despite achieving better accuracy, they often suffer extremely high computational costs.

- *SV Merging.* These methods attempt to maintain the budget by merging two existing SV's into a new one, such as the Twin Support Vector Machine (TVM) algorithm (Wang and Vucetic, 2009). The similar idea of SV merging was also explored

to bound the number of SV's in Budget Stochastic Gradient Descent (BSGD-M) algorithm in (Wang et al., 2012b).

In contrast to the above approaches, our work explores a completely different approach, i.e., kernel functional approximation techniques, for resolving budget online kernel learning tasks. As a summary, Table 1 compares the properties of different budget online kernel learning algorithms, including the proposed FOGD and NOGD algorithms, where $B$ is the budget on the desired SV size, $D$ is the number of Fourier components, and $k$ is the matrix approximation rank of Nyström.

| Algorithms | Budget Strategy | Update Time | Space |
|---|---|---|---|
| Budget Perceptron | Removal | $O(B^2)$ | $O(B)$ |
| RBP | Removal | $O(B)$ | $O(B)$ |
| Forgetron | Removal | $O(B)$ | $O(B)$ |
| BOGD | Removal | $O(B)$ | $O(B)$ |
| BPA-S | Removal | $O(B)$ | $O(B)$ |
| Projectron | Projection | $O(B^2)$ | $O(B^2)$ |
| BPA-P | Projection | $O(B^3)$ | $O(B^2)$ |
| BPA-NN | Projection | $O(B)$ | $O(B)$ |
| TVM | Merging | $O(B^2)$ | $O(B^2)$ |
| FOGD | Functional Approximation | $O(D)$ | $O(D)$ |
| NOGD | Functional Approximation | $O(kB)$ | $O(kB)$ |

Table 1: Comparison on different budget online kernel learning algorithms.

Moreover, our work is also related to kernel methods for classification tasks (Shawe-Taylor and Cristianini, 2004; Hoi et al., 2006, 2007), especially for some studies on large-scale kernel methods (Williams and Seeger, 2000; Rahimi and Recht, 2007). Our approach shares the similar idea with the Low-rank Linearization SVM (LLSVM) (Zhang et al., 2012), where the non-linear SVM is transformed into a linear problem via kernel approximation methods. Unlike their approach, we employ the technique of random Fourier features (Rahimi and Recht, 2007), which have been successfully explored for speeding up batch kernelized SVMs (Rahimi and Recht, 2007; Yang et al., 2012) and kernel-based clustering (Chitta et al., 2011, 2012) tasks. Besides, another kernel approximation technique used in our approach is the well-known Nyström method (Williams and Seeger, 2000), which has been widely applied in machine learning tasks, including Gaussian Processes (Williams and Seeger, 2000), Kernelized SVMs (Zhang et al., 2012), Kernel PCA, Spectral Clustering (Zhang and Kwok, 2009), and manifold learning (Talwalkar et al., 2008). Although these techniques have been applied for batch machine learning tasks, to the best of our knowledge, they have been seldom explored for online kernel learning tasks as studied in this paper. Finally, we note that the short version of this work had been published in the Proceedings of the Twenty-Third international joint conference on Artificial Intelligence (IJCAI2013) (Wang et al., 2013). The journal manuscript has made significant extension by including substantial amount of new contents and more extensive empirical studies.

## 3. Large Scale Online Kernel Learning for Binary Classification

In this section, we introduce the problem formulation of online kernel binary classification and the detailed steps of our proposed algorithms.

### 3.1 Problem Formulation

We consider the problem of online learning for binary classification by following online convex optimization settings. Our goal is to learn a function $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where instance $\mathbf{x}_t \in \mathbb{R}^d$ and class label $y_t \in \mathcal{Y} = \{+1, -1\}$. We refer to the output $f$ of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \to \mathbb{R}\}$. We will use $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \to \mathbb{R}$ 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 $\kappa(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ (Vapnik, 1998) implementing the inner product$\langle \cdot, \cdot \rangle$ such that: 1) $\kappa$ 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 $\kappa$. Throughout the analysis, we assume $\kappa(\mathbf{x}_i, \mathbf{x}_j) \leq 1$, $\forall \mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$.

Training an SVM classifier $f(\mathbf{x})$ can be formulated as the following optimization problem

$$\min_{f \in \mathcal{H}_\kappa} P(f) = \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2 + \frac{1}{T}\sum_{t=1}^{T} \ell(f(\mathbf{x}_t); y_t),$$

where $\lambda > 0$ is a regularization parameter used to control model complexity. While in an pure online setting, the regularized loss in the $t$-th iteration is

$$\mathcal{L}_t(f) = \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2 + \ell(f(\mathbf{x}_t); y_t).$$

The goal of an online learning algorithm is to find a sequence of functions $f_t$, $t \in [T]$ that achieve the minimum *Regret* along the whole learning process. The regret is defined as,

$$Regret = \sum_{t=1}^{T} \mathcal{L}_t(f_t) - \sum_{t=1}^{T} \mathcal{L}_t(f^*),$$

where $f^* = \arg\min_f \sum_{t=1}^{T} \mathcal{L}_t(f)$ is the optimal classifier assuming that we had foresight in all the training instances. In a typical online budgeted kernel learning algorithm, the algorithm learns the kernel-based predictive model $f(\mathbf{x})$ for classifying a new instance $\mathbf{x} \in \mathbb{R}^d$ as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}),$$

where $B$ is the number of Support Vectors (SV's), $\alpha_i$ denotes the coefficient of the $i$-th SV, and $\kappa(\cdot, \cdot)$ denotes the kernel function. The existing budget online kernel classification

approach aims to bound the number of SV's by a budget constant $B$ using different budget maintenance strategies. Unlike the existing budget online kernel learning methods using the budget maintenance strategies, we propose to tackle the challenge by exploring a completely different strategy, i.e., the kernel functional approximation approach that construct a kernel-induced new representation $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^D$ such that the inner product of instances in the new space is able to approximate the kernel function:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j).$$

By the above approximation, the predictive model can be rewritten:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}(\mathbf{x}),$$

where $\mathbf{w} = \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)$ denotes the weight vector to be learned in the new feature space. As a consequence, solving the regular online kernel classification task can be turned into a problem of an linear online classification task on the new feature space derived from the kernel approximation. In the following, we will present two online kernel learning algorithms for classification by applying two different kernel approximation methods: (i) Fourier Online Gradient Descent (FOGD) and (ii) Nyström Online Gradient Descent (NOGD) methods.

### 3.2 Fourier Online Gradient Descent

Random Fourier features can be used in shift-invariant kernels (Rahimi and Recht, 2007). A shift-invariant kernel is the kernel that can be written as $\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\Delta\mathbf{x})$, where $k$ is some function and $\Delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_2$ is the shift between the two instances. Examples of shift-invariant kernels include some widely used kernels, such as Gaussian and Laplace kernels. By performing an inverse Fourier transform of the shift-invariant kernel function, one can obtain:

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u}, \tag{1}$$

where $p(\mathbf{u})$ is a proper probability density function calculated from the Fourier transform of function $k(\Delta\mathbf{x})$,

$$p(\mathbf{u}) = (\frac{1}{2\pi})^d \int e^{-i\mathbf{u}^\top(\Delta\mathbf{x})} k(\Delta\mathbf{x}) d(\Delta\mathbf{x}). \tag{2}$$

More specifically, for a Gaussian kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2})$, we have the corresponding random Fourier component $\mathbf{u}$ with the distribution $p(\mathbf{u}) = \mathcal{N}(0, \sigma^{-2}I)$. And for a Laplacian kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{||\mathbf{x}_1 - \mathbf{x}_2||_1}{\sigma})$, we have $p(\mathbf{u}) = \sigma\Pi_d \frac{1}{\pi(1+\sigma^2 u_d^2)}$. Given a kernel function that is continuous and positive-definite, according to the Bochner's theorem (Rudin, 1990), the kernel function can be expressed as an expectation of function with a random variable $\mathbf{u}$:

$$\begin{aligned} &\int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u} = \mathrm{E}_{\mathbf{u}}[e^{i\mathbf{u}^\top\mathbf{x}_1} \cdot e^{-i\mathbf{u}^\top\mathbf{x}_2}] \\ &= \mathrm{E}_{\mathbf{u}}[\cos(\mathbf{u}^\top\mathbf{x}_1)\cos(\mathbf{u}^\top\mathbf{x}_2) + \sin(\mathbf{u}^\top\mathbf{x}_1)\sin(\mathbf{u}^\top\mathbf{x}_2)] \\ &= \mathrm{E}_{\mathbf{u}}[[\sin(\mathbf{u}^\top\mathbf{x}_1), \cos(\mathbf{u}^\top\mathbf{x}_1)] \cdot [\sin(\mathbf{u}^\top\mathbf{x}_2), \cos(\mathbf{u}^\top\mathbf{x}_2)]]. \end{aligned} \tag{3}$$

The equality (1) can be obtained by only keeping the real part of the complex function. From (3), we can see any shift-invariant kernel function can be expressed by the expectation of the inner product of the new representation of original data, where the new data representation is $\mathbf{z}(\mathbf{x}) = [\sin(\mathbf{u}^\top \mathbf{x}), \cos(\mathbf{u}^\top \mathbf{x})]^\top$. As a consequence, we can sample $D$ number of random Fourier components $\mathbf{u}_1, ...\mathbf{u}_D$ independently for constructing the new representation as follows:

$$\mathbf{z}(\mathbf{x}) = (\sin(\mathbf{u}_1^\top \mathbf{x}), \cos(\mathbf{u}_1^\top \mathbf{x}), ..., \sin(\mathbf{u}_D^\top \mathbf{x}), \cos(\mathbf{u}_D^\top \mathbf{x}))^\top.$$

The online kernel learning task in the original input space can thus be approximated by solving a linear online learning task in the new feature space. For data arriving sequentially, we can construct the new representation of a data instance on-the-fly, and then perform online learning in the new feature space using the online gradient descent algorithm. We refer to the proposed algorithm as the Fourier Online Gradient Descent (FOGD), as summarized in Algorithm 1.

### 3.3 Nyström Online Gradient Descent

The above random Fourier feature based approach attempts to approximate the kernel function explicitly, which is in general data independent for the given dataset and thus may not fully exploit the potential of data distribution for kernel approximation. To address this, we propose to explore the Nyström method (Williams and Seeger, 2000) to approximate a large kernel matrix by a data-dependent approach.

Before presenting the method, we first introduce some notations. We denote a kernel matrix by $\mathbf{K} \in \mathbb{R}^{T \times T}$ with rank $r$, the Singular Value Decomposition (SVD) of $\mathbf{K}$ as $\mathbf{K} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$, where the columns of $\mathbf{V}$ are orthogonal and $\mathbf{D} = \text{diag}(\sigma_1, \ldots, \sigma_r, )$ is diagonal. For $k < r$, $\mathbf{K}_k = \sum_{i=1}^{k} \sigma_i V_i V_i^\top = \mathbf{V}_k \mathbf{D}_k \mathbf{V}_k^\top$ is the best rank-$k$ approximation of $\mathbf{K}$, where $V_i$ is the $i$-th column of matrix $\mathbf{V}$.

Given a large kernel matrix $\mathbf{K} \in \mathbb{R}^{T \times T}$, the Nyström method randomly samples $B \ll T$ columns to form a matrix $\mathbf{C} \in \mathbb{R}^{T \times B}$, and then derive a much smaller kernel matrix $\mathbf{W} \in \mathbb{R}^{B \times B}$ based on the sampled $B$ instances. We can in turn approximate the original large kernel matrix by

$$\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^+ \mathbf{C}^\top \approx \mathbf{K}, \tag{4}$$

where $\mathbf{W}_k$ is the best rank-$k$ approximation of $\mathbf{W}$, $\mathbf{W}^+$ denotes the pseudo inverse of matrix $\mathbf{W}$.

We now apply the above Nyström based kernel approximation to tackle large-scale online kernel classification task. Similar to the previous approach, the key idea is to construct the new representation for every newly arrived data instance based on the kernel approximation principle. In particular, we propose the following scheme: (i) at the very early stage of the online classification task, we simply run any existing online kernel learning methods (e.g., kernel-based online gradient descent in our approach) whenever the number of SV's is smaller than the predefined budget $B$; (ii) once the budget is reached, we then use the stored $B$ SV's to approximate the kernel value of any new instances (which is equivalent to using the first $B$ columns to approximate the whole kernel matrix). From the approximated kernel matrix in (4), we could see the kernel value between $i$-th instance $\mathbf{x}_i$ and $j$-th instance

**Algorithm 1** FOGD — Fourier Online Gradient Descent for Binary Classification

**Input**: the number of Fourier components $D$, step size $\eta$, kernel function $k$;
**Initialize** $\mathbf{w}_1 = 0$.
Calculate $p(\mathbf{u})$ for kernel $k$ as (2).
Generate random Fourier components: $\mathbf{u}_1, ..., \mathbf{u}_D$ sampled from distribution $p(\mathbf{u})$
**for** $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t$;
  Construct new representation:
  $\mathbf{z}_t(\mathbf{x}_t) = (\sin(\mathbf{u}_1^\top \mathbf{x}_t), \cos(\mathbf{u}_1^\top \mathbf{x}_t), ..., \sin(\mathbf{u}_D^\top \mathbf{x}_t), \cos(\mathbf{u}_D^\top \mathbf{x}_t))^\top$
  Predict $\widehat{y}_t = \mathrm{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t))$;
  Receive $y_t$ and suffer loss $\ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$;
  **if** $\ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big) > 0$ **then**
    $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$.
  **end if**
**end for**

**Algorithm 2** NOGD — Nyström Online Gradient Descent for Binary Classification

**Input**: the budget $B$, step size $\eta$, rank approximation $k$.
**Initialize** support vector set $\mathcal{S}_1 = \emptyset$, and model $f_1 = 0$.
**while** $|\mathcal{S}_t| < B$ **do**
  Receive new instance $\mathbf{x}_t$;
  Predict $\widehat{y}_t = \mathrm{sgn}(f_t(\mathbf{x}_t))$;
  Update $f_t$ by regular Online Gradient Descent (OGD);
  Update $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{t\}$ whenever loss is nonzero;
  $t = t + 1$;
**end while**
Construct the kernel matrix $\hat{\mathbf{K}}_t$ from $\mathcal{S}_t$.
$[\mathbf{V}_k, \mathbf{D}_k] = \mathsf{eigs}(\hat{\mathbf{K}}_t, k)$, where $\mathbf{V}_k$ and $\mathbf{D}_k$ are Eigenvectors and Eigenvalues of $\hat{\mathbf{K}}_t$.
Initialize $\mathbf{w}_t^\top = [\alpha_1, ..., \alpha_B](\mathbf{D}_k^{-0.5}\mathbf{V}_k^\top)^{-1}$.
Initialize the instance index $T_0 = t$;
**for** $t = T_0, \ldots, T$ **do**
  Receive new instance $\mathbf{x}_t$;
  Construct the new representation of $\mathbf{x}_t$:
  $\mathbf{z}(\mathbf{x}_t) = \mathbf{D}_k^{-0.5}\mathbf{V}_k^\top(\kappa(\mathbf{x}_t, \hat{\mathbf{x}}_1), ..., \kappa(\mathbf{x}_t, \hat{\mathbf{x}}_B))^\top$.
  Predict $\widehat{y}_t = \mathrm{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t))$;
  Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$.
**end for**

$\mathbf{x}_j$ is approximated by the following

$$\begin{aligned}\hat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{C}_i \mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})(\mathbf{C}_j \mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})^\top \\ &= ([\kappa(\hat{\mathbf{x}}_1, \mathbf{x}_i), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x}_i)]\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})(\kappa(\hat{\mathbf{x}}_1, \mathbf{x}_j), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x}_j)\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})^\top,\end{aligned}$$

where $\hat{\mathbf{x}}_a, a \in \{1, ..., B\}$ is the $a$-th support vector.

For a new instance, we construct the new representation as follows:

$$\mathbf{z}(\mathbf{x}) = ([\kappa(\hat{\mathbf{x}}_1, \mathbf{x}), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x})]\mathbf{V}_k \mathbf{D}_k^{-\frac{1}{2}})^\top.$$

Similarly, we can then apply the existing online gradient descent algorithm to learn the linear predictive model on the new feature space induced from the kernel. We denote the proposed algorithm the Nyström Online Gradient Descent (NOGD), as summarized in Algorithm 2. Different from the FOGD algorithm, the algorithm follows the kernelized online gradient descent until the number of SV's reaches $B$. To initialize the linear classifier $\mathbf{w}$, we aim to achieve

$$\mathbf{w}^\top \mathbf{z}(\mathbf{x}) = [\alpha_1, ..., \alpha_B][\kappa(\hat{\mathbf{x}}_1, \mathbf{x}), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x})]^\top,$$

thus

$$\mathbf{w}^\top \mathbf{D}_k^{-\frac{1}{2}} \mathbf{V}_k^\top [\kappa(\hat{\mathbf{x}}_1, \mathbf{x}), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x})]^\top = [\alpha_1, ..., \alpha_B][\kappa(\hat{\mathbf{x}}_1, \mathbf{x}), ..., \kappa(\hat{\mathbf{x}}_B, \mathbf{x})]^\top.$$

The solution is

$$\mathbf{w}^\top \mathbf{D}_k^{-\frac{1}{2}} \mathbf{V}_k^\top = [\alpha_1, ..., \alpha_B];$$

$$\mathbf{w}^\top = [\alpha_1, ..., \alpha_B](\mathbf{D}_k^{-\frac{1}{2}} \mathbf{V}_k^\top)^{-1}.$$

## 4. Theoretical Analysis

In this section, we analyze the theoretical properties of the two proposed algorithms. We may use $\ell_t(f)$ instead of $\ell(f(\mathbf{x}_t); y_t)$ for simplicity.

**Theorem 1** *Assume we have a shift-invariant kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2)$, where $k$ is some function and the original data is contained by a ball $\mathbb{R}^d$ of diameter $R$. Let $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \to \mathbb{R}$ be a convex loss function that is Lipschitz continuous with Lipschitz constant $L$. Let $\mathbf{w}_t, t \in [T]$ be the sequence of classifiers generated by FOGD in Algorithm 1. Then, for any $f^*(\mathbf{x}) = \sum_{t=1}^T \alpha_t^* \kappa(\mathbf{x}, \mathbf{x}_t)$, we have the following with probability at least $1 - 2^8(\frac{\sigma_p R}{\epsilon})^2 \exp(\frac{-D\epsilon^2}{4(d+2)})$,*

$$\sum_{t=1}^T \ell_t(\mathbf{w}_t) - \sum_{t=1}^T \ell_t(f^*) \leq \frac{(1+\epsilon)\|f^*\|_1^2}{2\eta} + \frac{\eta}{2} L^2 T + \epsilon L T \|f^*\|_1,$$

*where $\|f^*\|_1 = \sum_{t=1}^T |\alpha_t^*|$, $\sigma_p^2 = E_p[\mathbf{u}^\top \mathbf{u}]$ is the second moment of the Fourier transform of the kernel function $k$ given that $p(\mathbf{u})$ is the probability density function calculated by the Fourier transform of function $k$.*

**Proof** Given $f^*(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t^* \kappa(\mathbf{x}, \mathbf{x}_t)$, according to the representer theorem (Schölkopf et al., 2001), we have a corresponding linear model: $\mathbf{w}^* = \sum_{t=1}^{T} \alpha_t^* \mathbf{z}(\mathbf{x}_t)$, where

$$\mathbf{z}(\mathbf{x}) = (\sin(\mathbf{u}_1^\top \mathbf{x}), \cos(\mathbf{u}_1^\top \mathbf{x}), ..., \sin(\mathbf{u}_D^\top \mathbf{x}), \cos(\mathbf{u}_D^\top \mathbf{x}))^\top.$$

The first step to prove our theorem is to bound the regret of our sequence of linear model $\mathbf{w}_t$ learned by our online learner with respect to the linear model $\mathbf{w}^*$ in the new feature space. First of all, we have the following:

$$\begin{aligned} &\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 = \|\mathbf{w}_t - \eta \nabla \ell_t(\mathbf{w}_t) - \mathbf{w}^*\|^2 \\ =&\|\mathbf{w}_t - \mathbf{w}^*\|^2 + \eta^2 \|\nabla \ell_t(\mathbf{w}_t)\|^2 - 2\eta \nabla \ell_t(\mathbf{w}_t)(\mathbf{w}_t - \mathbf{w}^*). \end{aligned}$$

Combining the above and the convexity of the loss function, i.e.,

$$\ell_t(\mathbf{w}_t) - \ell_t(\mathbf{w}^*) \leq \nabla \ell_t(\mathbf{w}_t)(\mathbf{w}_t - \mathbf{w}^*),$$

we then have the following

$$\ell_t(\mathbf{w}_t) - \ell_t(\mathbf{w}^*) \leq \frac{\|\mathbf{w}_t - \mathbf{w}^*\|^2 - \|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \|\nabla \ell_t(\mathbf{w}_t)\|^2.$$

Summing the above over $t = 1, ..., T$ leads to:

$$\begin{aligned} \sum_{t=1}^{T} (\ell_t(\mathbf{w}_t) - \ell_t(\mathbf{w}^*)) \ &\leq \frac{\|\mathbf{w}_1 - \mathbf{w}^*\|^2 - \|\mathbf{w}_{T+1} - \mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\nabla \ell_t(\mathbf{w}_t)\|^2 \\ &\leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} L^2 T. \end{aligned} \tag{5}$$

Next we further examine the relationship between $\sum_{t=1}^{T} \ell_t(\mathbf{w}^*)$ and $\sum_{t=1}^{T} \ell_t(f^*)$. According to the uniform convergence of Fourier features (Claim 1 in Rahimi and Recht (2007)), we have the high probability bound for the difference between the approximated kernel value and the true kernel value, i.e., with probability at least $1 - 2^8 (\frac{\sigma_p R}{\epsilon})^2 \exp(\frac{-D\epsilon^2}{4(d+2)})$, where $\sigma_p^2 = E_p[\mathbf{u}^\top \mathbf{u}]$ is the second moment of the Fourier transform of the kernel function $k$ given that $p(\mathbf{u})$ is the probability density function calculated by the Fourier transform of function $k$. We have $\forall i, j$

$$|\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) - \kappa(\mathbf{x}_i, \mathbf{x}_j)| < \epsilon.$$

Since we assume $\kappa(\mathbf{x}_i, \mathbf{x}_j) \leq 1$, we can assume $\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) \leq 1 + \epsilon$, which lead to:

$$\|\mathbf{w}^*\|^2 \leq (1 + \epsilon) \|f^*\|_1^2. \tag{6}$$

When $|\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) - \kappa(\mathbf{x}_i, \mathbf{x}_j)| < \epsilon$, we have

$$\begin{aligned} |\sum_{t=1}^{T} \ell_t(\mathbf{w}^*) - \sum_{t=1}^{T} \ell_t(f^*)| \ &\leq \ \sum_{t=1}^{T} |\ell_t(\mathbf{w}^*) - \ell_t(f^*)| \\ &\leq \ \sum_{t=1}^{T} L \sum_{i=1}^{T} |\alpha_i^*| |\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \kappa(\mathbf{x}_i, \mathbf{x}_t)| \\ &\leq \ \sum_{t=1}^{T} L\epsilon \sum_{i=1}^{T} |\alpha_i^*| = \epsilon L T \|f^*\|_1. \end{aligned} \tag{7}$$

Combining (5), (6) and (7) leads to complete the proof. ■

*Remark 1.* In general, the larger the dimensionality $D$, the higher the probability of the bound to be achieved. This means that by sampling more random Fourier components, one can approximate the kernel function more accurately and effectively. From the above theorem, it is not difficult to show that, by setting $\eta = \frac{1}{\sqrt{T}}$ and $\epsilon = \frac{1}{\sqrt{T}}$, we have

$$\sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \sum_{t=1}^{T} \ell_t(f^*) \quad \leq \quad (\frac{2\|f^*\|_1 + L^2}{2} + L\|f^*\|_1)\sqrt{T},$$

which leads to a sub-linear regret $O(\sqrt{T})$. However, setting $\epsilon = \frac{1}{\sqrt{T}}$ requires to sample $D = O(T)$ random components in order to achieve a high probability, which seems unsatisfactory since we will have to solve a very high-dimensional linear online learning problem. However, even in this case, for our FOGD algorithm, the learning time cost for each instance is $O(c_1 T)$, while the time cost for classifying an instance by regular online kernel classification is $O(c_2 T)$, here $c_1$ is the time for a scalar product by FOGD, while $c_2$ is the time for computing the kernel function. Since $c_2 \gg c_1$, our method is still much faster than the regular online kernel classification methods.

*Remark 2.* This theorem bounds the regret for any shift-invariant kernel. Specially, we can get the bound for a Gaussian kernel $k(\mathbf{x}_1 - \mathbf{x}_2) = \exp(-\gamma||\mathbf{x}_1 - \mathbf{x}_2||^2)$, by setting $\sigma_p^2 = 2d\gamma$ (Rahimi and Recht, 2007).

The theoretical analysis for the NOGD algorithm follows the similar procedure as used by the FOGD algorithm. We first introduce a lemma to facility our regret bound analysis.

**Lemma 1** *$P(f) = \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2 + \frac{1}{T}\sum_{t=1}^{T} \ell_t(f)$ is the objective function of an SVM problem, where $\ell_t(f)$ is the hinge loss function of the $t$-th iteration. Define $f^*$ to be the optimal solution when using the exact kernel matrix $\mathbf{K}$ and $f_N$ as the optimal when adopting the Nyström approximated kernel matrix $\hat{\mathbf{K}}$. We have*

$$P(f_N) - P(f^*) \leq \frac{1}{2T\lambda}||\mathbf{K} - \hat{\mathbf{K}}||_2,$$

*where $\|\mathbf{K} - \hat{\mathbf{K}}\|_2$ is the spectral norm of the kernel approximation gap.*

This lemma mainly follows the Lemma 1 in (Yang et al., 2012), we omit the proof for conciseness.

**Theorem 2** *Assume we learn with kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) \leq 1$, $\forall i, j \in [T]$. Let $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \to \mathbb{R}$ be the hinge loss function. Let the sequence of $T$ instances $\mathbf{x}_1, ..., \mathbf{x}_T$ form a kernel matrix $\mathbf{K} \in \mathbb{R}^{T \times T}$, and $\hat{\mathbf{K}}$ is the approximation of $\mathbf{K}$ using Nyström method. Let $f_t(\mathbf{x}) = \mathbf{w}_t^\top \mathbf{z}(\mathbf{x}), t \in [T]$ be the sequence of classifiers generated by NOGD in Algorithm 2. We assume the norm of the gradients in all iterations are always bounded by a constant $L$, which is easy to achieve by a few projection steps when necessary. In addition, define $f_N(\mathbf{x}) = \mathbf{w}_N^\top \mathbf{z}(\mathbf{x})$ be the optimal classifier when using Nyström kernel approximation and assuming we had foresight for all instances. By defining $f^*$ as the optimal classifier in the*

*original kernel space with the assumption of the foresight for all the instances, we have the following:*

$$\sum_{t=1}^{T}\mathcal{L}_t(\mathbf{w}_t)-\sum_{t=1}^{T}\mathcal{L}_t(f^*)\leq\frac{\|\mathbf{w}_N\|^2}{2\eta}+\frac{\eta}{2}L^2T+\frac{1}{2\lambda}||\mathbf{K}-\hat{\mathbf{K}}||_2.$$

**Proof** Following the similar analysis as in Theorem 1, the regularized loss function in the $t$-th iteration, $\mathcal{L}_t(\mathbf{w})=\frac{\lambda}{2}||\mathbf{w}||_2^2+\ell_t(\mathbf{w})$ satisfies the following bound,

$$\sum_{t=1}^{T}(\mathcal{L}_t(\mathbf{w}_t)-\mathcal{L}_t(\mathbf{w}_N))\leq\frac{\|\mathbf{w}_N\|^2}{2\eta}+\frac{\eta}{2}L^2T.$$

As proven in (Yang et al., 2012), the linear optimization problem in $\mathbf{z}(\mathbf{x})$ space, i.e., $P(\mathbf{w})=\frac{\lambda}{2}||\mathbf{w}||_2^2+\frac{1}{T}\sum_{t=1}^{T}\ell_t(\mathbf{w})$ is equivalent to the approximated SVM $P(f)=\frac{\lambda}{2}||f||_{\mathcal{H}}^2+\frac{1}{T}\sum_{t=1}^{T}\ell_t(f)$ when $f\in\mathcal{H}_N$ is the functional space of Nyström method. From Lemma 1, we have,

$$\sum_{t=1}^{T}\mathcal{L}_t(\mathbf{w}_N)=\sum_{t=1}^{T}\mathcal{L}_t(f_N)=TP(f_N)\leq TP(f^*)+\frac{1}{2\lambda}||\mathbf{K}-\hat{\mathbf{K}}||_2=\sum_{t=1}^{T}\mathcal{L}(f^*)+\frac{1}{2\lambda}||\mathbf{K}-\hat{\mathbf{K}}||_2.$$

We complete the proof by combining the above two formulas. ■

*Remark.* As shown in Theorem 5 of (Yang et al., 2012), the kernel approximation gap $||\mathbf{K}-\hat{\mathbf{K}}||_2\leq O(\frac{T}{B})$. Consequently when setting $B=\sqrt{T}$ and $\eta=\frac{1}{\sqrt{T}}$, we have

$$\sum_{t=1}^{T}\mathcal{L}_t(\mathbf{w}_t)-\sum_{t=1}^{T}\mathcal{L}(f^*)\leq O(\sqrt{T}).$$

This theorem bounds the regret of the NOGD algorithm when using all singular values of matrix $\mathbf{W}$ but only $O(\sqrt{T})$ support vectors. However, when the rank of the Nyström approximated matrix $\hat{\mathbf{K}}$ is only $k$, the bound should be slightly worse. As (Cortes et al., 2010)(Theorem 1) shows, the following holds with probability at least $1-\epsilon$,

$$\|\hat{\mathbf{K}}-\mathbf{K}\|_2\leq\|\mathbf{K}-\mathbf{K}_k\|_2+\frac{T}{\sqrt{B}}\mathbf{K}_{\max}(2+\log\frac{1}{\epsilon}),$$

where $\|\mathbf{K}-\mathbf{K}_k\|_2$ is the spectral norm of the best rank-$k$ approximated gap and $\mathbf{K}_{\max}$ is the maximum diagonal entry of $\mathbf{K}$. This indicates that to get the $O(\sqrt{T})$ regret bound, we should set the budget size $B=T/c$, where $c$ is some constant. This seems suboptimal, but we will demonstrate that only a small budget size is needed for satisfactory performance in our experimental results. In addition, the time cost of using $k$-rank approximation is only $k/B$ times of that when using all singular values, which makes the algorithm extremely efficient.

## 5. Large Scale Online Kernel Learning for Multi-class Classification

In this section, we extend the proposed Fourier Online Gradient Descent and Nyström Online Gradient Descent methods, which are originally designed for binary classification, to online multi-class classification task. We also give theoretical analysis of the two approaches.

### 5.1 Problem Settings

Similar to online binary classification tasks, online multi-class classification is performed over a sequence of training examples $(\mathbf{x}_t, y_t)$, $t = 1, \ldots, T$, where $\mathbf{x}_t \in \mathbb{R}^d$ is the observed features of the $t$-th training instance. Unlike binary classification where class label $y_t \in \mathcal{Y} = \{+1, -1\}$, in a multi-class classification task, each label belongs to a finite set $\mathcal{Y}$ of size $m > 2$, i.e., $y_t \in \mathcal{Y} = \{1, \ldots, m\}$. The true class label $y_t$ is only revealed after the prediction $\hat{y}_t \in \mathcal{Y}$ is made.

We follow the protocol of multi-prototype classification for deriving multi-class online learning algorithm (Crammer et al., 2006). Specifically, it learns a function $f^r : \mathbb{R}^d \to \mathbb{R}$ for each of the classes $r \in \mathcal{Y}$. During the $t$-th iteration, the algorithm predicts a sequence of scores for the classes:

$$\big(f_t^1(\mathbf{x}_t), \ldots, f_t^m(\mathbf{x}_t)\big).$$

The predicted class is set to be the class with the highest prediction score:

$$\hat{y}_t = \arg\max_{r \in \mathcal{Y}} f_t^r(\mathbf{x}_t). \tag{8}$$

We then define $s_t$ as the irrelevant class with the highest prediction score:

$$s_t = \arg\max_{r \in \mathcal{Y}, r \neq y_t} f_t^r(\mathbf{x}_t).$$

The margin with respect to the hypothesis in the $t$-th iteration is defined to be the gap between the prediction score of class $y_t$ and $s_t$:

$$\gamma_t = f_t^{y_t}(\mathbf{x}_t) - f_t^{s_t}(\mathbf{x}_t).$$

Obviously, in a correct prediction, the margin $\gamma_t > 0$. However, as stated in (Crammer et al., 2006), we are not satisfied by a positive margin value and thus we define a *hinge-loss* function:

$$\ell\big(\overline{f}_t, \mathbf{x}_t, y_t\big) = \max(0, 1 - \gamma_t),$$

where $\overline{f}_t$ denotes the set of all $m$ functions for all classes.

### 5.2 Multi-class Fourier Online Gradient Descent

As introduced in the previous section for binary classification task, the online multi-class kernel classification task in the original input space can also be approximated by solving a linear online learning task in the new feature space. First, we use the same Fourier feature mapping approach as in the binary case to map each input instance $\mathbf{x}_t$ to $\mathbf{z}(\mathbf{x}_t)$. Then Online Gradient Descent method is applied to learn a linear classifier.

Following the multi-class problem setting, we learn a weight vector $\mathbf{w}^r \in \mathbb{R}^d$ for each of the classes $r \in \mathcal{Y}$. And use the linear classifier $f^r(\mathbf{x}_t) = \mathbf{w}^r \cdot \mathbf{z}(\mathbf{x}_t)$ to approximate to kernel prediction score. During the $t$-th iteration, the algorithm predicts a sequence of scores for the $m$ classes:

$$\big(\mathbf{w}_t^1 \cdot \mathbf{z}(\mathbf{x}_t), \ldots, \mathbf{w}_t^m \cdot \mathbf{z}(\mathbf{x}_t)\big).$$

We define the *hinge-loss* function as following:

$$\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big) = \max(0, 1 - \gamma_t) = \max(0, 1 - \mathbf{w}_t^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) + \mathbf{w}_t^{s_t} \cdot \mathbf{z}(\mathbf{x}_t)), \tag{9}$$

where $\overline{\mathbf{w}}_t$ denotes the set of all $m$ weight vectors.

Following the Online Gradient Descent approach, the update strategy of $\overline{\mathbf{w}_t}$ when $\ell > 0$ is:

$$\mathbf{w}_{t+1}^r = \mathbf{w}_t^r - \eta \nabla \ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big),$$

where $\eta$ is a positive learning rate parameter and the gradient is taken with regards to $\mathbf{w}_t^r$. By rewriting the loss function explicitly, we can rewrite the above as follows:

$$\mathbf{w}_{t+1}^{y_t} = \mathbf{w}_t^{y_t} + \eta \mathbf{z}(\mathbf{x}_t); \tag{10}$$

$$\mathbf{w}_{t+1}^{s_t} = \mathbf{w}_t^{s_t} - \eta \mathbf{z}(\mathbf{x}_t). \tag{11}$$

Only two of the weight vectors are updated during each iteration. Thus, we can derive the FOGD algorithm for multi-class versions, as summarized in Algorithm 3.

### 5.3 Multi-class Nyström Online Gradient Descent

Similar to the binary task, in the first a few iterations of multi-class online Nyström algorithm (before the size of SV set reaches the predetermined budget size $B$), the algorithm performs a regular Online Gradient Descent update to the $m$ kernel classifiers when $\ell > 0$:

$$f_{t+1}^r = f_t^r - \eta \nabla \ell\big(\overline{f}_t, \mathbf{x}_t, y_t\big),$$

where $\eta > 0$ is the gradient descent step size and the gradient is taken with regard to $f_t^r$. By rewriting the loss function explicitly, we have

$$f_{t+1}^{y_t} = f_t^{y_t} + \eta \kappa(\mathbf{x}_t, \cdot); \tag{12}$$

$$f_{t+1}^{s_t} = f_t^{s_t} - \eta \kappa(\mathbf{x}_t, \cdot). \tag{13}$$

Therefore, we need to store a SV set $\mathcal{S}$ and update it when necessary: $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{t\}$. Note that all the $m$ classifiers share the same SV set, which is the same setting as that of binary case. However, the storage strategy for $\alpha_i$, i.e., the coefficient of the $i$-th SV, is different from that of binary case. In multi-class task, a vector $\alpha_i \in \mathbb{R}^m$ is used to represent the coefficients of the $i$-th SV. Each of its element $\alpha_i^r$ is the coefficient of the $i$-th SV for the kernel classifier $f^r$. Obviously, $\alpha_i^r = 0$ if $r \neq y_t$ and $r \neq s_t$.

After the size of SV set reaches the budget $B$, we do a Nyström feature mapping as the approach discussed in the binary section to map each input instance $\mathbf{x}_t$ to $\mathbf{z}(\mathbf{x}_t)$. The following linear update steps follow that in the multi-class Fourier Gradient Descent algorithm, as equation (10) and (11).

We derive the NOGD algorithm for multi-class version, as summarized in Algorithm 4.

**Algorithm 3** MFOGD — Multi-class Fourier Online Gradient Descent

**Input**: the number of Fourier components $D$, step size $\eta$, kernel function $k$;
**Initialize** $\mathbf{w}_1^r = 0$, $r = 1, ..., m$.
Calculate $p(\mathbf{u})$ for kernel $k$ as (2).
Generate random Fourier components: $\mathbf{u}_1, ..., \mathbf{u}_D$ sampled from distribution $p(\mathbf{u})$.
**for** $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t$;
  Construct new representation:
  $\mathbf{z}(\mathbf{x}_t) = (\sin(\mathbf{u}_1^\top \mathbf{x}_t), \cos(\mathbf{u}_1^\top \mathbf{x}_t), ..., \sin(\mathbf{u}_D^\top \mathbf{x}_t), \cos(\mathbf{u}_D^\top \mathbf{x}_t))^\top$
  Predict as in (8);
  Receive $y_t$ and suffer loss $\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big)$ (9);
  **if** $\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big) > 0$ **then**
    update $\overline{\mathbf{w}}_t$ as (10) and (11)
  **end if**
**end for**

**Algorithm 4** MNOGD — Multi-class Nyström Online Gradient Descent

**Input**: the budget $B$, step size $\eta$, rank approximation $k$.
**Initialize** support vector set $\mathcal{S}_1 = \emptyset$, and model $f_1^r = 0$, $r = 1, ..., m$.
**while** $|\mathcal{S}_t| < B$ **do**
  Receive $\mathbf{x}_t$;
  Predict as in (8);
  Update $\overline{f_t}$ by regular Online Gradient Descent (OGD), as (12) and (13);
  Update $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{t\}$ whenever loss is nonzero;
  $t = t + 1$;
**end while**
Construct the kernel matrix $\hat{\mathbf{K}}_t$ from $\mathcal{S}_t$.
$[\mathbf{V}_k, \mathbf{D}_k] = \mathsf{eigs}(\hat{\mathbf{K}}_t, k)$, where $\mathbf{V}_k$ and $\mathbf{D}_k$ are Eigenvectors and Eigenvalues of $\hat{\mathbf{K}}_t$.
Initialize $\mathbf{w}_t^{r\top} = [\alpha_1^r, ..., \alpha_B^r](\mathbf{D}_k^{-0.5}\mathbf{V}_k^\top)^{-1}$, $r = 1, ..., m$.
Initialize the instance index $T_0 = t$;
**for** $t = T_0, \ldots, T$ **do**
  Receive $\mathbf{x}_t$;
  Construct the new representation of $\mathbf{x}_t$:
  $\mathbf{z}(\mathbf{x}_t) = \mathbf{D}_k^{-0.5}\mathbf{V}_k^\top(\kappa(\mathbf{x}_t, \hat{\mathbf{x}}_1), ..., \kappa(\mathbf{x}_t, \hat{\mathbf{x}}_B))^\top$.
  Predict as in (8);
  Update $\overline{\mathbf{w}}_t$ as (10) and (11)
**end for**

### 5.4 Theoretical Analysis

In this section, we analyze the theoretical properties of the two proposed multi-class algorithms. We may use $\ell_t(\overline{f})$ instead of $\ell(\overline{f}, \mathbf{x}_t, y_t)$ for simplicity.

**Theorem 3** *Assume we have a shift-invariant kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2)$, where $k$ is some function and the original data is contained by a ball $\mathbb{R}^d$ of diameter $R$. Let $\ell\big(\overline{f}, \mathbf{x}, y\big)$ be the hinge-loss we talked above, where $\overline{f}$ denotes the set of $m$ classifiers for the $m$ classes and its output is a sequence of prediction scores. Let $\overline{\mathbf{w}}_t, t \in [T]$ be the sequence of classifiers generated by MFOGD in Algorithm 3. Then, for any $f_*^r(\mathbf{x}) = \sum_{t=1}^T \alpha_t^{r*}\kappa(\mathbf{x}, \mathbf{x}_t)$, $r \in \{1, ..., m\}$, we have the following with probability at least $1 - 2^8(\frac{\sigma_p R}{\epsilon})^2 \exp(\frac{-D\epsilon^2}{4(d+2)})$*

$$\sum_{t=1}^T \ell_t(\overline{\mathbf{w}}_t) - \sum_{t=1}^T \ell_t(\overline{f_*}) \leq \frac{m(1+\epsilon)\|f_*\|_1^2}{2\eta} + \eta T D + 2T\epsilon\|f_*\|_1,$$

*where $\|f_*\|_1 = \sum_{t=1}^T \max_{r\in\mathcal{Y}} |\alpha_t^{r*}|$, and $\sigma_p^2 = E_p[\mathbf{u}^\top \mathbf{u}]$ is the second moment of the Fourier transform of the kernel function $k$ given that $p(\mathbf{u})$ is the probability density function calculated by the Fourier transform of function $k$.*

**Proof** Given $f_*^r(\mathbf{x}) = \sum_{t=1}^T \alpha_t^{r*}\kappa(\mathbf{x}, \mathbf{x}_t)$, $r = 1, ...m$, according to the Representer Theorem (Schölkopf et al., 2001), we have a corresponding linear model: $\mathbf{w}_*^r = \sum_{t=1}^T \alpha_t^{r*}\mathbf{z}(\mathbf{x}_t)$, where

$$\mathbf{z}(\mathbf{x}) = (\sin(\mathbf{u}_1^\top \mathbf{x}), \cos(\mathbf{u}_1^\top \mathbf{x}), ..., \sin(\mathbf{u}_D^\top \mathbf{x}), \cos(\mathbf{u}_D^\top \mathbf{x}))^\top.$$

The first step to prove our theorem is to bound the regret of the sequence of linear models $\mathbf{w}_t$ learned by online learner with respect to the linear model $\mathbf{w}^*$ in the new feature space. We first have

$$\begin{aligned} \|\mathbf{w}_{t+1}^r - \mathbf{w}_*^r\|^2 - \|\mathbf{w}_t^r - \mathbf{w}_*^r\|^2 &= \|\mathbf{w}_t^r - \mathbf{w}_*^r + \beta_t^r \mathbf{z}(\mathbf{x}_t)\|^2 - \|\mathbf{w}_t^r - \mathbf{w}_*^r\|^2 \\ &= (\beta_t^r)^2\|\mathbf{z}(\mathbf{x}_t)\|^2 + 2\beta_t^r(\mathbf{w}_t^r \cdot \mathbf{z}(\mathbf{x}_t) - \mathbf{w}_*^r \cdot \mathbf{z}(\mathbf{x}_t)), \end{aligned}$$

where $\beta_t^r$ is the parameter used to update $\mathbf{w}_t^r$ as (10) and (11). Thus, it may be $\eta$, $-\eta$, or 0. Obviously, $\|\mathbf{z}(\mathbf{x}_t)\|^2 = D$. We first assume that $\ell_t(\overline{\mathbf{w}}_t) > 0$ and there are updates in the $t$-th iteration. Summing the above over $r = 1, ..., m$, we have

$$\begin{aligned} &\sum_{r=1}^m \left(\|\mathbf{w}_{t+1}^r - \mathbf{w}_*^r\|^2 - \|\mathbf{w}_t^r - \mathbf{w}_*^r\|^2\right) \\ &= 2\eta^2 D + 2\eta(\mathbf{w}_t^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) - \mathbf{w}_t^{s_t} \cdot \mathbf{z}(\mathbf{x}_t)) - 2\eta(\mathbf{w}_*^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) - \mathbf{w}_*^{s_t} \cdot \mathbf{z}(\mathbf{x}_t)), \end{aligned}$$

where $y_t$ is the true label in the $t$-th iteration and $s_t$ is the label of the largest scored irrelevant label classified by the classifier set $\overline{\mathbf{w}}_t$, not by the classifier set $\overline{\mathbf{w}}_*$. Thus we have:

$$\mathbf{w}_t^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) - \mathbf{w}_t^{s_t} \cdot \mathbf{z}(\mathbf{x}_t) = \gamma_{\mathbf{w},t} \qquad \mathbf{w}_*^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) - \mathbf{w}_*^{s_t} \cdot \mathbf{z}(\mathbf{x}_t) \geq \gamma_{\mathbf{w}*,t}.$$

As we have assumed $\ell_t(\overline{\mathbf{w}}_t) > 0$, we have $\ell_t(\overline{\mathbf{w}}_t) = 1 - \gamma_{\mathbf{w},t}$. And from the fact that $\ell_t(\overline{\mathbf{w}}_*) \geq 1 - \gamma_{\mathbf{w}*,t}$, we have:

$$\gamma_{\mathbf{w},t} - \gamma_{\mathbf{w}*,t} \leq 1 - \ell_t(\overline{\mathbf{w}}_t) - (1 - \ell_t(\overline{\mathbf{w}}_*)) = \ell_t(\overline{\mathbf{w}}_*) - \ell_t(\overline{\mathbf{w}}_t).$$

Combining the three formula above, we get

$$\ell_t(\overline{\mathbf{w}}_t) - \ell_t(\overline{\mathbf{w}}_*) \leq \eta D + \frac{\sum_{r=1}^m \|\mathbf{w}_t^r - \mathbf{w}_*^r\|^2 - \sum_{r=1}^m \|\mathbf{w}_{t+1}^r - \mathbf{w}_*^r\|^2}{2\eta}.$$

Note that in some iterations, $\ell_t(\overline{\mathbf{w}}_t) = 0$ and there is no update in the $t$-th iteration, i.e., $\mathbf{w}_t^r = \mathbf{w}_{t+1}^r$. The above formula still holds. Summing it over $t = 1, ...T$ and assuming $\mathbf{w}_1^r = 0$ for all $r = 1, ..., m$ leads to:

$$\sum_{t=1}^T \ell_t(\overline{\mathbf{w}}_t) - \sum_{t=1}^T \ell_t(\overline{\mathbf{w}}_*) \leq \eta T D + \frac{\sum_{r=1}^m \|\mathbf{w}_*^r\|^2}{2\eta}. \tag{14}$$

Next we further examine the relationship between $\sum_{t=1}^T \ell_t(\overline{\mathbf{w}}_*)$ and $\sum_{t=1}^T \ell_t(\overline{f_*})$. According to the uniform convergence of Fourier features (Claim 1 in Rahimi and Recht (2007)), we have the high probability bound for the difference between the approximated kernel value and the true kernel value, i.e., with probability at least $1 - 2^8(\frac{\sigma_p R}{\epsilon})^2 \exp(\frac{-D\epsilon^2}{4(d+2)})$, where $\sigma_p^2 = E_p[\mathbf{u}^\top \mathbf{u}]$ is the second moment of the Fourier transform of the kernel function $k$ given that $p(\mathbf{u})$ is the probability density function calculated by the Fourier transform of function $k$. We have $\forall i, j$

$$|\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) - \kappa(\mathbf{x}_i, \mathbf{x}_j)| < \epsilon.$$

Similar to the binary case:

$$\sum_{r=1}^m \|\mathbf{w}_*^r\|^2 \leq m\|f_*\|_1^2(1+\epsilon). \tag{15}$$

When $|\mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) - \kappa(\mathbf{x}_i, \mathbf{x}_j)| < \epsilon$, we have

$$\begin{aligned}
&|\sum_{t=1}^T \ell_t(\overline{\mathbf{w}}_*) - \sum_{t=1}^T \ell_t(\overline{f_*})| \leq \sum_{t=1}^T |\ell_t(\overline{\mathbf{w}}_*) - \ell_t(\overline{f_*})| \leq \sum_{t=1}^T L|\gamma_{\mathbf{w}^*,t} - \gamma_{\mathbf{f}^*,t}| \\
&= \sum_{t=1}^T |\sum_{i=1}^T (\alpha_i^{y_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \alpha_i^{s_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t)) - \sum_{i=1}^T (\alpha_i^{y_t *} \kappa(\mathbf{x}_i, \mathbf{x}_t) - \alpha_i^{s_t' *} \kappa(\mathbf{x}_i, \mathbf{x}_t))| \\
&\leq \sum_{t=1}^T \big( \sum_{i=1}^T |\alpha_i^{y_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \alpha_i^{y_t *} \kappa(\mathbf{x}_i, \mathbf{x}_t)| + \sum_{i=1}^T |\alpha_i^{s_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \alpha_i^{s_t' *} \kappa(\mathbf{x}_i, \mathbf{x}_t)| \big),
\end{aligned}$$

where $L$ in the first line is the Lipschitz constant and can be set to 1 in hinge loss case, and $s_t$ is the largest scored irrelevant label with regards to $\overline{\mathbf{w}}_*$ and $s_t'$ with regard to $\overline{f_*}$. Thus the bound of the first term is easy to find and we will focus on the second term. Without loss of generality, we assume $\alpha_i^{s_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) \geq \alpha_i^{s_t' *} \kappa(\mathbf{x}_i, \mathbf{x}_t)$. According to the definition of $s_t$ and $s_t'$, $\alpha_i^{s_t' *} \kappa(\mathbf{x}_i, \mathbf{x}_t) > \alpha_i^{s_t *} \kappa(\mathbf{x}_i, \mathbf{x}_t)$, leading to:

$$|\alpha_i^{s_t *} \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \alpha_i^{s_t' *} \kappa(\mathbf{x}_i, \mathbf{x}_t)| \leq |\alpha_i^{s_t *} \big( \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_t) - \kappa(\mathbf{x}_i, \mathbf{x}_t) \big)| \leq |\alpha_i^{s_t *}| \epsilon.$$

Consequently, we have

$$|\sum_{t=1}^{T}\ell_t(\overline{\mathbf{w}}_*) - \sum_{t=1}^{T}\ell_t(\overline{f_*})| \leq 2T\epsilon\|f_*\|_1. \tag{16}$$

Combining (14), (15) and (16) leads to complete the proof. ■

Similar to the analysis of the binary classification case, it is not difficult to observe that the proposed MFOGD algorithm also achieves sub-linear regret $O(\sqrt{T})$. We will continue the analysis of MNOGD method in the following. As in the binary analysis, we first propose a lemma that bounds the gap of averaged loss between the exact kernel SVM and approximated kernel SVM. Unlike the binary classification problem, there are many different problem settings for the multi-class SVM problem, as surveyed by Hsu and Lin (2002). For consistency, in the following analysis, we adopt the common slack variables for all classes setting (Crammer and Singer, 2001, 2002).

**Lemma 2** *$P(\overline{f}) = \frac{\lambda}{2}\sum_{r=1}^{m}\|f^r\|_{\mathcal{H}}^2 + \frac{1}{T}\sum_{t=1}^{T}\ell_t(\overline{f})$ is the objective function of an multi-class SVM problem, where $\ell_t(\overline{f})$ is the multi-class hinge loss function of the t-th iteration. Define $\overline{f^*}$ as the optimal solution of $P(\overline{f})$ when using the exact kernel matrix $\mathbf{K}$ and $\overline{f_N}$ as the optimal solution of SVM algorithm when adopting the Nyström approximated kernel matrix $\hat{\mathbf{K}}$. We have*

$$P(\overline{f_N}) - P(\overline{f^*}) \leq \frac{m}{2T\lambda}||\mathbf{K} - \hat{\mathbf{K}}||_2,$$

*where $\|\mathbf{K} - \hat{\mathbf{K}}\|_2$ is the spectral norm of the kernel approximation gap.*

**Proof** The dual problem of multi-class SVM is

$$\begin{aligned}\max P(\boldsymbol{\alpha}) &= -\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{K}\boldsymbol{\alpha}_r - \sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{e}_r,\\ s.t.\quad &\sum_{r=1}^{m}\alpha_{t,r} = 0,\ \forall t \in \{1,2,...,T\};\\ \alpha_{t,r} \leq 0,\ \text{if } y_t \neq r,\quad &\alpha_{t,r} \leq \frac{1}{T}, \text{if } y_t = r,\quad \forall t \in \{1,2,...,T\},\quad \forall r \in \{1,2,...m\}\end{aligned}$$

where $\boldsymbol{\alpha}_r = [\alpha_{1,r}, \alpha_{2,r}, ..., \alpha_{T,r}]^\top$, $\mathbf{e}_r = [e_{1,r}, e_{2,r}, ..., e_{T,r}]^\top$ and $e_{t,r} = 0$ if $y_t = r$, otherwise $e_{t,r} = 1$.

We can get the dual problem of the Nyström approximated multi-class SVM by replacing the kernel matrix $\mathbf{K}$ with $\hat{\mathbf{K}}$.

$$\begin{aligned}P(\overline{f_N}) &= \max\left[-\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\hat{\mathbf{K}}\boldsymbol{\alpha}_r - \sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{e}_r\right]\\ &= \max\left[-\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top(\hat{\mathbf{K}} - \mathbf{K} + \mathbf{K})\boldsymbol{\alpha}_r - \sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{e}_r\right]\\ &= \max\left[\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top(\mathbf{K} - \hat{\mathbf{K}})\boldsymbol{\alpha}_r\right] + \max\left[-\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{K}\boldsymbol{\alpha}_r - \sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top\mathbf{e}_r\right].\end{aligned}$$

Consequently,

$$P(\overline{f_N}) - P(\overline{f^*}) \leq \max \frac{1}{2\lambda} \sum_{r=1}^{m} \boldsymbol{\alpha}_r^\top (\mathbf{K} - \hat{\mathbf{K}}) \boldsymbol{\alpha}_r \leq \max \sum_{r=1}^{m} \frac{1}{2\lambda} ||\boldsymbol{\alpha}_r||_2^2 ||\mathbf{K} - \hat{\mathbf{K}}||_2.$$

We complete the proof by considering $|\alpha_{t,r}| \leq \frac{1}{T}$. ■

**Theorem 4** *Assume we learn with kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) \leq 1$, $\forall i, j \in [T]$. Let $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \to \mathbb{R}$ be the multi-class hinge loss function. Let the sequence of $T$ instances $\mathbf{x}_1, ..., \mathbf{x}_T$ form a kernel matrix $\mathbf{K} \in \mathbb{R}^{T \times T}$, and $\hat{\mathbf{K}}$ is the approximation of $\mathbf{K}$ using Nyström method . Let $f_t^r(\mathbf{x}) = \mathbf{w}_{t,r}^\top \mathbf{z}(\mathbf{x}), t \in [T], r \in \{1, 2, ..., m\}$ be the sequence of classifiers generated by NOGD in Algorithm 4. We assume the norm of the gradients in all iterations are always bounded by a constant $L$, which is easy to achieve by a few projection steps when necessary. In addition, define $f_N^r(\mathbf{x}) = \mathbf{w}_{N,r}^\top \mathbf{z}(\mathbf{x}), r \in \{1, 2, ..., m\}$ be the optimal classifier when using Nyström kernel approximation and assuming we had foresight for all instances. By defining $\overline{f}^*$ as the optimal classifier in the original kernel space with the assumption of the foresight for all the instances, we have the following:*

$$\sum_{t=1}^{T} \mathcal{L}_t(\overline{\mathbf{w}_t}) - \sum_{t=1}^{T} \mathcal{L}_t(\overline{f^*}) \leq \sum_{r=1}^{m} \frac{\|\mathbf{w}_{N,r}\|^2}{2\eta} + \frac{\eta}{2} L^2 T + \frac{m}{2\lambda} ||\mathbf{K} - \hat{\mathbf{K}}||_2.$$

This theorem is a combination of standard online gradient descent analysis and Lemma 2. The proof is omitted since it is similar to the binary analysis and straightforward. It's easy to find that the multi-class NOGD algorithm enjoys the similar regret bound as the binary algorithm.

## 6. Large Scale Online Kernel Learning for Regression

In this section, we extend the proposed FOGD and NOGD algorithms to tackle online regression tasks. Consider a typical online regression task with a sequence of instances $(\mathbf{x}_t, y_t), t = 1, ..., T$, where $\mathbf{x}_t \in \mathbb{R}^d$ is the feature vector of the $t$-th instance and $y_t \in \mathbb{R}$ is the real target value, which is only revealed after the prediction is made at each iteration. The goal of online kernel regression task is to learn a model $f(\mathbf{x})$ that maps a new input instance $\mathbf{x} \in \mathbb{R}^d$ to a real value prediction:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}).$$

We apply the squared loss as the evaluation metric of regression accuracy:

$$\ell(f(\mathbf{x}_t); y_t) = (f(\mathbf{x}_t) - y_t)^2.$$

As the same approximation strategy with the previous task, with a feature mapping function $\mathbf{z}(\mathbf{x})$, the kernel regression task can be tackled by solving its approximated problem: to find a linear model $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}(\mathbf{x})$ that minimizes the accumulated squared loss of all the

training instances. We use online gradient descent algorithm in this new feature space. In order to reduce the frequency of update, we define $\epsilon$ as the threshold. Update is only performed when the loss exceeds threshold $\epsilon$. We denote the proposed algorithms the FOGD for Regression (FOGD-R) and NOGD (NOGD-R) for regression tasks, as summarized in Algorithm 5 and Algorithm 6.

We omit the theoretical analysis of regression since it is similar to the binary case.

## 7. Experimental Results

In this section, we conduct an extensive set of experiments to examine the efficacy of the proposed algorithms for several kinds of learning tasks in varied settings. Specifically, our first experiment is to evaluate the empirical performance of the proposed FOGD and NOGD algorithms for regular binary classification tasks by following a standard batch learning setting where each dataset is divided into two parts: training set and test set. This experiment aims to make a direct comparison of the proposed algorithms with some state-of-the-art approaches for solving batch classification tasks.

Our second major set of experiments is to evaluate the effectiveness and efficiency of the proposed FOGD and NOGD algorithms for online learning tasks by following a purely online learning setting, where the performance measures are based on average mistake rate and time cost accumulated in the online learning process on the entire dataset (there is no split of training and test sets). In particular, we conduct such experiments for three different online learning tasks: binary classification, multi-class classification, and regression, by comparing the proposed algorithms with a variety of state-of-the-art budget online kernel learning algorithms.

All the source code and datasets for our experiments in this work can be downloaded from our project web page:`http://LSOKL.stevenhoi.org/`. We are planning to release our algorithms in the future release of the LIBOL library (Hoi et al., 2014).

### 7.1 Experiment for Binary Classification Task in Batch Setting

In this section, we compare our proposed algorithms with many state-of-the-art batch classification algorithms. Different from online learning, the aim of a batch learning task is to train a classifier on the training dataset so that it achieves the best generalized accuracy on the test dataset.

#### 7.1.1 Experimental Test bed and Setups

Table 2 summarizes the details of the datasets used in this experiment. All of them can be downloaded from LIBSVM website [1] or KDDCUP competition site [2]. We follow the original splits of training and test sets in LIBSVM. For KDD datasets, a random split of 4/1 is used.

We compare the proposed algorithms with the following widely used algorithms for training kernel SVM for batch classification tasks:

1. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/`
2. `http://www.sigkdd.org/kddcup/`

**Algorithm 5** FOGD-R — Fourier Online Gradient Descent for Regression

**Input**: the number of Fourier components $D$, step size $\eta$, threshold $\epsilon$;
**Initialize** $\mathbf{w}_1 = 0$.
Calculate $p(\mathbf{u})$ as (2). Generate random Fourier components:$\mathbf{u}_1, ..., \mathbf{u}_D$ sampled from distribution $p(\mathbf{u})$.
**for** $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t$;
  Construct new representation:
  $\mathbf{z}(\mathbf{x}_t) = (\sin(\mathbf{u}_1^\top \mathbf{x}_t), \cos(\mathbf{u}_1^\top \mathbf{x}_t), ..., \sin(\mathbf{u}_D^\top \mathbf{x}_t), \cos(\mathbf{u}_D^\top \mathbf{x}_t))^\top$
  Predict $\widehat{y}_t = \mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t)$;
  Receive $y_t$ and suffer loss $\ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$;
  **if** $\ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big) > \epsilon$ **then**
    $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$.
  **end if**
**end for**

**Algorithm 6** NOGD-R — Nyström Online Gradient Descent for Regression

**Input**: the budget $B$, step size $\eta$, rank approximation $k$, threshold $\epsilon$.
**Initialize** support vector set $\mathcal{S}_1 = \emptyset$, and model $f_1 = 0$.
**while** $|\mathcal{S}_t| < B$ **do**
  Receive new instance $\mathbf{x}_t$;
  Predict $\widehat{y}_t = f_t(\mathbf{x}_t)$;
  Update $f_t$ by regular Online Gradient Descent (OGD);
  Update $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{t\}$ whenever loss exceeds threshold;
  $t = t + 1$;
**end while**
Construct the kernel matrix $\hat{\mathbf{K}}_t$ from $\mathcal{S}_t$.
$[\mathbf{V}_k, \mathbf{D}_k] = \mathsf{eigs}(\hat{\mathbf{K}}_t, k)$, where $\mathbf{V}_k$ and $\mathbf{D}_k$ are Eigenvectors and Eigenvalues of $\hat{\mathbf{K}}_t$, respectively.
Initialize $\mathbf{w}_t^\top = [\alpha_1, ..., \alpha_B](\mathbf{D}_k^{-0.5}\mathbf{V}_k^\top)^{-1}$.
Initialize the instance index $T_0 = t$;
**for** $t = T_0, \ldots, T$ **do**
  Receive new instance $\mathbf{x}_t$;
  Construct the new representation of $\mathbf{x}_t$:
  $\mathbf{z}(\mathbf{x}_t) = \mathbf{D}_k^{-0.5}\mathbf{V}_k^\top(\kappa(\mathbf{x}_t, \hat{\mathbf{x}}_1), ..., \kappa(\mathbf{x}_t, \hat{\mathbf{x}}_B))^\top$.
  Predict $\widehat{y}_t = \mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t)$;
  Update when loss exceeds $\epsilon$: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$.
**end for**

| Dataset | # training instances | # testing instances | # features |
|---|---|---|---|
| codrna | 59,535 | 271,617 | 8 |
| w7a | 24,692 | 25,057 | 300 |
| w8a | 49,749 | 14,951 | 300 |
| a9a | 32,561 | 16,281 | 123 |
| KDDCUP08 | 81,835 | 20,459 | 117 |
| KDDCUP99 | 905,257 | 226,314 | 127 |

Table 2: Details of Binary Classification Datasets.

- "LIBSVM": one of state-of-the-art implementation for batch kernel SVM available at the LIBSVM website Chang and Lin (2011);
- "LLSVM": Low-rank Linearization SVM algorithm that transfers kernel classification to a linear problem using low-rank decomposition of the kernel matrix (Zhang et al., 2012);
- "BSGD-M": The Budgeted Stochastic Gradient Descent algorithm which extends the Pegasos algorithm (Shalev-Shwartz et al., 2011) by exploring the SV Merging strategy for budget maintenance (Wang et al., 2012b);
- "BSGD-R": The Budgeted Stochastic Gradient Descent algorithm which extends the Pegasos algorithm (Shalev-Shwartz et al., 2011) by exploring the SV Random Removal strategy for budget maintenance (Wang et al., 2012b).

To make a fair comparison of algorithms with different parameters, all the parameters, including regularization parameter ($C$ in LIBSVM, $\lambda$ in pegasos), the learning rate ($\eta$ in FOGD and NOGD) and the RBF kernel width ($\sigma$) are optimized by following a standard 5-fold cross validation on the training datasets. The budget size $B$ in NOGD and pegasos algorithms and the feature dimension $D$ in FOGD are set individually for different datasets, as indicated in the tables of experimental results. In general, these parameters are chosen such that they are roughly proportional to the size of support vectors output by the batch SVM algorithm in LIBSVM, since we would expect a relatively larger budget size for tackling more challenging classification tasks in order to achieve competitive accuracy. The rank $k$ in NOGD is set to $0.2B$ for all datasets. For the online learning algorithms, all models are trained by a single pass through the training sets and the reported accuracy and time cost are averaged over the five experiments conducted on different random permutations of the training instances. All the algorithms were implemented in C++, and conducted on a Windows machine with CPU of 3.0GHz. For the existing algorithms, all the codes can be downloaded from LIBSVM website and BudgetedSVM website [3].

#### 7.1.2 Performance Evaluation Results

Table 3 shows the experimental results of batch binary classification tasks. We can drawn several observations from the results.

3. http://www.dabi.temple.edu/budgetedsvm/algorithms.html

| Algorithm | codrna, B=400, D=1600 | | | w7a, B=400, D=1600 | | |
|---|---|---|---|---|---|---|
| | Train (s) | Test (s) | Accuracy | Train (s) | Test (s) | Accuracy |
| LIBSVM | 80.20 | 126.02 | 96.67% | 54.91 | 20.21 | 98.33% |
| LLSVM | 19.04 | 37.34 | 95.93% | 25.96 | 4.20 | 97.92% |
| BSGD-M | 132.50 | 12.77 | 94.89%± 0.41 | 37.41 | 3.91 | 97.50%± 0.02 |
| BSGD-R | 3.32 | 12.77 | 67.16%± 0.68 | 1.79 | 1.71 | 97.50%± 0.01 |
| FOGD | 5.47 | 24.49 | 94.24%± 0.22 | 1.58 | 1.46 | 97.59%± 0.28 |
| NOGD | 2.55 | 9.90 | **95.92%± 0.18** | 1.57 | 1.44 | **97.71%± 0.09** |
| Algorithm | w8a, B=1000, D=4000 | | | a9a, B=1000, D=4000 | | |
| | Train (s) | Test (s) | Accuracy | Train (s) | Test (s) | Accuracy |
| LIBSVM | 254.03 | 40.42 | 99.40% | 97.81 | 24.80 | 85.04% |
| LLSVM | 146.97 | 13.02 | 98.51% | 70.85 | 14.43 | 84.89% |
| BSGD-M | 230.51 | 2.42 | 97.52%± 0.02 | 150.16 | 3.32 | 84.21%± 0.18 |
| BSGD-R | 8.72 | 2.34 | 97.06%± 0.04 | 7.03 | 3.28 | 81.96%± 0.27 |
| FOGD | 13.04 | 3.85 | 97.93%± 0.08 | 9.35 | 4.57 | 84.93%± 0.03 |
| NOGD | 14.10 | 2.94 | **98.36%± 0.10** | 16.94 | 3.37 | **84.99%± 0.07** |
| Algorithm | KDD08, B=200, D=800 | | | KDD99, B=200, D=400 | | |
| | Train (s) | Test (s) | Accuracy | Train (s) | Test (s) | Accuracy |
| LIBSVM | 1052.12 | 124.22 | 99.43% | 20772.10 | 48.91 | 99.996% |
| LLSVM | 24.31 | 3.53 | 99.38% | 86.71 | 17.49 | 99.995% |
| BSGD-M | 197.78 | 3.07 | 99.37%± 0.01 | 948.21 | 12.11 | 99.994%± 0.001 |
| BSGD-R | 12.34 | 3.03 | 99.12%± 0.23 | 52.82 | 12.20 | 99.980%± 0.031 |
| FOGD | 17.22 | 4.33 | 98.95%± 3.34 | 26.81 | 6.80 | **99.996%± 0.001** |
| NOGD | 7.60 | 2.14 | **99.43%± 0.04** | 20.71 | 9.03 | 99.993%± 0.001 |

Table 3: Performance Evaluation Results on Batch Binary Classification Tasks.

First of all, by comparing the four online algorithms against the two batch algorithms, we found that the online algorithms in general enjoy significant advantage in terms of computational efficiency especially for large scale datasets. By further examining the learning accuracy, we found that some online algorithms, especially the proposed FOGD and NGOD algorithms, are able to achieve slightly lower but fairly competitive learning accuracy compared with the state-of-the-art batch SVM algorithm. This demonstrates that the proposed online kernel learning algorithms could be potentially a good alternative solution of the existing SVM solvers when solving large scale batch kernel classification tasks in real-world applications due to their significant advantage of much lower learning time and memory costs.

Further, by comparing the four different online algorithms, we found that, in terms of learning accuracy, despite running faster, the BSGD-R ("pegasos+remove") algorithm suffers from very high mistake rate in most of the datasets. This is due to its naive budget maintenance strategy that simply discards the oldest support vector that may contain important information. While for BSGD-M ("pegasos+merging") algorithm, the main drawback is its relatively high computational cost. This can be easily observed in some datasets

(e.g., a9a, w7a and codrna), in which the difference between the number of support vectors of LIBSVM and the budget size is relatively larger than that of the other datasets. Thus, we can conclude that the high time cost of the BSGD-M("pegasos+merging") is due to the complex computation in the merging steps. Compared with the other online learning algorithms, the proposed NOGD algorithm achieves the highest accuracy for most cases while spending almost the lowest learning time cost. Similarly, FOGD algorithm also obtains more accurate result than the two budget Pegasos algorithms on most of the datasets with comparable or sometimes ever lower learning time cost. These facts indicate that the two proposed budget online kernel learning algorithms are both efficient and effective in solving large scale kernel classification problems.

Finally, by comparing the two proposed algorithms, we found that the performance of NOGD is better than that of FOGD. This reflects that the Nyström kernel approximation tends to have a better approximation of the original RBF kernel than the Fourier feature based approximation.

### 7.2 Experiments for Online Binary Classification Tasks

In this section, we test the performance of our proposed algorithms on the online binary classification task.

#### 7.2.1 Experimental Test beds and Setup

Table 4 shows the details of 9 publicly available datasets of diverse sizes for online binary classification tasks. All of them can be downloaded from LIBSVM website, UCI machine learning repository [4] and KDDCUP competition site. As a yardstick for evaluation, we

| Dataset | # instances | # features |
|---|---|---|
| german | 1,000 | 24 |
| spambase | 4,601 | 57 |
| w7a | 24,692 | 300 |
| w8a | 64,700 | 300 |
| a9a | 48,842 | 123 |
| KDDCUP08 | 102,294 | 117 |
| ijcnn1 | 141,691 | 22 |
| codrna | 271,617 | 8 |
| KDDCUP99 | 1,131,571 | 127 |

Table 4: Details of Online Binary Classification Datasets.

include the following two popular algorithms for regular online kernel classification without concerning budget:

- "Perceptron": the kernelized Perceptron (Freund and Schapire, 1999) without budget;
- "OGD": the kernelized online gradient descent (Kivinen et al., 2001) without budget.

4. http://www.ics.uci.edu/~mlearn/

Further, we compare the proposed budget online kernel learning algorithms with the following state-of-the-art budget online kernel learning algorithms:

- "RBP": the random budget perceptron by random removal strategy (Cavallanti et al., 2007);
- "Forgetron": the Forgetron by discarding oldest support vectors (Dekel et al., 2005);
- "Projectron": the Projectron algorithm using the projection strategy (Orabona et al., 2009);
- "Projectron++": the aggressive version of Projectron algorithm (Orabona et al., 2008, 2009);
- "BPA-S": the Budget Passive-Aggressive algorithm with simple SV removal strategy in (Wang and Vucetic, 2010);
- "BOGD": the Budget Online Gradient Descent algorithm by SV removal strategy (Zhao et al., 2012);

To make fair comparisons, all the algorithms follow the same setups. We adopt the hinge loss as the loss function $\ell$. Note that hinge loss is not a smooth function, whose gradient is undefined at the point that the classification confidence $yf(\mathbf{x}) = 1$. Following the subgradient definition, in our experiment, gradient is only computed under the condition that $yf(\mathbf{x}) < 1$, and set to 0 otherwise. The Gaussian kernel bandwidth is set to 8. The step size $\eta$ in the all online gradient descent based algorithms is chosen through a random search in range {2, 0.2, ..., 0.0002}. We adopt the same budget size $B = 100$ for NOGD and other budget algorithms. In the setting of FOGD algorithm, $D = \rho_f B$, where $0 < \rho_f < \infty$ is a predefined parameter that controls the number of random Fourier components. For NOGD algorithm, $k = \rho_n B$, where $0 < \rho_n < 1$ is a predefined parameter that controls the accuracy of matrix approximation. We set $\rho_f = 4$ and $\rho_n = 0.2$ and will evaluate their influence on the algorithm performance in the following discussion. For each data set, all the experiments were repeated 20 times using different random permutation of instances in the dataset. All the results were obtained by averaging over these 20 runs. For performance metrics, we evaluate the online classification performance by standard mistake rates and running time (seconds). All algorithms are implemented in Matlab R2013b, on a Windows machine with 3.0 GHZ CPU,6 cores.

#### 7.2.2 Performance Evaluation Results

Table 5 summarizes the empirical evaluation results on the nine diverse data sets. From the results, we can draw the following observations.

First of all, in terms of time efficiency, we found that the budget online classification algorithms in general run much faster than the regular online kernel classification algorithms (Perceptron and OGD) especially on the large datasets, validating the importance of studying scalable online kernel methods. By further examining their results of mistake rates, we found that the budget online classification algorithms are generally worse than the two non-budget algorithms, validating the motivation of exploring effective techniques for budget online kernel classification.

| Algorithm | german | | spambase | | w7a | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| Perceptron | 35.2 %± 0.9 | 0.112 | 24.5 %± 0.1 | 1.606 | 4.01 %± 0.10 | 74.0 |
| OGD | 29.5 %± 0.5 | 0.130 | 22.0 %± 0.1 | 4.444 | 2.96 %± 0.10 | 119.9 |
| RBP | 37.5 %± 1.1 | 0.086 | 33.3 %± 0.4 | 0.613 | 5.07 %± 0.13 | 11.20 |
| Forgetron | 38.1 %± 0.9 | 0.105 | 34.6 %± 0.5 | 0.743 | 5.28 %± 0.06 | 11.77 |
| Projectron | 35.6 %± 1.5 | 0.101 | 30.8 %± 1.2 | 0.644 | 5.38 %± 1.15 | 11.22 |
| Projectron++ | 35.1 %± 1.1 | 0.299 | 30.4 %± 1.0 | 1.865 | 4.79 %± 1.87 | 13.43 |
| BPA-S | 33.9 %± 0.9 | 0.092 | 30.8 %± 0.8 | 0.604 | 2.99 %± 0.06 | 11.60 |
| BOGD | 31.6 %± 1.5 | 0.114 | 32.2 %± 0.6 | 0.720 | 3.49 %± 0.16 | 11.56 |
| FOGD | **29.9 %± 0.7** | **0.045** | **26.9 %± 1.0** | **0.263** | **2.75 %± 0.03** | **1.474** |
| NOGD | 30.4 %± 0.8 | 0.109 | 29.1 %± 0.4 | 0.633 | 2.98 %± 0.01 | 11.58 |

| Algorithm | w8a | | a9a | | ijcnn1 | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| Perceptron | 3.47 %± 0.01 | 642.8 | 20.9 %± 0.1 | 948.7 | 12.27 %± 0.01 | 812.6 |
| OGD | 2.81 %± 0.01 | 1008.5 | 16.3 %± 0.1 | 1549.5 | 9.52 %± 0.01 | 1269.0 |
| RBP | 5.10 %± 0.08 | 37.8 | 27.1 %± 0.2 | 15.4 | 16.40 %± 0.10 | 18.5 |
| Forgetron | 5.28 %± 0.07 | 40.0 | 27.8 %± 0.4 | 19.3 | 16.99 %± 0.32 | 21.2 |
| Projectron | 5.42 %± 1.10 | 38.1 | 21.6 %± 1.9 | 15.3 | 12.38 %± 0.09 | 19.2 |
| Projectron++ | 5.41 %± 3.30 | 38.7 | 18.6 %± 0.5 | 23.4 | 9.52 %± 0.03 | 30.3 |
| BPA-S | 2.84 %± 0.03 | 39.2 | 21.1 %± 0.2 | 15.4 | 11.33 %± 0.04 | 18.3 |
| BOGD | 3.43 %± 0.08 | 38.9 | 27.9 %± 0.2 | 15.9 | 11.67 %± 0.13 | 19.2 |
| FOGD | **2.43 %± 0.03** | **3.0** | **17.4 %± 0.1** | **1.8** | **9.06 %± 0.05** | **3.3** |
| NOGD | 2.92 %± 0.03 | 38.9 | **17.4 %± 0.2** | 15.6 | 9.55 %± 0.01 | 19.1 |

| Algorithm | codrna | | KDDCUP08 | | KDDCUP99 | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| Perceptron | 14.0 %± 0.1 | 1015.7 | 0.90 %± 0.01 | 72.6 | 0.02 %± 0.00 | 1136 |
| OGD | 9.7 %± 0.1 | 1676.7 | 0.52 %± 0.01 | 421.7 | 0.01 %± 0.00 | 8281 |
| RBP | 20.3 %± 0.1 | 24.9 | 1.06 %± 0.03 | 34.3 | 0.02 %± 0.00 | 682 |
| Forgetron | 19.9 %± 0.1 | 28.9 | 1.07 %± 0.03 | 34.8 | 0.03 %± 0.00 | 684 |
| Projectron | 15.8 %± 0.5 | 26.0 | 0.94 %± 0.02 | 34.1 | 0.02 %± 0.00 | 642 |
| Projectron++ | 13.6 %± 1.2 | 83.0 | 0.84 %± 0.03 | 77.2 | **0.01 %± 0.00** | 520 |
| BPA-S | 15.4 %± 0.3 | 26.5 | 0.62 %± 0.01 | 37.8 | **0.01 %± 0.00** | 796 |
| BOGD | 15.2 %± 0.1 | 32.5 | 0.61 %± 0.01 | 38.2 | 0.81 %± 0.06 | 805 |
| FOGD | **10.3 %± 0.1** | **10.3** | 0.71 %± 0.01 | **4.1** | **0.01 %± 0.00** | **45** |
| NOGD | 13.8 %± 2.1 | 27.2 | **0.59 %± 0.01** | 38.9 | **0.01 %± 0.00** | 511 |

Table 5: Evaluation of Large-scale Online Kernel Learning on Binary Classification Task .

Second, by comparing the proposed algorithms (FOGD and NOGD) with the budget online classification algorithms, we found that they generally achieve the best classification performance for most cases using fairly comparable or even lower time cost. While other algorithms, are either too slow because of their extremely complex updating methods or of low accuracy because of their simply SV removal steps. Similarly to the batch setting, this demonstrates the effectiveness and efficiency of the proposed algorithms.

Third, it might seem surprising to find that the FOGD algorithm achieves extremely low mistake rate and even outperforms the OGD algorithm in some datasets ($w7a, w8a, ijcnn1$). Ideally, FOGD should perform nearly the same as the kernel-based OGD approach if the number of Fourier components $D$ is extremely large. However, choosing a too large value of $D$ will result in underfitting for a relatively small data set, meanwhile choosing a too small value of $D$ will result in overfitting. In our experiments, we choose an appropriate value of $D$ ($D = 4B$) , which not only could save computational cost, but also may prevent both underfitting and overfitting. In contrast, the kernel OGD always add a new support vector whenever the loss is nonzero. Thus, the predicted model learned by the kernel OGD will become more and more complicated as time goes, and thus would likely suffer from overfitting for noisy examples.

Finally, we note that there are several differences in this result compared with the previous section in batch setting. To begin with, FOGD achieves extremely low time cost in all datasets. While in batch setting using C++ implementation, its time cost is comparable with that of NOGD. This can be explained by the different settings of the two implementation methods. In C++ setting, the most time consuming step in FOGD is to compute the large number of random features, while in Matlab setting, it is automatic transformed to a matrix calculation and parallelized on all cores of CPU. In addition, FOGD tends to performance better than NOGD in terms of accuracy. This is the result of different budget size. For NOGD, it is difficult to approximate the whole kernel matrix with small number of support vectors (such as the setting in this section $B = 100$). But with larger budget size, as in the batch case, the approximation accuracy is better than that of FOGD.

### 7.3 Experiments for Multi-class Classification Tasks

This section tests the performance of our proposed algorithms on online multi-class classification task.

#### 7.3.1 Experimental Test beds and Setup

In this section, we evaluate the multi-class versions of FOGD and NOGD algorithms on 9 real-world datasets for multi-class classification tasks from the LIBSVM website. Table 6 summarizes the details of these datasets.

We adopt the same set of compared algorithms and similar parameter settings in multi-class task as that of binary case. Larger the budget size parameter $B$ is used for multiclass classification than binary case since we should ensure enough support vectors for each class label. We set $B = 200$ for the first 3 datasets and $B = 100$ for the last 6 large scale datasets. For time efficiency, we omit the experiments of non-budget algorithms on extremely large datasets.

| Dataset | # instances | # features | # classes |
|---|---|---|---|
| dna | 2,000 | 180 | 3 |
| satimage | 4,435 | 36 | 6 |
| usps | 7,291 | 256 | 10 |
| mnist | 10,000 | 780 | 10 |
| letter | 15,000 | 16 | 26 |
| shuttle | 43,500 | 9 | 7 |
| acoustic | 78,823 | 50 | 3 |
| covtype | 581,012 | 54 | 7 |
| poker | 1,000,000 | 10 | 10 |

Table 6: Details of Multi-class Classification Datasets.

#### 7.3.2 Performance Evaluation Results

Table 7 summarizes the average performance evaluation results for the compared algorithms on multi-class classification task. To further inspect more details of online multi-class classification performance, Figure 1 and Figure 2 also show the online performance convergence of all the compared algorithms in the entire online learning process. From these results, we can draw some observations as follows.

First of all, similar to the binary case, budget online kernel learning algorithms are much more efficient than the regular online kernel learning algorithms without budget, which is more obvious for larger scale datasets. For the three largest datasets (*acoustic*, *covtype* and *poker*), some of which consists of nearly one million instances, we have to exclude the non-budget online learning algorithms due to their extremely expensive costs in both time and memory. This again demonstrates the importance of exploring budget online kernel learning algorithms. Among the two non-budget online kernel learning algorithms, we found that OGD often achieves the highest accuracy, which is much better than Perceptron. However, its high-accuracy performance is paid by spending significantly higher computational time cost in comparison to the Perceptron algorithm. This is because OGD performs much more aggressive updates than Perceptron in the online learning process, which thus results in a significantly larger number of support vectors.

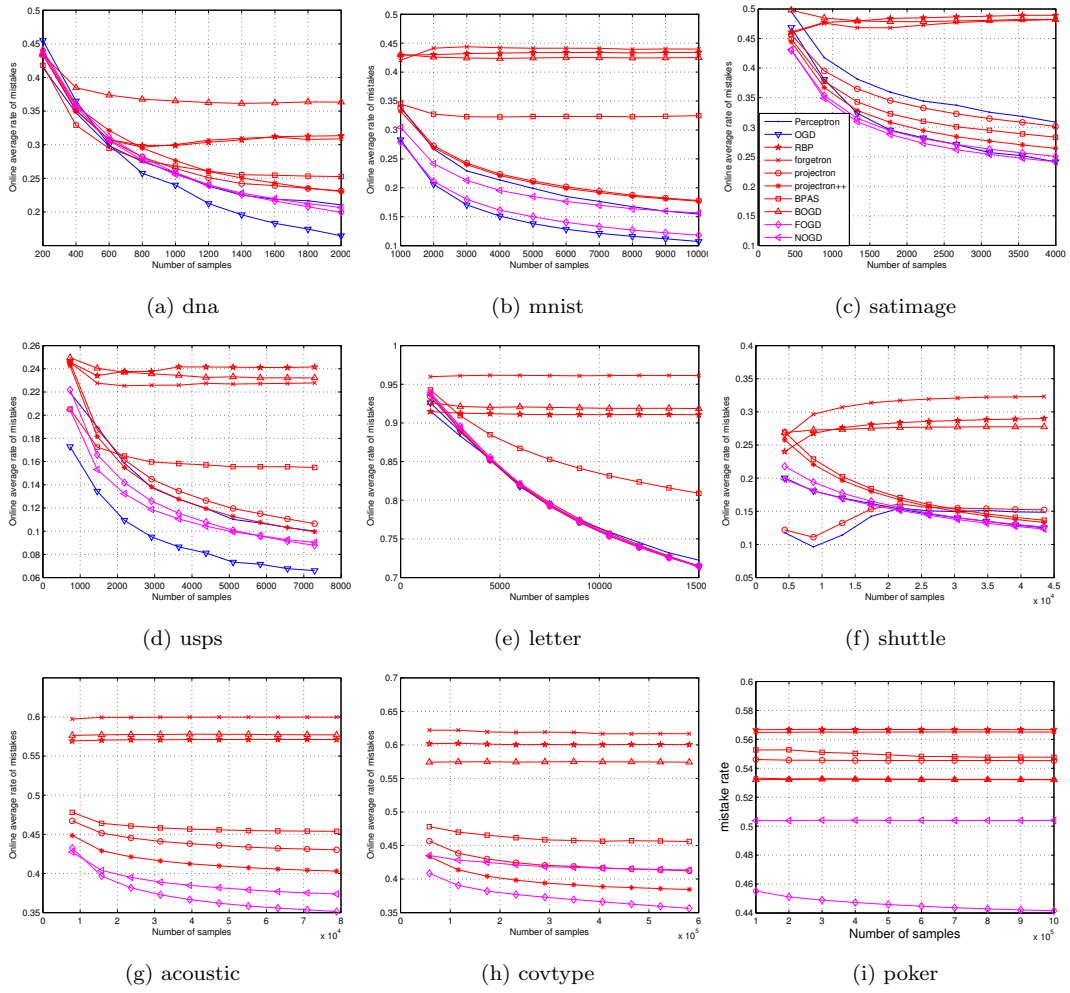
Second, when comparing the performance of different existing budget online kernel learning algorithms, it is clear to observe that the algorithms based on support vector projection strategy (projectron and projectron++) achieve significantly higher accuracy than the algorithms using simple support vector removal strategy. However, the gain of accuracy is paid by the sacrifice of efficiency, as shown by the time cost results in the table. Furthermore, one might be surprised to observe that BPA-S, which is relatively efficient in binary case, is extremely slow in multi-class case. This is due to the different updating approach of BPA-S for multi-class classification. In particular, for other budget multi-class algorithms, their time complexity of each prediction is $O(2B)$, i.e., only 2 out of the $m$ classes ($y$ and $s$) are updated when adding a new support vector. By contrast, during the update of BPA-S at each iteration, every class has to be updated, leading to the overall time complexity of $O(mB)$. Consequently, the BPA-S is much more expensive than the other algorithms.

| Algorithm | dna | | mnist | | satimage | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| Perceptron | 20.4 %±0.7 | 4.891 | 15.4 %±0.1 | 456.76 | 29.6 %±0.5 | 4.675 |
| OGD | 16.1 %±0.4 | 25.068 | 10.7 %±0.2 | 1004.65 | 23.6 %±0.3 | 6.917 |
| RBP | 31.1 %±1.4 | 2.707 | 43.4 %±0.5 | 59.98 | 49.3 %±0.8 | 2.409 |
| Forgetron | 30.9 %±2.1 | 2.949 | 43.9 %±0.6 | 64.32 | 48.2 %±1.4 | 2.503 |
| Projectron | 22.7 %±4.4 | 4.254 | 17.7 %±0.1 | 434.53 | 29.6 %±0.5 | 3.685 |
| Projectron++ | 23.1 %±6.1 | 4.330 | 17.7 %±0.3 | 430.38 | 25.9 %±0.4 | 3.771 |
| BPA-S | 25.3 %±1.2 | 11.055 | 32.4 %±1.2 | 91.25 | 27.9 %±0.3 | 17.337 |
| BOGD | 36.3 %±0.9 | 3.103 | 42.5 %±0.4 | 62.51 | 48.2 %±0.6 | 2.670 |
| FOGD | 20.8 %±0.7 | **0.887** | **11.8 %±0.2** | **1.792** | 29.5 %±0.4 | **0.915** |
| NOGD | **20.7 %±0.9** | 2.292 | 15.6 %±0.6 | 46.90 | **23.7 %±0.3** | 1.869 |

| Algorithm | usps | | letter | | shuttle | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| Perceptron | 10.0 %±0.2 | 89.790 | 71.5 %±0.5 | 80.901 | 15.2 %±0.3 | 137.886 |
| OGD | 6.7 %±0.2 | 310.072 | 71.2 %±0.3 | 94.222 | 12.3 %±0.1 | 377.328 |
| RBP | 24.0 %±0.4 | 30.180 | 91.7 %±0.2 | 18.783 | 29.3 %±0.5 | 12.989 |
| Forgetron | 22.7 %±0.5 | 30.478 | 96.1 %±0.1 | 19.034 | 34.1 %±0.4 | 12.776 |
| Projectron | 10.6 %±0.1 | 192.347 | 71.5 %±0.5 | 26.921 | 15.3 %±0.3 | 20.034 |
| Projectron++ | 9.9 %±0.2 | 194.366 | **71.4 %±0.3** | 27.584 | 16.8 %±0.5 | 20.879 |
| BPA-S | 15.4 %±0.6 | 54.457 | 84.6 %±0.5 | 47.459 | 14.1 %±0.2 | 58.856 |
| BOGD | 23.2 %±0.4 | 30.966 | 92.7 %±0.2 | 18.510 | 27.9 %±0.2 | 14.399 |
| FOGD | 10.1 %±0.2 | **9.589** | 71.5 %±0.6 | **2.322** | 15.6 %±0.5 | **4.039** |
| NOGD | **9.0 %±0.2** | 24.332 | 71.5 %±0.2 | 3.380 | **12.3 %±0.1** | 8.484 |

| Algorithm | acoustic | | covtype | | poker | |
|---|---|---|---|---|---|---|
| | Mistake Rate | Time(s) | Mistake Rate | Time(s) | Mistake Rate | Time(s) |
| RBP | 57.4 %±0.2 | 40.469 | 60.1 %±0.1 | 445.238 | 56.6 %±0.0 | 398.423 |
| Forgetron | 61.2 %±0.4 | 46.865 | 60.4 %±0.4 | 491.403 | 56.5 %±0.0 | 413.807 |
| Projectron | 43.0 %±0.1 | 46.469 | 41.1 %±0.2 | 930.646 | 54.5 %±0.1 | 810.421 |
| Projectron++ | 40.3 %±0.1 | 47.400 | 38.3 %±0.2 | 937.860 | 53.2 %±0.1 | 825.526 |
| BPA-S | 46.2 %±0.3 | 210.916 | 45.2 %±0.2 | 1569.255 | 54.7 %±0.4 | 2566.812 |
| BOGD | 58.0 %±0.1 | 40.266 | 57.4 %±0.0 | 456.692 | 53.2 %±0.0 | 465.020 |
| FOGD | 43.0 %±0.2 | **12.637** | **40.4 %±0.1** | **80.774** | 52.6 %±0.1 | **190.102** |
| NOGD | **37.8 %±0.1** | 25.883 | 41.0 %±0.6 | 211.354 | **50.3 %±0.2** | 216.717 |

Table 7: Evaluation of Large-scale Online Kernel Learning on Multi-class Classification Task .



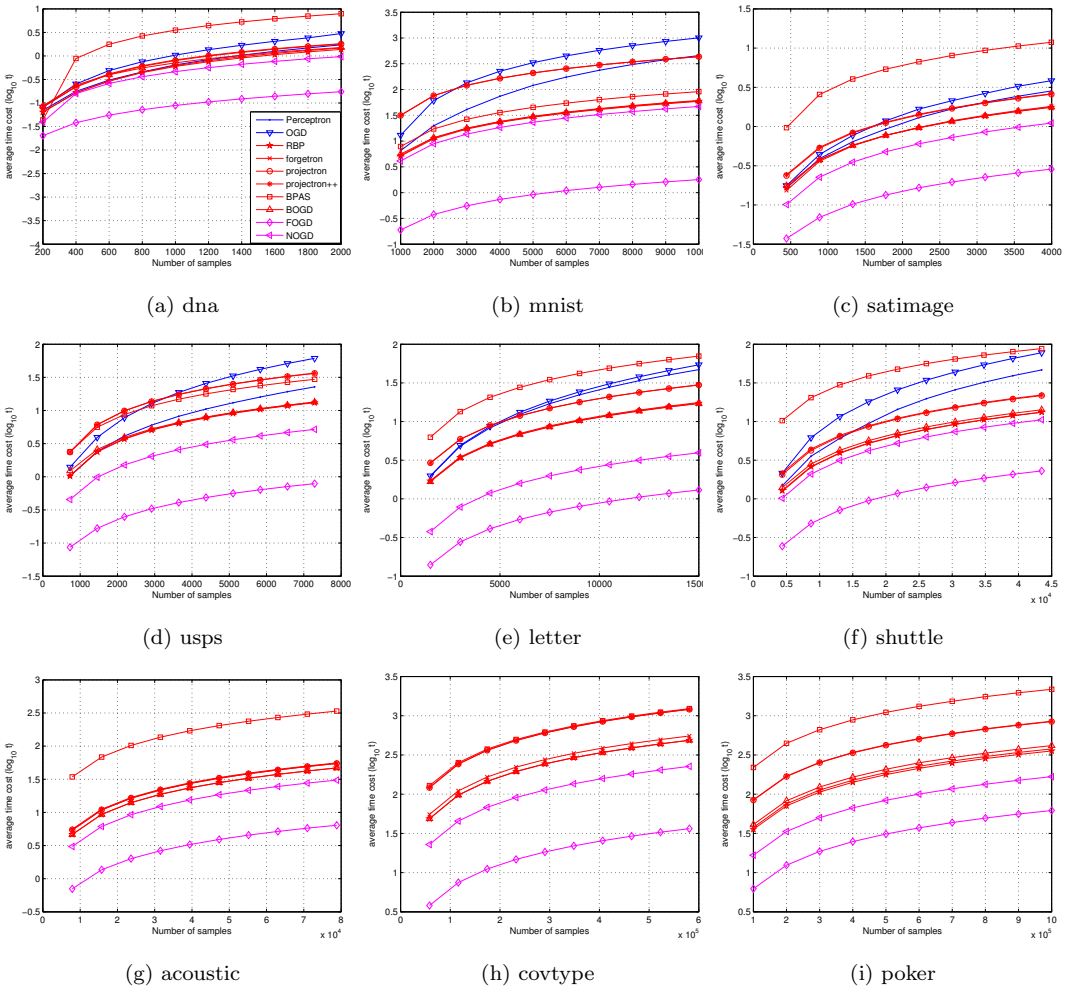

(a) dna (b) mnist (c) satimage

(d) usps (e) letter (f) shuttle

(g) acoustic (h) covtype (i) poker

Figure 1: Convergence evaluation of multi-class datasets: mistake rate (best viewed in color).

Furthermore, by comparing the two proposed algorithms, FOGD and NOGD, with the existing budget online kernel learning algorithms, we observe that the proposed algorithms achieve the highest accuracy for most cases, and meanwhile run significantly faster than the other algorithms, which again validates the effectiveness and efficiency of our proposed technique. Thus, we can conclude that the proposed functional approximation approach for budget online kernel learning is a promising technique for building scalable online kernel learning algorithms for large scale learning tasks. Finally, by comparing FOGD and NOGD, we found that their accuracy performance is nearly comparable while FOGD is relatively faster. As mentioned in the binary section, this indicates that FOGD is easier for parallelization.




(a) dna (b) mnist (c) satimage

(d) usps (e) letter (f) shuttle

(g) acoustic (h) covtype (i) poker

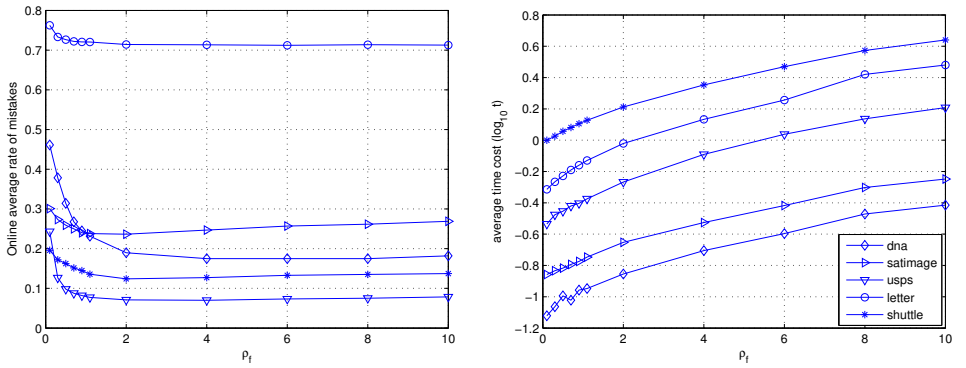
Figure 2: Convergence evaluation of multi-class datasets: time cost (best viewed in color).

### 7.3.3 Evaluation for the $\rho_f$ and $\rho_n$ on Multi-class Tasks

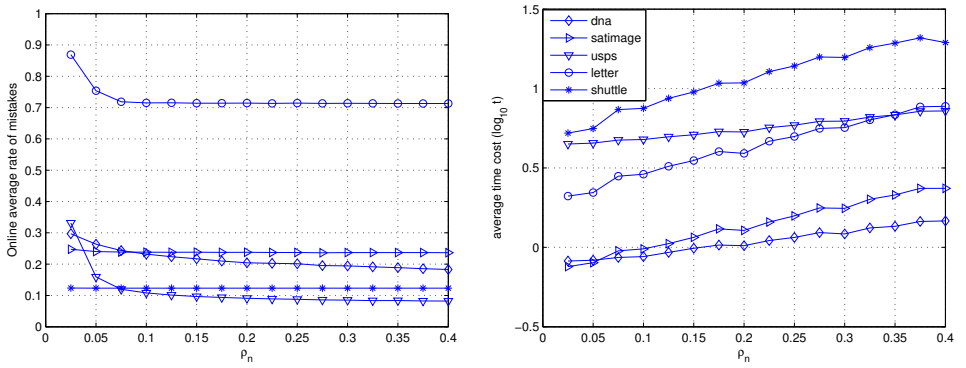
As mentioned in the previous experiments, we set the parameter for the number of Fourier components $D = \rho_f \times B$ and the rank of Nyström matrix approximation $k = \rho_n B$, where different choices of parameters $\rho_f$ and $\rho_n$ could affect the resulting performance of FOGD and NOGD, respectively. In this section, we evaluate the sensitivity of these two parameters and examine their influence to both learning accuracy and time cost of multi-class classification tasks.

Specifically, we fix the budget size $B$ to 200 for all datasets, and set the other parameters (except $B$, $\rho_f$, and $\rho_n$) by following the same settings as the previous multi-class classification tasks. Figure 3 summarizes the performance evaluation results, including average mistake rates and average time costs. From the results, we can draw some observations as follows.

First of all, we observe that increasing the value of $\rho_f$ or $\rho_n$ leads to better classification accuracy but higher running time cost. This is not difficult to understand since increasing




(a) The effect of $\rho_f$ on the mistake rate of FOGD (b) The effect of $\rho_f$ on the time cost of FOGD

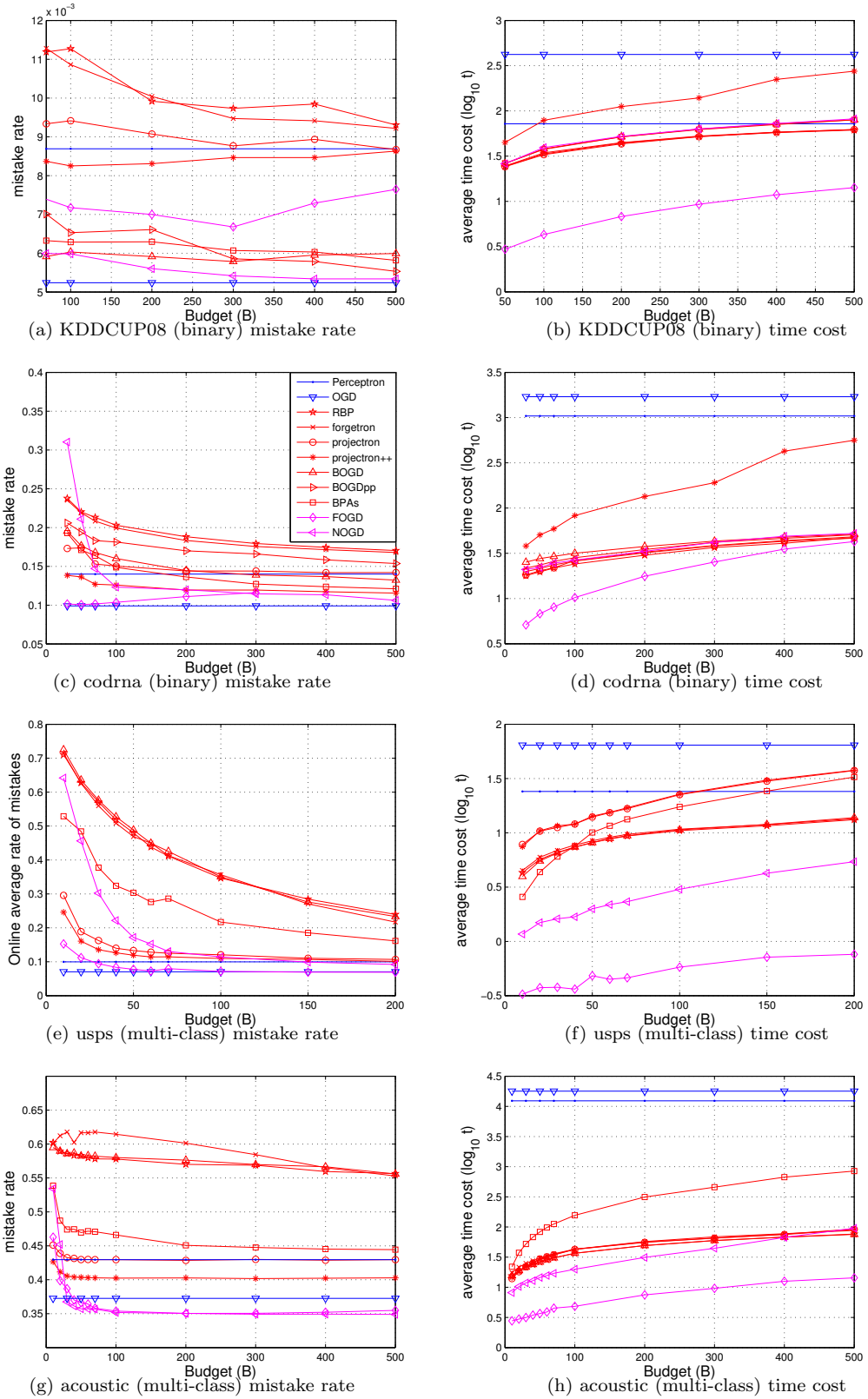



(c) The effect of $\rho_n$ on the mistake rate of NOGD (d) The effect of $\rho_n$ on the time cost of NOGD

Figure 3: Performance evaluation on different values of $\rho_f$ and $\rho_n$.

the value $\rho_f$ is essential to increasing the number of Fourier components, leading to a better approximation of lower variance and thus higher classification accuracy. Meanwhile the computational time cost of FOGD is proportional to the number of Fourier components, and thus is proportional to the value of $\rho_f$. Similarly, for NOGD, the large the value of $\rho_n$, the more accurate approximation achieved by the Nystrom kernel matrix approximation, and meanwhile the more computational cost required. Thus, the choice of parameter $\rho_f$ or $\rho_n$ for FOGD or NOGD is essentially a trade off between learning effectiveness and computational efficiency.

Second, we found there is some common tendency of the impact on the learning accuracy by the two parameters, although different datasets may have slightly different results. In particular, we observe that when the value of $\rho_f$ or $\rho_n$ is large enough (e.g., $\rho_f > 5$ or $\rho_n > 0.1$), increasing their value has limited impact on the improvement of the learning accuracy while the time cost keeps growing linearly. This gives an important guideline for one to choose the two parameters properly in order to gain computational efficiency without sacrificing learning accuracy. Specifically, as shown in the figure, by choosing the two parameters roughly in the ranges of $\rho_f \in (4, 6)$ and $\rho_n \in (0.2, 0.4)$, we are able to achieve satisfactory tradeoff for most cases.




(a) KDDCUP08 (binary) mistake rate

(b) KDDCUP08 (binary) time cost

(c) codrna (binary) mistake rate

(d) codrna (binary) time cost

(e) usps (multi-class) mistake rate

(f) usps (multi-class) time cost

(g) acoustic (multi-class) mistake rate

(h) acoustic (multi-class) time cost

Figure 4: The effect of different budget sizes (B) (best viewed in color).

#### 7.3.4 Evaluation for the Selection of Budget Size on Multi-class Classification Task

For all the budget online kernel learning algorithms, the choice of budget size parameter $B$ can have a considerable impact on the resulting performance. In our previous experiments, we simply fix the budget size parameter $B$ to some constants for the compared budget online kernel learning algorithms to enable a fair and simplified comparison. In this section, we aim to evaluate the sensitivity of the budget size parameter $B$ and examine if the proposed algorithms are consistently better than the other budget online kernel learning algorithms under varied settings of the budget size parameter. Specifically, in this experiment, we follow the same experimental setups as the previous experiments, except that we evaluate the influence of varied values of budget size parameter $B$.

Figure 4 shows the experimental results of both average mistake rates and average time costs of different algorithms during the online learning processes under different values of the budget size parameter $B$ on four randomly chosen datasets, including two binary classification datasets and two multi-class classification datasets. From the experimental results, we can draw several observations on the impact of the budget size parameter to the performances as follows.

First of all, we observe that increasing the budget size generally results in better classification accuracy and higher learning time cost for all the budget online kernel learning algorithms. This is not difficult to understand since a larger budget size potentially leads to a more precise approximation to their non-budget original algorithm, and thus a better prediction accuracy. Second, similar to the previous experiments, we notice that when the budget size is large enough, further increasing the budget size has limited gain on the improvement of classification accuracy. This observation indicates that selecting a proper budget size parameter $B$ is a tradeoff between classification accuracy and learning time cost. Moreover, by comparing different budget learning algorithms under varied values of $B$, we found that the projectron algorithms and the proposed two algorithms (FOGD and NOGD) tend to achieve the best classification accuracy results for most cases, particularly when the value of budget size $B$ is small. By further examining the time costs, we found that the proposed algorithms (especially FOGD) are significantly more efficient than the Projectron for varied values of $B$. These encouraging results again validate that the proposed algorithms not only achieve consistently better accuracy results than the existing budget online kernel learning methods for most cases, but also have a significant advantage in computational efficiency for large-scale online kernel learning tasks.

### 7.4 Experiments for Online Regression Tasks

This section tests the performance of our proposed algorithms on online regression tasks.

#### 7.4.1 Experimental Test beds and Setup

Table 8 summarizes the details of the 9 datasets of diverse sizes in our online regression experiments. All of them are publicly available at the LIBSVM and UCI websites.

For comparison schemes, we compare the proposed FOGD-R and NOGD-R algorithms with three non-budget online regression algorithms including OGD, Perceptron, and Norma

| Dataset | # instances | # features |
|---|---|---|
| housing | 506 | 13 |
| mg | 1,385 | 6 |
| abalone | 4,177 | 8 |
| parkinsons | 5,875 | 20 |
| cpusmall | 8,192 | 12 |
| cadata | 20,640 | 8 |
| casp | 45,730 | 9 |
| slice | 53,500 | 385 |
| year | 463,715 | 90 |

Table 8: Details of Regression Datasets.

(Kivinen et al., 2001), and four other existing budget online kernel learning algorithms including RBP, Forgetron, Projectron, and BOGD.

For parameter setting, we follow the same setup as the previous experiments for most of the parameters. For the Norma algorithm, the adaptable threshold parameter $\epsilon$ is learned and updated at each iteration. For all the other algorithms, this parameter $\epsilon$ is simply fixed to 0.1. We set $\rho_f = 15$ and $B = 30$ for all the regression datasets. According to our empirical experience on online regression tasks, the regular perceptron based algorithms that simply use the default step size 1 would perform extremely poor because of the inappropriate learning rate. In order to have a stronger baseline for comparison, we conduct a validation experiment by searching for the best learning rate parameter (about 0.1) for all the perceptron-based algorithms.

### 7.4.2 Performance Evaluation Results

Table 9 shows the summary of empirical evaluation results on the nine datasets, and Figures 6 and 6 show the detailed regressions results in terms of both regression errors and time cost in the online learning processes. From these results, we can draw several observations as follows.

First of all, by examining the running time costs of different algorithms, it is clear to see that the budget online kernel learning algorithms are more efficient than the non-budget algorithms, particularly on larger scale datasets. This observation is consistent to the previous classification experiments, again validating the importance of studying budget online kernel learning methods. By examining the non-budget algorithms, we found that NORMA runs faster than the other two algorithms (OGD and Perceptron) which is primarily because of it the adaptive threshold which reduces the frequency of update and thus obtains a relatively smaller support vector set size. Among all the budget algorithms, the proposed FOGD algorithm is able to achieve the smallest time cost for all cases.

Second, in terms of regression accuracy, among the existing budget algorithms, the projectron algorithm outperforms the other existing budget online learning algorithms due to its sophisticated projection strategy. By further comparing the proposed FOGD and NOGD algorithms with the existing ones, we found that our algorithms achieve the lowest squared

| Algorithm | housing | | mg | | abalone | |
|---|---|---|---|---|---|---|
| | Squared Loss | Time | Squared Loss | Time | Squared Loss | Time |
| OGD | 0.04017±0.00043 | 0.028 | 0.05341±0.00071 | 0.103 | 0.01137±0.00007 | 0.388 |
| Perceptron | 0.04018±0.00080 | 0.029 | 0.05682±0.00084 | 0.103 | 0.01280±0.00010 | 0.388 |
| Norma | 0.04329±0.00065 | 0.028 | 0.06446±0.00073 | 0.086 | 0.01224±0.00006 | 0.448 |
| RBP | 0.05837±0.00140 | 0.028 | 0.09652±0.00253 | 0.075 | 0.02498±0.00034 | 0.200 |
| Forgetron | 0.05848±0.00216 | 0.037 | 0.09742±0.00334 | 0.106 | 0.02483±0.00042 | 0.269 |
| Projectron | 0.04023±0.00080 | 0.027 | 0.05683±0.00084 | 0.070 | 0.01280±0.00010 | 0.183 |
| BOGD | 0.05270±0.00134 | 0.024 | 0.08936±0.00198 | 0.064 | 0.01558±0.00017 | 0.175 |
| FOGD | **0.04009±0.00071** | **0.016** | 0.05590±0.00073 | **0.037** | 0.01169±0.00005 | **0.104** |
| NOGD | 0.04063±0.00043 | 0.031 | **0.05356±0.00076** | 0.073 | **0.01138±0.00007** | 0.202 |

| Algorithm | parkinsons | | cpusmall | | cadata | |
|---|---|---|---|---|---|---|
| | Squared Loss | Time | Squared Loss | Time | Squared Loss | Time |
| OGD | 0.04835±0.00018 | 2.025 | 0.02508±0.00009 | 1.905 | 0.03976 ±0.00018 | 11.63 |
| Perceptron | 0.05306±0.00045 | 2.116 | 0.02660±0.00015 | 1.257 | 0.04155±0.00019 | 11.50 |
| Norma | 0.05084±0.00018 | 1.385 | 0.03403±0.00014 | 2.060 | 0.05739±0.00008 | 8.45 |
| RBP | 0.07540±0.00102 | 0.349 | 0.04895±0.00058 | 0.449 | 0.08115±0.00029 | 1.09 |
| Forgetron | 0.07488±0.00114 | 0.496 | 0.04905±0.00062 | 0.581 | 0.08128±0.00061 | 1.54 |
| Projectron | 0.05306±0.00046 | 0.320 | 0.02660±0.00015 | 0.375 | 0.04155±0.00020 | 1.00 |
| BOGD | 0.06159±0.00037 | 0.295 | 0.04972±0.00048 | 0.406 | 0.07259±0.00031 | 0.94 |
| FOGD | 0.04909±0.00020 | **0.187** | 0.02577±0.00050 | **0.217** | 0.04097±0.00015 | **0.55** |
| NOGD | **0.04896±0.00068** | 0.336 | **0.02559±0.00024** | 0.427 | **0.03983±0.00018** | 1.05 |

| Algorithm | casp | | slice | | year | |
|---|---|---|---|---|---|---|
| | Squared Loss | Time | Squared Loss | Time | Squared Loss | Time |
| RBP | 0.12425±0.00048 | 2.56 | 0.04799±0.00025 | 22.13 | 0.03151±0.00007 | 89.7 |
| Forgetron | 0.12455±0.00046 | 3.76 | 0.04843±0.00024 | 35.43 | 0.03148±0.00005 | 139.7 |
| Projectron | 0.08709±0.00021 | 2.40 | 0.01493±0.00142 | 21.84 | 0.01627±0.00013 | 87.1 |
| BOGD | 0.09683±0.00012 | 2.23 | 0.04730±0.00011 | 21.61 | 0.05430±0.00002 | 88.3 |
| FOGD | 0.08021±0.00031 | **1.37** | **0.00726±0.00019** | **4.65** | **0.01427±0.00004** | **19.3** |
| NOGD | **0.07844±0.00008** | 2.51 | 0.02636±0.00460 | 22.05 | 0.01519±0.00021 | 89.1 |

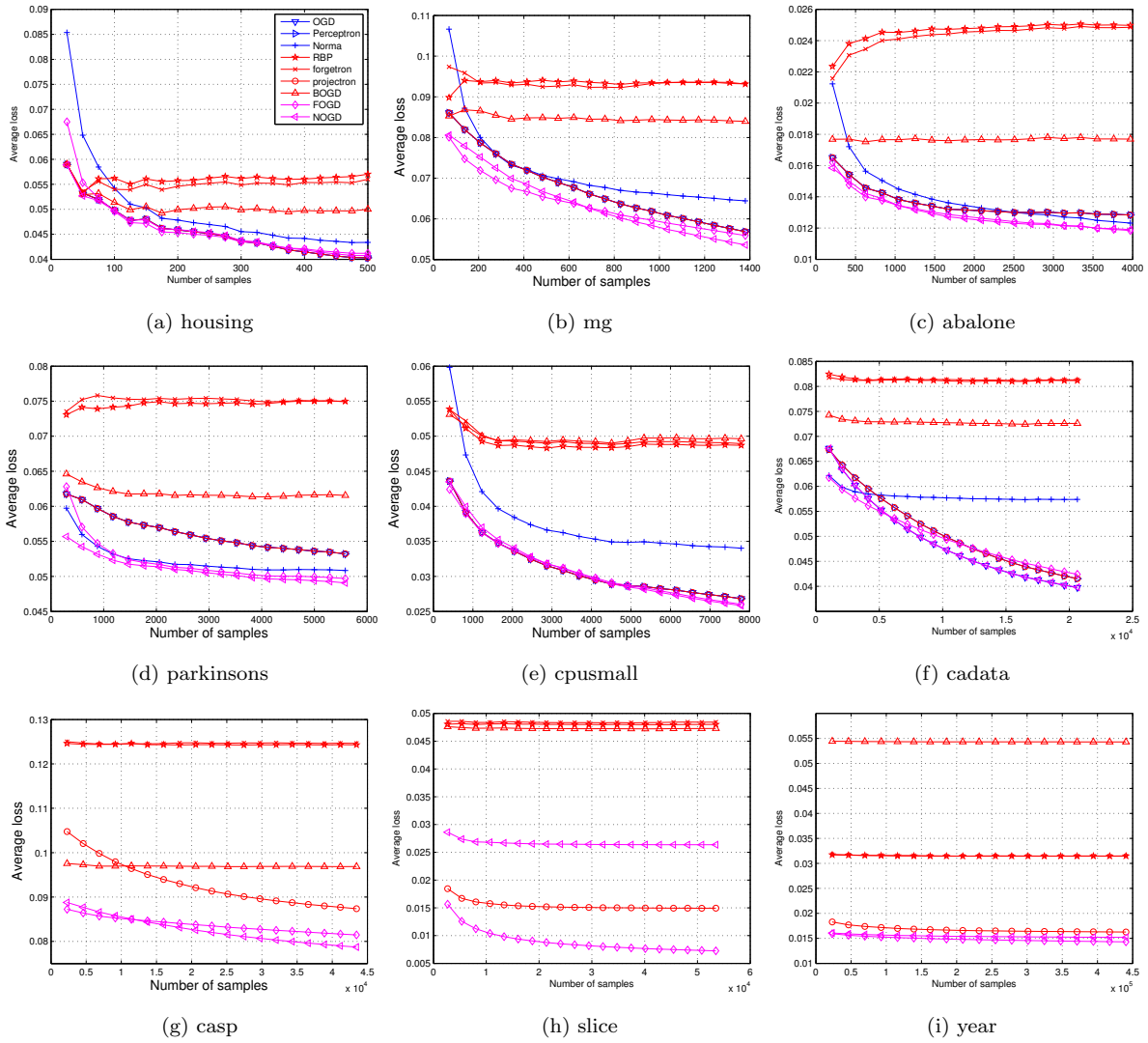
Table 9: Evaluation of Large-scale Online Kernel Learning on Regression Task (Time in Seconds).

loss for most cases while spending comparable or even lower time cost. This encouraging results again validate the advantages of the proposed technique for online kernel regression tasks.

Finally, by examining the two proposed algorithms, FOGD and NOGD, we found that they in general achieve fairly comparable regression accuracy, while FOGD tends to perform more efficiently than NOGD in terms of running time cost. This is primarily because NOGD has to involve the Nystrom matrix approximation which could be computationally intensive.

### 7.5 Comparison between FOGD and NOGD

In the previous experiments, we have made some comparisons of different budget online kernel learning algorithms for different learning tasks, in which the proposed algorithms




(a) housing (b) mg (c) abalone

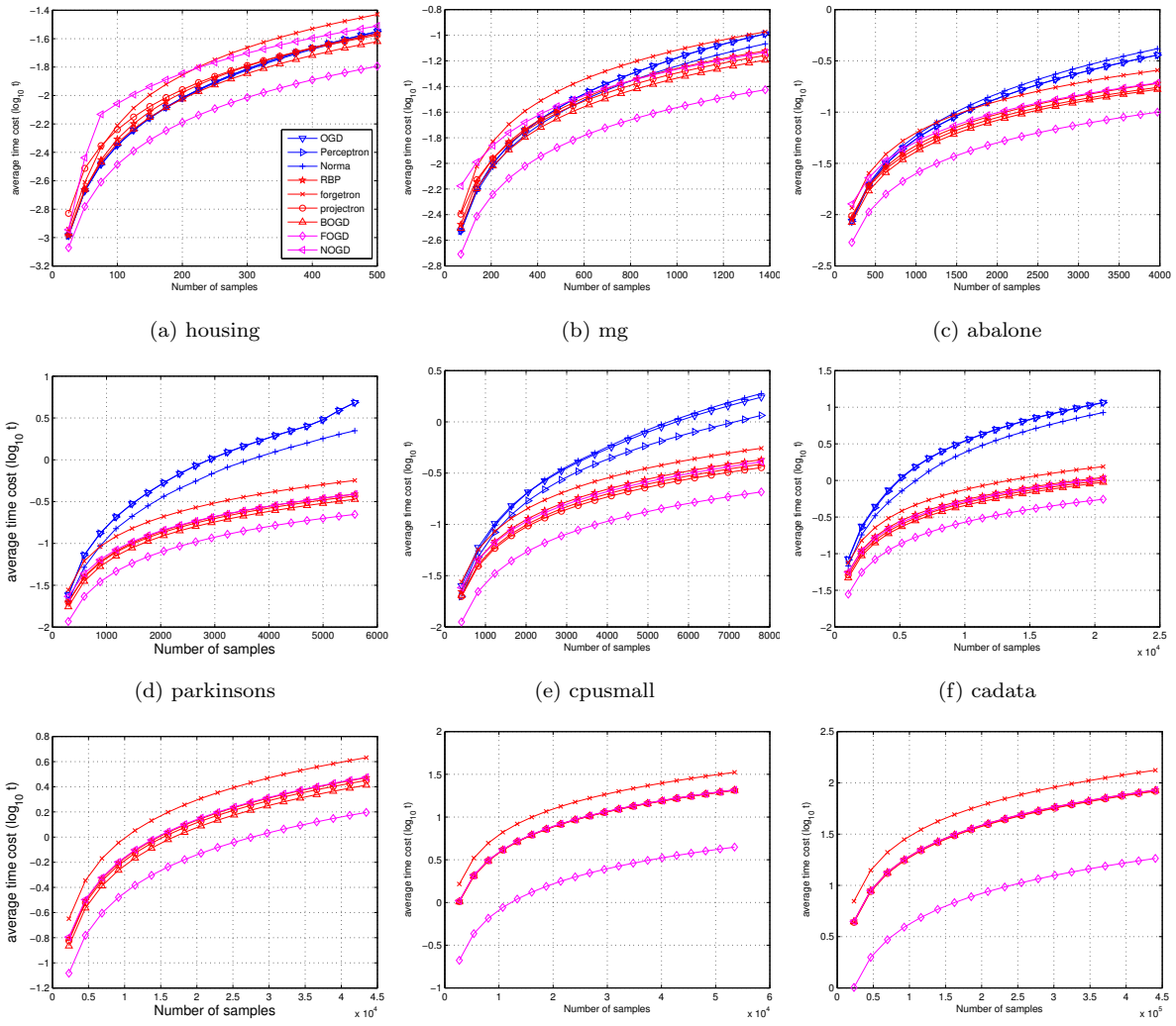
(d) parkinsons (e) cpusmall (f) cadata

(g) casp (h) slice (i) year

Figure 5: Evaluation of online average squared loss on the regression tasks (best viewed in color).

show promising performance. In this section, we conduct both quantitative comparison and in-depth qualitative analysis of the two proposed algorithms in order to better understand their strengths and weaknesses in different scenarios. Specifically, we summarize the comparison of the two algorithms as follows.

First of all, as observed in the previous experiments, the two proposed algorithms in general tends to achieve comparable learning accuracy for most cases. However, NOGD outperforms FOGD in batch setting while FOGD is more accurate in online setting. In terms of running time costs, the result seems relatively implementation dependent. Specifically, when comparing the Matlab implementations of both algorithms, FOGD is faster,




(a) housing (b) mg (c) abalone

(d) parkinsons (e) cpusmall (f) cadata

(g) casp (h) slice (i) year

Figure 6: Evaluation of online average time cost on online regression tasks (best viewed in color).

while NOGD is faster when comparing their C++ implementations. We conjecture that the reason is primarily because the FOGD algorithm is naturally easier for parallelization than NOGD. When running the Matlab implementations, FOGD may take advantages of Matlab-embedded speedup with implicit multi-core parallelization. While running the C++ implementations, we do not explicitly engage any parallelization, and thus NOGD is faster than FOGD when no parallelization is exploited.

Second, the efficiency performance of the two proposed algorithms also depends on the dataset size. For small-sized datasets, FOGD tends to be more efficient, while NOGD tends to be more efficient on larger-sized datasets. The main reason is that a key step of

NOGD is the Nystrom approximation that involves the eigen-decomposition. The eigen-decomposition computation could be potentially very computationally intensive a small-scale dataset, but relatively small or even negligible for a large-scale dataset. By contrast, FOGD does not involve eigen-decomposition and thus does not suffer from such issue for small-scale datasets.

Third, FOGD suffers from high space complexity (high memory cost) when handling datasets with relatively high dimensionality. This is because FOGD has to maintain a $\rho_f B \times d$ matrix in memory for the random Fourier features computation, while NOGD only needs to keep $B \times \rho_n B$ matrix for storing the feature mapping matrix. For a large-scale high-dimensional learning task where $d \gg B$ and $\rho_f > 1$, FOGD will clearly suffer a much higher memory cost than NOGD.

Forth, FOGD has some restrictions in terms of applicable kernels, e.g., shift-invariant kernels as mentioned before. It may be difficult to be generalized for other divers kernels. By contrast, NOGD is based on the the Nyström approximation which only requires the computation of kernel matrix and does not have a restriction on the applicable kernel type as long as it is a valid kernel.

Finally, FOGD may suffer from some practical limitations and implementation challenges for novel feature extension in some real-world applications. For example, consider learning tasks with stream data where novel features may arrive at different time periods in the online learning process. At the beginning of the online learning task, it is impossible to know the full set of features. During the online learning process, whenever a novel feature appears, FOGD has to update the list of $D$ random Fourier components by expanding their dimensionality. Such kind of updating process usually involves a series of memory operations, such as new memory space allocation, copying existing vectors, and freeing memory space of obsolete data, which could be quite expensive if novel feature appears frequently. By contrast, NOGD suffers less for the novel feature extension issue in that we can simply treat the value of a novel feature as zero when computing kernel value between an existing support vector and a new example with the novel feature.

## 8. Conclusions

This paper presented a novel framework of large-scale online kernel learning via functional approximation, going beyond conventional online kernel methods that often adopt the budget maintenance strategy for ensuring the size of support vector is bounded. The basic idea of our framework is to approximate a kernel function or kernel matrix by exploring functional approximation techniques, which transforms the online kernel learning task into an approximate linear online learning problem in a new kernel-inducing feature space that can be further resolved by applying existing efficient and scalable online algorithms. We presented two new algorithms for large-scale online kernel learning tasks: Fourier Online Gradient Descent (FOGD) and Nyström Online Gradient Descent (NOGD), and applied them to tackle different tasks, including binary classification, multi-class classification, and regression tasks. Our promising results on large-scale datasets show the proposed new algorithms are able to achieve the state-of-the-art performance in both learning efficacy and efficiency in comparison to a variety of existing techniques. By comparing the two proposed algorithms, we found that they in general achieve quite comparable learning performance

for most cases, while have different advantages and disadvantages under different scenarios. As the first comprehensive work of exploring functional approximation for large-scale online kernel learning, our framework is generic and can be extended to tackle different learning tasks in other settings (e.g., structured prediction). To facilitate other researchers to re-produce our results, we have released the source code of our implementations. In our future work, we plan to extend our work by exploring parallel computing techniques to make kernel methods practical for massive-scale data analytics tasks.

## Acknowledgments

This work was supported by Singapore MOE tier 1 research grant (C220/MSS14C003). This work was done when Jialei Wang visited Dr Hoi's group.

## References

Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3):143–167, 2007.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

Radha Chitta, Rong Jin, Timothy C Havens, and Anil K Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 895–903. ACM, 2011.

Radha Chitta, Rong Jin, and Anil Jain. Efficient kernel clustering using random fourier features. In *IEEE International Conference on Data Mining*, 2012.

Corinna Cortes, Mehryar Mohri, and Ameet Talwalkar. On the impact of kernel approximation on learning accuracy. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002.

Koby Crammer, Jaz S Kandola, and Yoram Singer. Online classification on a budget. In *Neural Information Processing Systems*, volume 2, page 5, 2003.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: A kernel-based perceptron on a fixed budget. In *Neural Information Processing Systems*, 2005.

Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine Learning*, pages 264–271, 2008.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Maching Learning.*, 37(3):277–296, 1999.

Steven C. H. Hoi, Rong Jin, and Michael R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *Proceedings of the International Conference on Machine Learning*, pages 361–368, Corvalis, Oregon, 2007. ISBN 978-1-59593-793-3.

Steven C. H. Hoi, Rong Jin, Peilin Zhao, and Tianbao Yang. Online multiple kernel classification. *Machine Learning*, 90(2):289–316, 2013.

Steven CH Hoi, Michael R Lyu, and Edward Y Chang. Learning the unified kernel machines for classification. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 187–196. ACM, 2006.

Steven C.H. Hoi, Jialei Wang, and Peilin Zhao. Libol: A library for online learning algorithms. *Journal of Machine Learning Research*, 15:495–499, 2014. URL `http://jmlr.org/papers/v15/hoi14a.html`.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

Jyrki Kivinen, Alex J. Smola, and Robert C. Williamson. Online learning with kernels. In *Neural Information Processing Systems*, pages 785–792, 2001.

Bin Li, Peilin Zhao, Steven CH Hoi, and Vivekanand Gopalkrishnan. Pamr: Passive aggressive mean reversion strategy for portfolio selection. *Machine learning*, 87(2):221–258, 2012.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the International Conference on Machine Learning*, pages 681–688. ACM, 2009.

Francesco Orabona, Joseph Keshet, and Barbara Caputo. The projectron: a bounded kernel-based perceptron. In *Proceedings of the International Conference on Machine Learning*, pages 720–727, 2008.

Francesco Orabona, Joseph Keshet, and Barbara Caputo. Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10:2643–2666, 2009.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Neural Information Processing Systems*, 2007.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

W. Rudin. *Fourier Analysis on Groups*. Wiley-Interscience, 1990.

Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *Conference on Learning Theory*, pages 416–426, 2001.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Math. Program.*, 127(1):3–30, 2011.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge university press, 2004.

Ameet Talwalkar, Sanjiv Kumar, and Henry A. Rowley. Large-scale manifold learning. In *Computer Vision and Pattern Recognition*, 2008.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

Vladimir N Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Jialei Wang, Peilin Zhao, and Steven C. H. Hoi. Cost-sensitive online classification. In *IEEE International Conference on Data Mining*, pages 1140–1145, 2012a.

Jialei Wang, Steven CH Hoi, Peilin Zhao, Jinfeng Zhuang, and Zhi-yong Liu. Large scale online kernel classification. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1750–1756. AAAI Press, 2013.

Zhuang Wang and Slobodan Vucetic. Twin vector machines for online learning on a budget. In *International Conference on Data Mining*, pages 906–917. SIAM, 2009.

Zhuang Wang and Slobodan Vucetic. Online passive-aggressive algorithms on a budget. In *International Conference on Artificial Intelligence and Statistics*, pages 908–915, 2010.

Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13:3103–3131, 2012b.

Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Neural Information Processing Systems*, pages 682–688, 2000.

Hao Xia, Pengcheng Wu, and Steven C. H. Hoi. Online multi-modal distance learning for scalable multimedia retrieval. In *ACM International Conference on Web Search and Data Mining*, pages 455–464, 2013.

Tianbao Yang, Yufeng Li, Mehrdad Mahdavi, Rong Jin, and Zhi hua Zhou. Nystrom method vs random fourier features: A theoretical and empirical comparison. In *Neural Information Processing Systems*, 2012.

Kai Zhang and James T. Kwok. Density-weighted nyström method for computing large kernel eigensystems. *Neural Computation*, 21(1):121–146, 2009.

Kai Zhang, Liang Lan, Zhuang Wang, and Fabian Moerchen. Scaling up kernel svm on limited resources: A low-rank linearization approach. In *International Conference on Artificial Intelligence and Statistics*, pages 1425–1434, 2012.

Peilin Zhao and Steven CH Hoi. Otl: A framework of online transfer learning. In *Proceedings of the International Conference on Machine Learning*, 2010.

Peilin Zhao, Steven C. H. Hoi, and Rong Jin. Double updating online learning. *Journal of Machine Learning Research*, 12:1587–1615, 2011.

Peilin Zhao, Jialei Wang, Pengcheng Wu, Rong Jin, and Steven C. H. Hoi. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. In *Proceedings of the International Conference on Machine Learning*, 2012.