11-2017

Scalable online kernel learning

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

Scalable Online Kernel Learning

by

LU Jing

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

Steven HOI (Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

Jing JIANG
Associate Professor of Information Systems
Singapore Management University

David LO
Associate Professor of Information Systems
Singapore Management University

James KWOK
Professor
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Singapore Management University
2017

Scalable Online Kernel Learning

LU Jing

Abstract

One critical deficiency of traditional online kernel learning methods is their increasing and unbounded number of support vectors (SV's), making them inefficient and non-scalable for large-scale applications. Recent studies on budget online learning have attempted to overcome this shortcoming by bounding the number of SV's. Despite being extensively studied, budget algorithms usually suffer from several drawbacks.

First of all, although existing algorithms attempt to bound the number of SV's at each iteration, most of them fail to bound the number of SV's for the final averaged classifier, which is commonly used for online-to-batch conversion. To solve this problem, we propose a novel bounded online kernel learning method, Sparse Passive Aggressive learning (SPA), which is able to yield a final output kernel-based hypothesis with a bounded number of support vectors. The idea is to attain the bounded number of SV's using an efficient stochastic sampling strategy which samples an incoming training example as a new SV with a probability proportional to its loss suffered. Since the SV's are added wisely and no SV's are removed during the learning, the proposed SPA algorithm achieves a bounded final averaged classifier. We theoretically prove that the proposed SPA algorithm achieves an optimal regret bound in expectation, and empirically show that the new algorithm outperforms various bounded kernel-based online learning algorithms.

Secondly, existing budget learning algorithms are either too simple to achieve satisfactory approximation accuracy, or too computationally intensive to scale for large datasets. To overcome this limitation and make online kernel learning efficient and scalable, we explore two functional approximation based online kernel machine learning algorithms, Fourier Online Gradient Descent (FOGD) and Nys-

tröm Online Gradient Descent (NOGD). The main idea is to adopt the two methods to approximate the kernel model with a linear classifier, so that the efficiency is highly improved. 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

Thirdly, we also extend the proposed algorithms to solve the online multiple kernel learning (OMKL) problem, in which the goal is to significantly reduce the learning complexity of Online Multiple Kernel Classification (OMKC) while achieving satisfactory accuracy as compared with existing unbounded OMKC algorithms. We theoretically prove that the proposed algorithms achieve an optimal regret bound, and empirically show that the new algorithms outperform various bounded kernel-based online learning algorithms.

In conclusion, this work presents several novel solutions for scaling up online kernel learning algorithms for large scale applications. To facilitate other researchers, we also provide an open source software tool-box that includes the implementation of all proposed algorithms in this paper, as well as many state-of-the-art online kernel learning algorithms.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor, Dr. Steven HOI, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. Over the past 5 years, he not only inspires me by sharing his sharp insights for various problems, but also encourages me by his passion and enthusiasm in pursuing great research. He is a perfect supervisor not only for my research, but also for my life.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Jiang Jing, Prof. David Lo, and Prof. James Kwok, for their insightful comments and encouragement, but also for the questions which incented me to widen my research from various perspectives.

My sincere thanks also goes to my collaborators: Peilin Zhao, Jialei Wang, Yue Wu, Doyen Sahoo, Shuji Hao, Chenghao Liu. I feel deeply honored to cooperate with them in my past research.

I am very grateful to Singapore Management University, where I have attended interesting courses, obtained plenty research materials and enjoyed strong academic atmosphere.

Finally, I thank my parents for their support and love that encourage me to pursue my dreams.

List of Publications

Jing Lu, Steven C.H. Hoi, Jialei Wang, Peilin Zhao, Zhi-yong Liu. Large Scale Online Kernel Learning. Journal of Machine Learning Research (JMLR), 17(47), 1-43. 2016. 1. Gold Prize of Best Student Paper Awards from PREMIA in Singapore, 2017

Jing Lu, Peilin Zhao, Steven C. H. Hoi. Online Sparse Passive Aggressive Learning with Kernels. 2016 SIAM International Conference on Data Mining (SDM 2016), May 5-7, 2016, Miami, Florida, USA.

Jing Lu, Doyen Sahoo, Peilin Zhao, Steven HOI. Sparse Passive Aggressive Learning with Application to Bounded Online Multiple Kernel Learning. ACM Transactions on Intelligent Systems and Technology (TIST), accepted.

Jing Lu, Peilin Zhao, Steven C. H. Hoi. Online Passive Aggressive Active Learning. Machine Learning (MLJ), 2016.

Jing Lu, Peilin Zhao, Steven C.H. Hoi. Online Passive Aggressive Active Learning and Its Applications. Journal of Machine Learning Research - Proceedings Track (ACML2014), November 26-28, 2014, Nha Trang, Vietnam.

Jing Lu, Steven C.H. Hoi, Jialei Wang, Peilin Zhao. Second Order Online Collaborative Filtering. Journal of Machine Learning Research - Proceedings Track (ACML2013), Canberra, Australia, November 13-15, 2013.

Shuji Hao, Peilin Zhao, **Jing Lu**, Steven C.H. Hoi, Chunyan Miao, Chi Zhang. SOAL: Second-order Online Active Learning. IEEE International Conference on Data Mining (ICDM2016) Barcelona Dec 12-15, 2016

Shuji Hao, **Jing Lu**, Peilin Zhao, Steven C.H. Hoi. Chunyan Miao and Chi Zhang. Second-order Online Active Learning and Its Applications. IEEE Transactions on Knowledge and Data Engineering (TKDE), accepted

Wu Yue, Steven CH Hoi, Chenghao Liu, **Jing Lu**, Doyen Sahoo, and Nenghai Yu. "SOL: A Library for Scalable Online Learning Algorithms." Neurocomputing 2017 Accepted.

Under Review

Steven Hoi, **Jing Lu**, Doyen Sahoo, and Peilin Zhao, Online Learning: A Comprehensive Survey.

Chapter 1

Introduction

In the introduction chapter, we first present a brief background introduction to Online Kernel Learning, including the problem setting, basic methodology and applications. Following, we discuss the main drawbacks of existing algorithms, which motivate our research. Finally, we present the key ideas of our proposed algorithms and summarize our contributions.

1.1 Background and Motivation

This section provides an easy start for the background and motivates our research.

1.1.1 Batch Learning Vs Online Learning

To warm up, we take the Malicious URL Detection, a well known binary classification problem, as an example. The goal of this task is to acquire an accurate model that can distinguish malicious URLs from benign ones.

Traditional batch/offline learning algorithms usually operate in two separated stages. During the *training* stage, the algorithm is provided with a set of labeled instances $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathcal{X}$ is the feature vector extracted from the i-th URL and y_i is the class label. Based on the statistical properties of this training dataset, the algorithm learns a model $f : \mathcal{X} \to \{\text{Malicious, Benign}\}$ such that the model fits

the training instances properly. Then in the *test* stage, the model is used to predict the class labels of another set of instances.

Despite being studied actively, these batch based algorithms usually suffer from several critical drawbacks. First, most of the batch learning algorithms require that the whole training set to be loaded into the memory for training, which is difficult in real-world applications where data are often of high volume and high velocity. Second, even if the memory cost was not important, searching for the optimal model for large scale datasets is usually very time consuming. Third, each time new instances are added to the training set, the model has to be retrained, which is expensive in scenarios where new data are collected sequentially from time to time. Fourth, in some real world applications, the data pattern might change frequently. As a result, the model trained on old data may not be applicable to new data, which is termed as the "concept drift" problem.

By contrast, online learning algorithms [28] overcome such drawbacks in that the model can be updated instantly for any new data instances. Consider the Malicious URL Detection problem again in online setting. At first we only have a weak model f_0, which might be from simple initialization. During each iteration t, the model receives a new URL feature $\mathbf{x}_t$ and tells whether it is malicious or benign based on the current knowledge f_t. After this prediction, the environment reveals the true label. Depending on the difference between the prediction and the ground truth, the learner will suffer a loss. This loss will be used to update the model f_t to a new and probably more accurate model f_{t+1}.

Online learning techniques can be applied to many real-world applications, such as online spam detection [41], online advertising, multimedia retrieval [74] and computational finance [36]. Compared with batch learning algorithms, online algorithms enjoy many advantages. First, since the instances are processed sequentially, the memory cost for online learning is greatly reduced and there is no retraining cost. Second, the simple update strategy in each iteration can avoid solving large-scale optimization problems, which makes online learning more efficient and scal-

able for large-scale tasks. Third, online models are updated to follow the changing data distributions and overcome the concept drift problem.

In conclusion, online learning is more efficient and scalable compared to batch models, which makes it more suitable for the age of big data and real time applications. In this paper, we will focus on the study of online learning algorithms.

1.1.2 Online Kernel Learning

Different from linear models, which are restricted to making effective classification if data are linearly separable, kernel learning [58] is a family of powerful algorithms for learning nonlinear patterns. This is especially important for problems with complicated decision boundary in relatively low dimensional space. When online learning is combined with kernel learning, we are expecting to get a model that enjoys the advantages of both. It should be able to learn a complicated non-linear decision boundary on sequentially arriving data, with high scalability and efficiency for large scale real time applications.

This combination, however, is nontrivial due to the challenge called "the curse of kernelization". An online kernel learning algorithm usually has to maintain a set of support vectors in memory. During the online learning process, whenever a new incoming training instance is misclassified, it typically will be added to the support vector set, making the size of support vector set unbounded. This is not only a huge burden for the prediction efficiency, but also a danger for memory overflow for a large-scale online learning task.

In this paper, we will study online kernel learning. To make it scalable to large-scale applications, we will address the above problem, the curse of kernelization.

1.1.3 Budget Online Learning

Budget online learning is a straight forward solution to address the critical challenge of online kernel learning. The idea is to set a fixed budget size for the support vec-

tor set. Whenever the number of support vectors extends the budget size, a budget maintenance step is conducted so that the number of support vectors remains the same. For example, the algorithm may discard some support vectors randomly or combine two support vectors [69]. Consequently, the prediction time cost for one instance and the total memory cost remain bounded, making the algorithm relatively scalable compared to non-budget online kernel learning algorithms. Despite extensive studied, existing budget learning algorithms still suffer from sever limitations.

First, existing budget online kernel learning algorithms are not suitable for online-to-batch conversion, a process that aims to convert online classifiers for batch classification purposes [18, 15]. Specifically, one of the most commonly used approaches for online-to-batch conversion is to take the *averaging classifier*, that is the mean of all the online classifiers at all online iterations, as the final classifier for batch classification. Unfortunately, most existing budget online kernel learning algorithms only guarantee that the support vector size at each online iteration is bounded, but fail to yield a sparse averaging classifier after online-to-batch conversion. Consequently, we would like to address this problem by proposing a novel online kernel learning algorithm, whose support vector set of averaged classifier is also bounded.

Second, some efficient algorithms which simply discard old support vectors and add on new ones, are too simple to achieve satisfactory approximation accuracy. While some other algorithms, for example these algorithms based on support vector projection, are, despite more effective, too computationally intensive to run for large datasets. This makes them harmful for the crucial merit of high efficiency of online learning techniques for large-scale applications. So, there is a strong need to develop novel online kernel learning algorithms that adopt totally different approaches from budget learning for large scale applications.

1.1.4 Online Multiple Kernel Learning

Online multiple kernel learning (OMKL) [26, 43] is a technique that learns multiple kernel classifiers and their linear combination weights in online learning process simultaneously. This setting enables not only an automatic selection of kernel functions on the fly but also the combination of information from different sources. Similar to the online single kernel learning algorithms, traditional OMKL algorithms are also not scalable to large scale applications due to the unbounded number of support vectors. And this becomes even more challenging due to the large number of kernel functions. Although one may apply an existing budget online algorithm to bound each individual single-kernel classifier in OMKL with a uniform budget, such a simple approach is not desirable since it treats all the kernels equally and wastes resources in learning with poor kernels, which could result in a large total budget due to many kernels (or equally very poor learning performance if the given total budget is small). This motivates our study on scalable OMKL algorithms.

1.2 Methodology

In this section, we will provide a brief introduction to the main idea of our proposed algorithms. The aim of this paper is to study Online Kernel Learning for large scale applications. As discussed in the introduction section, existing solutions suffer from three critical drawbacks and thus are not scalable. In the following, we will introduce our approaches for addressing these three problems.

1.2.1 Online Sparse Passive Aggressive Learning with Kernels

To address the first problem, i.e. to make the online kernel learning algorithms suitable for online-to-batch conversion, we propose a novel online sparse kernel learning algorithm. In particular, we present a new method for bounded kernel-based online learning method, named "Sparse Passive Aggressive" (SPA) learning,

which extends the online Passive Aggressive (PA) learning method [10] to ensure the final output averaging classifier has the bounded number of support vectors in online-to-batch conversion. Specifically, the basic idea of our method is to explore a simple stochastic sampling rule, which assigns an incoming training example to be a support vector with a higher probability when it suffers a higher loss.

1.2.2 Functional Approximation Based Online Kernel Learning

As discussed previously, the existing budget learning algorithms are either too simple to achieve satisfactory accuracy or too expensive in computational time cost. To solve this problem, we adopt a completely different approach from budget learning, the functional approximation methods.

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.

1.2.3 Online Multiple Kernel Learning

To make the OMKL scalable to large scale applications, we extend the above mentioned three algorithms to address the OMKL problem. In addition to bounding the SV's of each component classifier, we adopt the Hedge algorithm in the combination of multiple kernels. Specially, the combination weight of each classifier is set

according to its historical accuracy. As a results, the performance of the combined multiple kernel classifier can track the best performing component classifier.

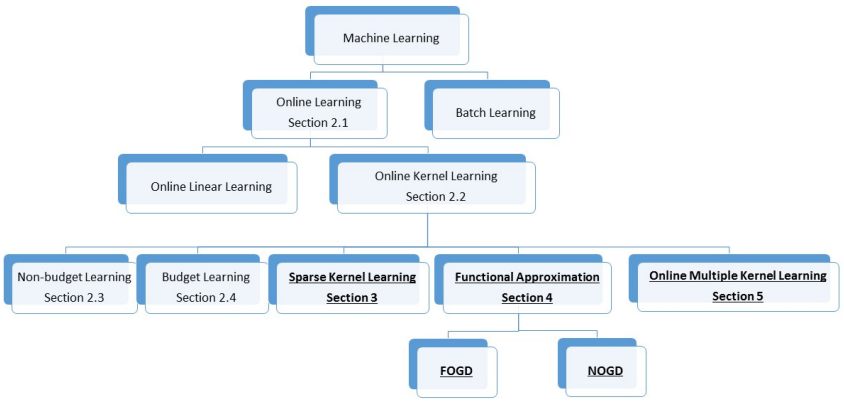


Figure 1.1: The Dependence Structure of the Topics of this paper.

1.3 Summary of Contributions

Firstly, to make online kernel learning applicable to online-to-batch conversion, we propose "Sparse Passive Aggressive" (SPA) learning. We theoretically prove that the propose algorithm not only bounds the number of support vectors in each iteration but also achieves an optimal regret bound in expectation. We conduct an extensive set of empirical studies which show that the proposed algorithm outperforms a variety of bounded kernel-based online learning algorithms.

Secondly, to make online kernel learning efficient and scalable with reasonable accuracy, we propose two novel functional approximation based algorithms, FOGD and NOGD algorithms. 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.

Thirdly, we extend the three proposed online kernel learning algorithms to OMKL cases. We make each of the component classifiers scalable and efficient

and learns the optimal combination weights of these classifiers. Theoretical analysis and empirical results are also provided.

1.4 Organization

The reminder of this dissertation is organized as follows: Chapter 2 is a literature review which examines closely related research including online learning, kernel learning, online kernel learning, budget online learning and online multiple kernel learning. Chapter 3 studies a sparse kernel learning algorithm that is applicable to online-to-batch conversion. Chapter 4 proposes two novel online kernel learning algorithms that are based on functional approximation methods and are scalable to large scale applications. Chapter 5 introduces the extension of the three algorithms to multiple kernel learning. Finally, Chapter 6 concludes this dissertation and discusses the future work.

Figure 1.1 describes the dependence structure of the topics of this thesis. Specifically, we highlight our main contributions in Section 3, 4 and 5.

Chapter 2

Literature Review

In this chapter, we will provide a comprehensive literature review on the most closely related works to the topic Large Scale Online Kernel Learning. Including:

- *Online Learning*, an important family of scalable machine learning algorithms which are devised to learn in a sequential manner.
- *Kernel Learning*, a family of machine learning algorithms that is able to learn complicated nonlinear patterns.
- *Online Kernel Learning*, which combines the advantages of the two above and can learn nonlinear patterns from sequential data. We will also discuss "the curse of kernelization", the key drawbacks of online kernel learning algorithms that limits its applications in large scale learning tasks.
- *Budget Online Learning*. Through different budget maintenance strategies, budget learning algorithms bound the number of support vectors and can achieve relatively high efficiency compared to non-budget online kernel learning algorithms.
- *Online Multiple Kernel Learning*. OMKL is a more challenging task compared to the single kernel learning, in which an algorithm learns all component classifiers and their combination weights simultaneously.

2.1 Online Learning

In traditional offline / batch machine learning methods, a prediction model is typically learnt from a collection of training data in a batch fashion, whose performance would be tested later on a test dataset. These batch algorithms usually suffer from several drawbacks: (1) the burdensome or sometimes unrealistic requirement that the entire training dataset should be available before the training; (2) low efficiency in both time and space costs; (3) poor scalability for large scale applications and (4) the high re-training cost when new training data arrive frequently.

Online learning algorithms overcome such drawbacks by updating the model instantly whenever new data instances arrive. As a result, online algorithms are efficient and scalable to large scale applications especially when the data arrive sequentially.

Algorithm 1 The Online Binary Classification Procedure.

Initialize the prediction function as $\mathbf{w}_1$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;

Predict $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$ as the label of $\mathbf{x}_t$;

Receive correct label: $y_t \in \{-1, +1\}$;

Suffer loss: $\ell_t(\mathbf{w}_t)$, which depends on the difference between $\mathbf{w}_t^\top \mathbf{x}_t$ and y_t;

Update the prediction function $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$;

end for

2.1.1 Problem Formulation

Without loss of generality, we first present the formulation of linear binary online classification, which is a classical online learning formulation. Consider a sequence of instances, $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $t \in \{1, ..., T\}$ is the time index, $\mathbf{x}_t \in \mathbb{R}^d$ is the feature vector of the instance we receive at time t, and $y_t \in \{+1, -1\}$ is the true class label of $\mathbf{x}_t$.

The online binary classification takes place sequentially. On the t-th round, an instance $\mathbf{x}_t$ is received by the learner, which then employs a binary classifier $\mathbf{w}_t$ to make prediction, $\hat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$. The real value $f_t(\mathbf{x}_t) = \mathbf{w}_t^\top \mathbf{x}_t$ is called prediction score whose absolute value indicates the confidence of the prediction. After making the prediction, the learner receives the true class label y_t and thus can measure the suffered loss, hinge loss for example, $\ell_t(\mathbf{w}_t) = \max\left(0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t\right)$. Whenever the loss is nonzero, the learner updates the prediction model from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ by some strategy on the training example $(\mathbf{x}_t, y_t)$. The procedure of Online Binary Classification is summarized in Algorithm 1.

By running an online learner over T rounds, the regret is defined as

$$R_T = \sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \min_{\mathbf{w}} \sum_{t=1}^{T} \ell_t(\mathbf{w}) \tag{2.1}$$

where the first term is the accumulated loss suffered by an online learning algorithm over T rounds and the second term is the accumulated loss suffered by the optimal classifier assuming that we had foresight in all the training instances. The goal of online learning is to minimize the regret in the long run.

2.1.2 Online Learning Algorithms

In the following, we will review some classical online learning algorithms.

Perceptron

As the classical and the oldest algorithm for online learning, for an online binary classification task, Perceptron [49] runs as follows:

Algorithm 2 Perceptron

Initialize the prediction function as $\mathbf{w}_1$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;

Predict $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$ as the label of $\mathbf{x}_t$;

Receive correct label: $y_t \in \{-1, +1\}$;

if $\widehat{y}_t \neq y_t$ **then**

Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$;

end if

end for

The update rule is simply adding or subtracting the current instance from the model. In theory, by assuming the data is separable with some margin, the Perceptron algorithm makes at most $\left(\frac{R}{\gamma}\right)^2$ mistakes, where the margin γ is defined as $\gamma = \min_{t \in [T]} |\mathbf{x}_t \cdot \mathbf{w}^*|$, $\mathbf{w}^*$ is the optimal classifier and R is a constant such that $\forall t \in [T], \|\mathbf{x}_t\| \leq R$. The larger the margin γ is, the tighter the mistake bound will be.

Online Gradient Descent

The Online Gradient Descent algorithm (OGD) [83] runs as follows:

Algorithm 3 Online Gradient Descent algorithm (OGD)

Initialize the prediction function as $\mathbf{w}_1$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;

Predict $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$ as the label of $\mathbf{x}_t$;

Receive correct label: $y_t \in \{-1, +1\}$;

Loss is $\ell_t(\mathbf{w}_t) = \max\left(0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t\right)$;

The algorithm updates the model by $\mathbf{w}_{t+1} = \Pi_{\S}(\mathbf{w}_t - \eta_t \nabla \ell_t(\mathbf{w}_t))$

end for

At every iteration, the algorithm takes a step from the current model, in the direction of the gradient of the current loss function. The algorithm then makes a projection of the model onto the feasible domain, i.e., $\Pi_{\S}(\mathbf{u}) = \arg\min_{\mathbf{w}\in\S} \|\mathbf{w}-\mathbf{u}\|$.

OGD is simple and easy to implement. In theory [83], a simple OGD algorithm achieves sublinear regret $O(\sqrt{T})$ for an arbitrary sequence of T convex cost functions (of bounded gradients), with respect to the best single decision in hindsight. Under a strong assumption of strictly convex cost functions (with bounded first and second derivatives), OGD can yield logarithmic regret $O(\log T)$ [25].

Online Passive Aggressive

The Online Passive Aggressive algorithm (PA) is a popular family of first-order online learning algorithms which generally follows the principle of margin-based learning [10]. Specifically, given an instance $\mathbf{x}_t$ at round t, PA formulates the updating optimization as follows:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}||\mathbf{w}-\mathbf{w}_t||^2 \quad s.t.\ \ell_t(\mathbf{w}) = 0 \tag{2.2}$$

where $\ell_t(\mathbf{w}) = \max(0, 1 - y_t\mathbf{w}\cdot\mathbf{x}_t)$ is the hinge loss. PA ensures the updated classifier $\mathbf{w}_{t+1}$ should stay close to the previous classifier ("passiveness") and every incoming instance should be classified by the updated classifier correctly ("aggressiveness"). The regular PA algorithm assumes training data is always separable, which may not be true for noisy training data from real-world applications. To overcome the above limitation, two variants of PA relaxes the assumption as follows:

$$\begin{aligned}
&\text{PA}-\text{I}: \mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}||\mathbf{w}-\mathbf{w}_t||^2 + C\xi \\
&\quad s.t. \ell_t(\mathbf{w}) \le \xi \ \text{ and } \ \xi \ge 0 \\
&\text{PA}-\text{II}: \mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}||\mathbf{w}-\mathbf{w}_t||^2 + C\xi^2 \quad s.t. \ell_t(\mathbf{w}) \le \xi
\end{aligned} \tag{2.3}$$

where C is a positive parameter to balance the tradeoff between "passiveness" (first regularization term) and "aggressiveness" (second slack-variable term).

It is important to note a major difference between PA and Perceptron algorithms. Perceptron makes an update only when there is a classification mistake. However, PA algorithms aggressively make an update whenever the loss is nonzero (even if the classification is correct). As a result, PA algorithms often outperform Perceptron significantly.

Others

There are many other online learning algorithms that proven successful in literature, mainly categorized into two major types: first-order online learning and second-order online learning algorithms.

First order online learning algorithms usually learns a simple model of linear weight vector $\mathbf{w}_t$, and thus are efficient in both prediction and update step. Apart from the Perceptron, OGD and PA algorithms we introduced, there are also some others, such as Approximate Large Margin Algorithms (ALMA) [23] which is a large margin variant of the p-norm Perceptron algorithm and the Relaxed Online Maximum Margin Algorithm (ROMMA) [39]. Many of these algorithms often follow the principle of large margin learning. The metaGrad algorithm [63] tries to adapt the learning rate automatically for faster convergence.

Unlike the first-order online learning algorithms which only exploit the first order derivative information of the gradient for the online optimization tasks, second-order online learning algorithms exploit both first-order and second-order information. Consequently, second-order online learning algorithms often fall short in higher computational complexity while their convergence rate usually outperforms that of first order algorithms. Representative works include the Second Order Perceptron (SOP) [4], Confidence Weighted Learning (CW) [18], Adaptive Regularization of Weight Vectors (AROW) [12] ,Soft Confidence weighted Learning (SCW) [67] and Ada-Gradient [20].

2.2 Kernel Learning

In the previous section, algorithms learn a linear predictor $\mathrm{sign}(\mathbf{w}^\top \mathbf{x})$. Obviously, this predictor can only make accurate prediction under the assumption that the instances are linear separable. In practice, however, the data may probability contain complicated pattern and be linear inseparable in most of the cases. A straight forward way to solve this problem is to map the data to another high dimensional (possibly infinite dimensional) space using a mapping function $\phi(\cdot)$, so that the new training instances $\phi(\mathbf{x}_t)$ are linearly separable in the new space.

Consider a linear classification algorithm whose weight function $\mathbf{w}$ is the weighted sum of all training instances, i.e. $\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}_i)$, which is a common setting in many machine learning algorithms both batch setting (the Support Vector Machine, SVM [64] for example) and online setting (Perceptron, OGD and PA ect). The prediction score is

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \sum_i \alpha_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x})$$

Obviously, the prediction score only depends on the data through the inner product in $\mathcal{H}$ space. If we define a *kernel* function $\kappa(\cdot, \cdot)$ such that $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$, then we can avoid computing the expensive feature mapping and inner product explicitly. This is very beneficial when the feature mapping is complicated or even to an infinite dimensional space. Here are two examples of the most widely used kernels, the Gaussian kernel, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\delta})$ and Polynomial Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^d$. We recommend the reader to read a good book [58] for the detailed concepts and methodologies of kernel learning.

We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \to \mathbb{R}\}$. Further, we consider $\mathcal{H}$ a Reproducing Kernel Hilbert Space (**RKHS**) endowed with a kernel function $\kappa(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ [64] implementing the inner product$\langle \cdot, \cdot \rangle$ such that: 1) κ has the reproducing property $\langle f, \kappa(\mathbf{x}, \cdot) \rangle = f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$; 2) $\mathcal{H}$ is the closure of the

span of all $\kappa(\mathbf{x}, \cdot)$ with $\mathbf{x} \in \mathbb{R}^d$, that is, $\kappa(\mathbf{x}, \cdot) \in \mathcal{H}\ \forall \mathbf{x} \in \mathcal{X}$. The inner product $\langle \cdot, \cdot \rangle$ induces a norm on $f \in \mathcal{H}$ in the usual way: $\|f\|_{\mathcal{H}} := \langle f, f \rangle^{\frac{1}{2}}$. To make it clear, we denote by $\mathcal{H}_\kappa$ an RKHS with explicit dependence on kernel κ. Throughout the analysis, we assume $\kappa(\mathbf{x}, \mathbf{x}) \leq X^2, \forall \mathbf{x} \in \mathbb{R}^d$.

It then seems trivial to generalize the linear algorithms to the nonlinear case by simply replacing all inner products with kernel functions. So the prediction score is

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \sum_i \alpha_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

where the $\mathbf{x}_i$'s are instances used in training. If the weight $\alpha_i \neq 0$, we call the instance $\mathbf{x}_i$ a *Support Vector* (SV). Thus, we rewrite the previous classifier as $f(\mathbf{x}) = \sum_{i \in \mathcal{SV}} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$, where $\mathcal{SV}$ is the set of SV's and i is its index. We use the notation $|\mathcal{SV}|$ to denote the SV set size.

This generalization, however, is non-trivial because of the extremely high time cost of kernel calculations compared to a single inner product calculation . For batch training, an algorithm needs to calculate a kernel matrix $K \in \mathbb{R}^{N \times N}$, where $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ and N is the number of training instances. The time complexity of this step is $O(N^2)$. And the time complexity for each prediction is nearly $O(N)$ if most of the weights α_i are non-zero. This makes kernel learning algorithms unscalable to large scale application (with large N value).

Here comes the key question of designing kernel learning algorithms, i.e. how to make the kernel algorithms efficient and scalable? In literature, many algorithms attempt to address this problem in batch setting, including LLSVM [78] which approximates the kernel matrix by a low rank matrix to speed up the kernel SVM training, the Random Fourier Features algorithm [47] that approximates the kernel function by the inner product of two random Fourier feature vectors, Core Vector Machine (CVM) [62] which also reduces the time complexity by reasonable approximation of the original kernel SVM problem, etc. There are also lots of outstanding kernel learning algorithms in online setting, which will be introduced in the later

section.

2.3 Online Kernel Learning

In the previous section, we introduced kernel learning, which is a powerful tool for nonlinear learning, and online learning, which is efficient and scalable to large scale sequential learning. Now combining the advantages of the two, we discuss the topic Online Learning Learning.

2.3.1 Problem Formulation

Without loss of generality, we will adopt online binary classification setting and reuse the previous notations for this section.

The goal of Online Kernel Learning is to learn a nonlinear classifier $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of labeled instances $(\mathbf{x}_t, y_t), t = 1, ..., T$ and build the classification rule as: $\widehat{y_t} = \text{sign}(f(\mathbf{x}_t))$. We measure the classification confidence of certain instance $\mathbf{x}_t$ by $|f(\mathbf{x}_t)|$. Similarly to the linear case, for an online classification task, one can define the hinge loss function $\ell(\cdot)$ for the t-th instance using the classifier at the t-th iteration:

$$\ell((\mathbf{x}_t, y_t); f_t) = \max(0, 1 - y_t f_t(\mathbf{x}_t))$$

After T iterations, an online kernel learner aims to achieve the lowest regret R_T:

$$R_T = \sum_{t=1}^{T} \ell_t(f_t) - \sum_{t=1}^{T} \ell_t(f^*),$$

where $\ell_t(\cdot)$ is the loss for the classification of instance $(\mathbf{x}_t, y_t)$, which is short for $\ell((\mathbf{x}_t, y_t); \cdot)$, f^* is the optimal fixed solution assuming we had foresight for all the instances, i.e. $f^* = \text{argmin}_f \sum_{t=1}^{T} \ell_t(f)$

2.3.2 Online Kernel Learning Algorithm

In literature, different online kernel methods have been proposed.

Kernelized Perceptron

The Kernelized Perceptron algorithm [22] runs as follows:

Algorithm 4 Kernelized Perceptron

INIT: $f_0 = 0$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;

Predict $\hat{y}_t = \text{sign}(f_t(\mathbf{x}_t))$;

Receive correct label: $y_t \in \{-1, +1\}$;

if $\hat{y}_t \neq y_t$ **then**

$\mathcal{SV}_{t+1} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$, $f_{t+1} = f_t + y_t \kappa(\mathbf{x}_t, \cdot)$;

end if

end for

The algorithm works similarly to the linear Perceptron algorithm, except that the inner product in the linear classifier, i.e., $f_t(\mathbf{x}_t) = \sum_i \alpha_i \mathbf{x}_i^\top \mathbf{x}_t$, is replaced by a kernel function in the kernel Percetron.

Kernelized OGD The OGD algorithm can be extended with kernels [34]:

Algorithm 5 Kernelized OGD

INIT: $f_0 = 0$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$ and predict $\hat{y}_t = \text{sign}(f_t(\mathbf{x}_t))$;

Receive correct label: $y_t \in \{-1, +1\}$;

if $\ell_t(f_t) > 0$ **then**

$\mathcal{SV}_{t+1} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$, $f_{t+1} = f_t - \eta_t \nabla \ell_t(f_t(\mathbf{x}_t)) = f_t - \eta_t \ell_t' \kappa(\mathbf{x}_t, \cdot)$;

end if

end for

where $\eta_t > 0$ is the learning rate parameter, and ℓ'_t is the derivative of loss function with respect to the classification score $f_t(\mathbf{x}_t)$.

Others

The kernel trick implies that the inner product between any two instances can be replaced by a kernel function, i.e., $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j), \forall i, j$, where $\phi(\mathbf{x}_t) \in R^D$ denotes the feature mapping from the original space to a new D-dimensional space which can be infinite. Using the kernel trick, many existing linear online learning algorithms can be easily extended to the kernelized variants, such as the kernelized Perceptron and kernelized OGD as well as the kernel PA variants [10]. However, some algorithms that use complex update rules are non-trivial to be converted into kernelized versions, such as the Confidence Weighted algorithms [18]. Finally, some online kernel learning methods also attempt to make more effective updates at each iteration. For example, the Double Updating Online Learning (DUOL) [81] improves the efficacy of traditional online kernel learning methods by not only updating the weight of the newly added SV, but also the weight for one existing SV.

2.3.3 The Curse of Kernelization

The key advantage of online kernel learning is the ability of solving linearly non-separable tasks. Despite enjoying better performance over linear models, online kernel learning falls short in a critical drawback, that is, the growing unbounded number of support vectors — a challenge termed as "Curse of Kernelization" [69].

During the online learning process, whenever a new prediction error appears, a new SV is added to the classifier. Note that the time complexity of the prediction step is proportionate to the SV set size, $O(|\mathcal{SV}|)$. Usually, the time complexity of the whole learning process is $O(T^2)$, which is unrealistic for large scale problems. Apart from the efficiency issue, the continually growing of $\mathcal{SV}$ is also a huge burden for the memory cost.

2.4 Budget Online Learning

Budget Online Learning is designed to address the key challenge of online kernel learning, i.e. the unbounded number of SV's. In literature, a family of algorithms, termed "budget online kernel learning", have been proposed to bound the number of SV's with a fixed budget $B = |\mathcal{SV}|$ using diverse budget maintenance strategies whenever the budget overflows. Most existing budget online kernel learning methods maintain the budget by three strategies: (i) SV Removal, (ii) SV Projection, and (iii) SV Merging.

2.4.1 SV Removal

This strategy maintains the budget by a simple and efficient way: 1) update the classifier by adding a new SV whenever necessary (depending on the prediction mistake/loss); 2) if the SV size exceeds the budget, discard one existing SV and update the classifier accordingly. The detailed framework is summarized as follows:

Algorithm 6 SV Removal

INIT: $f_0 = 0$;

for $t = 1, 2, \ldots, T$ **do**

Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$ and predict $\hat{y}_t = \text{sign}(f_t(\mathbf{x}_t))$;

Receive correct label: $y_t \in \{-1, +1\}$;

if Update is needed **then**

Update the classifier from f_t to $f_{t+\frac{1}{2}}$ and $\mathcal{SV}_{t+\frac{1}{2}} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$

end if

if $|\mathcal{SV}_{t+\frac{1}{2}}| > B$ **then**

Discard one existing SV $(\mathbf{x}_{del}, y_{del})$,

Update the classifier from $f_{t+\frac{1}{2}}$ to f_{t+1} and $\mathcal{SV}_{t+1} = \mathcal{SV}_{t+\frac{1}{2}} - (\mathbf{x}_{del}, y_{del})$

end if

end for

The above framework has two critical steps: (i) how to update the classifier

and (ii) how to choose one existing SV for removal. The first step depends on which online learning method is used. Fore example, the update is based on the Perceptron algorithm in RBP [3], Forgetron [16], and Budget Perceptron [11], the OGD algorithm is adopted as the update step for BOGD [82] and BSGD+ removal [69], while PA is used for performing update in BPA-S [71].

The second step of SV removal, is to decide which existing SV $(\mathbf{x}_{del}, y_t)$ to remove in order to minimize the harm to the resulting classifier. One simple way is to randomly discard one existing SV uniformly with probability $\frac{1}{B}$, as adopted by RBP [3] and BOGD [82]. Besides, instead of choosing randomly, another way as used in "Forgetron" [16] is to discard the oldest SV by assuming that an older SV is less representative for the distribution of fresh training data streams. Despite enjoying the merits of simplicity and high efficiency, these methods are often too simple to achieve satisfactory learning accuracy results.

To optimize the performance, some approaches have tried to perform exhaustive search in deciding the best SV for removal. For instance, the Budget Perceptron algorithm [11] searches for one SV that is classified with high confidence by the classifier:

$$y_{del}(f_{t+\frac{1}{2}}(\mathbf{x}_{del}) - \alpha_{del}\kappa(\mathbf{x}_{del}, \mathbf{x}_{del})) > \beta$$

where $\beta > 0$ is a fixed tolerance parameter. BPA-S shares the similar idea of exhaustive search. For every $r \in [B]$, a candidate classifier $f^r = f_{t+\frac{1}{2}} - \alpha_r \kappa(\mathbf{x}_r, \cdot)$ is generated by discarding the r-th SV from $f_{t+\frac{1}{2}}$. By comparing the B candidate classifiers, the algorithm selects the one that minimizes the current objective function of PA:

$$f_{t+1} = \underset{r\in[B]}{\operatorname{argmin}} \frac{1}{2}||f^r - f_t||^2_{\mathcal{H}} + C\ell_t(f^r)$$

where $C > 0$ is the regularization parameter of PA that balances aggressiveness and passiveness.

Comparing the principles of different SV removal strategies, we found that a simple rule may not always generate satisfactory accuracy, while an exhaustive

search often incurs non-trivial computational overhead, which again may limit the application to large-scale problems. When deploying a solution in practice, one would need to balance the trade-off between effectiveness and efficiency.

2.4.2 SV Projection

This strategy first appeared in [45] where two new algorithms, Projectron and Projectron++, were proposed, which significantly outperform the previous SV removal based algorithms such RBP and Forgetron. The SV projection method works as follows:

(1) update classifier f_t to $f_{t+\frac{1}{2}}$ with a new SV added (if necessary), which is similar to the steps discussed in the SV removal section.

(2) choose a SV $(\mathbf{x}_{del}, y_{del})$ to remove.

(3) choose the a subset of $\mathcal{SV}$ as the projection base, which will be denoted by $\mathcal{P}$.

(4) use a linear combination of kernels in $\mathcal{P}$ to approximate the removed SV. The procedure of finding the optimal linear combination can be formulated as a convex optimization of minimizing the projection error:

$$\beta = \underset{\beta \in \mathbb{R}^{|\mathcal{P}|}}{\operatorname{argmin}} E_{proj} = \underset{\beta \in \mathbb{R}^{|\mathcal{P}|}}{\operatorname{argmin}} ||\alpha_{del}\kappa(\mathbf{x}_{del}, \cdot) - \sum_{i \in \mathcal{P}} \beta_i \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}}$$

(5) combines this linear combination with the original classifier:

$$f_{t+1} = f_{t+\frac{1}{2}} - \alpha_{del}\kappa(\mathbf{x}_{del}, \cdot) + \sum_{i \in \mathcal{P}} \beta_i \kappa(\mathbf{x}_i, \cdot)$$

There are several algorithms adopting the projection strategy, for example Projectron, Projectron++, BPA-P, BPA-NN [71] and BSGD+Project [69]. These methods differ in a few aspects. First, the update rules are based on different online learning algorithms. Generally speaking, PA based and OGD based trend to outperform Perceptron based algorithms because of their effective update. Second,

the choice of discarded SV is different. Since projection itself is relative slow, exhaustive search based algorithms (BPA-NN, BPA-P) are extremely time consuming. Thus algorithms with simple selecting rules are prefered (Projectron, Projectron++, BSGD+Project). Third, the choice of projection base set $\mathcal{P}$ is different. In Projectron, Projectron++ BPA-P and BSGD+Project, the discarded SV is projected onto the whole SV set, i.e. $\mathcal{P} = \mathcal{SV}$. While in BPA-NN, $\mathcal{P}$ is only a small subset of $\mathcal{SV}$, made up of the nearest neighbors of the discarded SV $(\mathbf{x}_{del}, y_t)$. In general, a larger projection base set implies a more complicated optimization problem and thus more time costs. The research direction of SV projection based budget learning is to find a proper way of selecting $\mathcal{P}$ so that the algorithm achieves the minimized projection error with a relative small projection base set.

Table 2.1: Comparisons of different budget online kernel learning algorithms.

Algorithms	Update Strategy	Budget Strategy	Time	Space
Stoptron [45]	Perceptron	Stop	$O(1)$	$O(B)$
Budget Perceptron [11]	Perceptron	Removal	$O(B^2)$	$O(B)$
RBP [3]	Perceptron	Removal	$O(B)$	$O(B)$
Forgetron[16]	Perceptron	Removal	$O(B)$	$O(B)$
BOGD[82]	OGD	Removal	$O(B)$	$O(B)$
BPA-S [71]	PA	Removal	$O(B)$	$O(B)$
BSGD+removal [69]	OGD	Removal	$O(B)$	$O(B)$
Projectron [45]	Perceptron	Projection	$O(B^2)$	$O(B^2)$
Projectron++ [45]	Perceptron	Projection	$O(B^2)$	$O(B^2)$
BPA-P [71]	PA	Projection	$O(B^3)$	$O(B^2)$
BPA-NN [71]	PA	Projection	$O(B)$	$O(B)$
BSGD+projection [69]	OGD	Projection	$O(B^2)$	$O(B^2)$
BSGD+merging [69]	OGD	Merging	$O(B)$	$O(B)$

2.4.3 SV Merging

In [69], a SV merging method BSGD+Merge was proposed that attempts to replace the sum of two SV's $\alpha_m\kappa(\mathbf{x}_m, \cdot) + \alpha_n\kappa(\mathbf{x}_n, \cdot)$ by a newly created support vector $\alpha_z\kappa(\mathbf{z}, \cdot)$, where α_m, α_n and α_z are the corresponding coefficients of $\mathbf{x}_m$, $\mathbf{x}_n$ and $\mathbf{z}$. Following the previous discussion, the goal of online budget learning through SV

merging strategy is to find the optimal $\alpha_z \in \mathbb{R}$ and $\mathbf{z} \in \mathbb{R}^d$ that minimizes the gap between f_{t+1} and $f_{t+\frac{1}{2}}$.

As it is relatively complicated to optimize the two terms simultaneously, this optimization is divided into two steps. First, assuming the coefficient of $\mathbf{z}$ is $\alpha_m + \alpha_n$, this algorithm tries to create the optimal support vector that minimizes the merging error. The first optimization is

$$\min_{\mathbf{z}} ||(\alpha_m + \alpha_n)\kappa(\mathbf{z}, \cdot) - (\alpha_m \kappa(\mathbf{x}_m, \cdot) + \alpha_n \kappa(\mathbf{x}_n, \cdot))||$$

The solution is $\mathbf{z} = h\mathbf{x}_m + (1-h)\mathbf{x}_n$, where $0 < h < 1$ is a real number that can be found by a line search method. This solution indicates that the optimal created SV lies on the line connecting $\mathbf{x}_m$ and $\mathbf{x}_n$. After obtaining the optimal created SV $\mathbf{z}$, the next is to find the optimal coefficient α_z, which can be formulated as

$$\min_{\alpha_z} ||(\alpha_z \kappa(\mathbf{z}, \cdot) - (\alpha_m \kappa(\mathbf{x}_m, \cdot) + \alpha_n \kappa(\mathbf{x}_n, \cdot))||.$$

The solution becomes $\alpha_z = \alpha_m \kappa(\mathbf{x}_m, \mathbf{z}) + \alpha_n \kappa(\mathbf{x}_n, \mathbf{z})$. The remaining is how to select the two SV's $\mathbf{x}_m$ and $\mathbf{x}_n$ for merging. The ideal solution is to find the pair with the minimal merging error through an exhaustive search, which however often requires $O(B^2)$ time complexity. How to perform exhaustive search efficiently SV merging remains an open challenge.

2.4.4 Comparison of Budget Online Learning Algorithms

Among the various algorithms of budget online kernel learning using the idea of budget maintenance, the key differences are the updating rules and budget maintenance strategies. Table 2.1 gives a summary of different algorithms and their properties.

2.5 Online Multiple Kernel Learning

Multiple Kernel Learning (MKL) [35, 2] aims to find the optimal (linear) combination of a pool of predefined kernel functions in learning kernel-based predictive models. In comparison to single kernel learning, MKL can not only avoid heuristic manual selection of best kernels, but also achieve better performance by combining multiple kernels, particularly for learning from data with heterogeneous representations.

2.5.1 Problem Formulation

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

$$\min_{f \in \mathcal{H}_\kappa} P(f) = R(f) + \frac{1}{T} \sum_{t=1}^{T} \ell_t(f),$$

where R is a regularization function used to control model complexity and $\mathcal{H}_\kappa$ is a Reproducing Kernel Hibert Space endowed with a kernel function $\kappa(\cdot, \cdot)$. Different from the single kernel learning problem, in the OMKL problem, the kernel function is usually defined as a linear combination of m kernels $\mathcal{K} = \{\kappa_i : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, i = 1, \ldots, m\}$, whose weights are denoted by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$ and lie in a simplex, $\Delta = \{\boldsymbol{\theta} \in \mathbb{R}_+^m \, | \boldsymbol{\theta}^T \mathbf{1}_m = 1\}$. Consequently, the kernel function can be denoted as

$$\kappa(\boldsymbol{\theta})(\cdot, \cdot) = \sum_{i=1}^{m} \theta_i \kappa_i(\cdot, \cdot)$$

Although batch MKL methods have been extensively studied recently [59, 24, 65, 1, 48], unfortunately, the generalization from batch MKL to its online counterpart is far from straightforward. First of all, different from batch MKL that can in principle be solved via cross-validation, online learning with multiple kernels has no foresight on the best kernel function before data arrival but needs to learn kernel classifiers

and their combination weights simultaneously from sequential data. Second, online learning with multiple kernels suffers more from the curse of kernelization because more kernel classifiers getting updated would result in even greater complexity and higher computation cost.

2.5.2 OMKL Algorithms

In literature, existing OMKL algorithms usually adopt one of the two following strategies. The first method is to learn the weights using the Hedge algorithm. Specially, the weight of each kernel is reduced by some factor whenever this single kernel component classifier makes a mistake [32, 26, 53, 73]. Consequently, kernels with higher historical accuracy are assigned with larger weight values. The second method is to add a group sparsity regularizer to the loss function [40, 42, 43, 44]. This then results in a group sparse kernel classifier, where a subset of kernels are assigned with zero weight. These algorithms are efficient since only effective kernels are selected.

Group Sparsity Strategies

To perform efficient kernel selection in the learning process, in literature, many works formulate the above multiple kernel learning problem as an optimization problem with a group sparsity regularizer [2, 42, 59], i.e.

$$\min_{f_1 \in \kappa_1, \ldots, f_i \in \kappa_i, .., f_m \in \kappa_m} \frac{\lambda}{2} (\sum_{i=1}^{m} \|f_i\|_{\mathcal{H}_{k_i}})^2 + \frac{1}{T} \sum_{t=1}^{T} \ell_t (\sum_{i=1}^{m} f_i),$$

where the first term is the squared $L_{2,1}$ norm regularizer and the second term is the averaged loss of the classifier $f = \sum_{i=1}^{m} f_i$ which is the sum of all classifiers in the m kernel spaces. A state-of-the-art work, the Online Proximal MKL algorithm [42], attempts to solve this problem in an online mode. However, like many non-budget kernel learning algorithms, the number of support vectors is not bounded, which limits its application to large scale problems.

Hedge Strategy

The Hedge algorithms have been widely used in learning the combination weights in ensemble models. In a Hedge based OMKL algorithm [26], the weights are initialized to be $\theta_i = 1/m$ for all classifiers. When f_i makes a mistake, its weight will be reduced by a constant factor $\theta_{i,t+1} = \theta_{i,t}\gamma$. As the result, the OMKL algorithm always follows the best kernel functions and is proven to be able to achieve an optimal mistake bound.

To further reduce the number of support vectors, some existing works also adopt a stochastic update strategy and the probability of updating f_i is

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

where $\delta > 0$ is a parameter that controls the trade-off between exploration and exploitation. Obviously, better kernels with higher combination weights get larger chance to be updated. Consequently, we may avoid wasting SV's on worse kernels. This stochastic updating strategy can alleviate the problem of unbounded number of SV's, but there are still some open problems.

First, the update probability only depends on the historical performance of the classifier, which is far from optimal. Since different instances have difference importance level, it is nature to relate the probability of update with the prediction confidence of a certain instance. Second, this stochastic update strategy might be mislead by the instances in the first a few rounds. Consider the case where the best component classifier f_i did not perform well in the first a few rounds. f_i will get less chance to be updated and thus get fewer SV's, which in turn results in bad performance in later rounds. Consequently, the algorithm may be mislead by the first a few instances and fail to catch the best kernel during the whole learning process.

In summary, although existing stochastic OMKL algorithms can reduce the number of SV's, it is still not enough for a powerful OMKL algorithm in large scale applications.

2.6 Summary

In this chapter, we first introduced the main idea of Online Learning. Online Learning algorithms process data sequentially and thus are more suitable for large scale and stream datasets compared to traditional batch learning algorithms. We discussed some of the most representative works in online learning field.

Secondly, a brief introduction of kernel learning was provided. Kernel Learning is a powerful tool to address the linear inseparable problem. By adopting the kernel functions, we can easily get the inner product in the mapped space without explicitly calculation of the feature mapping and product.

Thirdly, by combining the background knowledge of the above two sections, we introduced the Online Kernel Learning algorithms. They enjoy the advantages of both and are suitable for handling sequentially data with nonlinear patterns. However, the unbounded number of SV's limit its application to large scale datasets.

Fourthly, we discussed many existing budget learning algorithms, which aim to address the unbounded number of SV's problem. This discussion of budget algorithms provides a clear answer to the following questions. To make online kernel learning algorithm scalable to large scale applications, what are the state-of-the-art existing algorithms? What are their pros and cons? What are we expecting to achieve when designing a novel large scale online kernel learning algorithm?

Finally, we reviewed the related works for online multiple kernel learning, which motivate our proposed algorithms and will be used as compared algorithms in our experiments.

Chapter 3

Online Sparse Passive Aggressive Learning with Kernels

3.1 Introduction

Due to the curse of kernelization, a major limitation of many kernel-based online learning techniques is that the number of support vectors is unbounded and potentially large for large-scale applications. This has raised a huge challenge for applying them in practical applications since computational complexity (of both time and space) for a kernel-based online learning algorithm is often proportional to the support vector size. Recent years have witnessed a variety of emerging studies for bounded kernel-based online learning. Examples include RBP [3], Forgetron [17], Projectron [45], Budget Passive Aggressive learning [71], BOGD [82, 69], among others.

Although bounded kernel-based online learning has been actively studied, most existing algorithms suffer from a key drawback when applying them in online-to-batch conversion, a process that aims to convert online classifiers for batch classification purposes [18, 15]. Specifically, one of most commonly used approaches in online-to-batch conversion is to take the *averaging classifier*, that is the mean of all the online classifiers at all online iterations, as the final classifier for batch classifi-

cation. This simple technique is not only computationally efficient, but also enjoys theoretical superiority in generalization performance compared to the classifier obtained in the last online iteration [57]. Unfortunately, most existing budget online kernel learning algorithms only guarantee that the support vector size at each online iteration is bounded, but fail to yield a sparse averaging classifier in online-to-batch conversion.

In this section, we present a new method for bounded kernel-based online learning method, named "Sparse Passive Aggressive" (SPA) learning, which extends the online Passive Aggressive (PA) learning method [10] to ensure the final output averaging classifier has the bounded number of support vectors in online-to-batch conversion. Specifically, the basic idea of our method is to explore a simple stochastic sampling rule, which assigns an incoming training example to be a support vector with a higher probability when it suffers a higher loss.

The rest of this section is organized as follows. Subsection 2 formally formulates the problem and then presents the proposed SPA algorithm. Subsection 3 gives theoretical analysis. Subsection 4 presents our experimental studies and empirical observations, and finally Subsection 5 concludes this section.

3.2 Related Work

Our work is closely related to Online Learning, Online Kernel Learning and Budget Learning algorithms. We have made a comprehensive survey in the previous sections.

In addition, our focus is on the sparse solution of kernel learning. Which is also related to two sparse kernel methods for online logistic regression [79, 80]. The main limitation of the existing work is that they only considered the problem settings with several smooth loss functions. This is a relatively strict setting since there are lots of situations (eg. SVM) where nonsmooth loss functions , hinge loss for example, are adopted. Our paper studied the nonsmooth loss setting where the

loss function is not limited by assumptions in [80]. The second improvement of our paper compared to the previous work is the better theoretical analysis, which follows the standard analysis of online learning and thus is simpler and easier to follow. Our analysis also suggests an approach to fix a subtle mistake in the proof of bound in [79].

3.3 Sparse PA Learning with Kernels

In this section, we first formulate the problem setting for online learning with kernels, then review online Passive Aggressive (PA) algorithm [10], and finally present the details of the proposed Sparse PA learning with kernels (SPA) algorithm.

3.3.1 Problem Setting and Preliminaries

We consider the problem of online learning by following online convex optimization settings. Our goal is to learn a function $f : \mathbb{R}^d \to \mathbb{R}$ 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 $y_t \in \mathcal{Y}$. 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 $\ell(f; (\mathbf{x}, y)) : \mathcal{H} \times (\mathbb{R}^d \times \mathcal{Y}) \to \mathbb{R}$ as the loss function that penalizes the deviation of estimating $f(\mathbf{x})$ from observed labels y.

Passive Aggressive (PA) algorithms [10] are a family of margin based online learning algorithms, which can achieve a bound on the cumulative loss comparable with the smallest loss that can be attained by any fixed hypothesis.

Specifically, consider a classification problem, an online PA algorithm sequentially updates the online classifier. At the t-step, the online hypothesis will be updated by the following strategy

$$f_{t+1} = \arg\min_{f \in \mathcal{H}_\kappa} \frac{1}{2}\|f - f_t\|^2_{\mathcal{H}_\kappa} + \eta \ell(f; (\mathbf{x}_t, y_t))$$

where $\eta > 0$. This optimization involves two objectives: the first is to keep the new function close to the old one, while the second is to minimize the loss of the new function on the current example. To simplify the discussion, we denote $\ell_t(f) = \ell(f; (\mathbf{x}_t, y_t))$ throughout the section.

3.3.2 Sparse Passive Aggressive Algorithm

Similar to conventional kernel-based online learning methods, the critical limitation of PA is that it does not bound the number of support vectors, making it very expensive in both computational time and memory cost for large-scale applications. Some existing work has attempted to propose budget PA [71] for learning a bounded kernel classifier at each step using some budget maintenance strategy (e.g., discarding an old SV and replacing it by a new one). Although the kernel classifier is bounded at each step, their approach cannot bound the number of support vectors for the average of classifiers over all learning iterations. This drawback of the existing budgeted kernel algorithms limits their application to online-to-batch conversion tasks where the output final classifier is typically obtained by averaging classifiers at every learning steps. Furthermore, even in pure online setting, the prediction of the new coming instance $\mathbf{x}_t$ using $\frac{1}{t}\sum_{i=1}^{t} f_i$ often outperforms the result using a single classifier f_t. In the proposed algorithm in this section, we aim to overcome the above limitation by proposing a novel Sparse Passive Aggressive (SPA) learning algorithm for learning a sparse kernel classifier which guarantees not only the ratio of support vectors to total received examples is always bounded in expectation, but also the support vector size of the final average classifier is bounded.

Unlike the conventional budget maintenance idea, we propose a stochastic support vector sampling strategy which sequentially constructs the set of support vectors by sampling from the sequence of instances in which an instance added into the support vector set will never be discarded. This ensures that the support vector set of any intermediate classifier is always a subset of the final average classifier. The rest

challenge then is how to design an appropriate sampling strategy so that we ensure that the support vector size of kernel classifiers is always bounded while maximizing the learning accuracy of the classifier. To tackle this challenge, we propose a simple yet effective sampling rule which decides if an incoming instance should be a support vector by performing a Bernoulli trial as follows:

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

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

(i) The probability of making the t-th step update is always less than α/β, which avoids assigning too high probability on a noisy instance.

(ii) An example suffering higher loss has a higher probability of being assigned to the support vector set.

In the above, the first is to guarantee that the ratio of support vectors to total received instances is always bounded in expectation, and the second is to maximize the learning accuracy by adding informative support vectors or equivalently avoid making unnecessary updates. For example, for the extreme case of $\ell_t(f_t) = 0$, we always have $Pr(Z_t = 1) = 0$, which means we never makes an update if an instance does not suffer loss. Different from the existing work where the sampling probability is related to the derivative of classification loss, our probability is directly based on the scale of hinge loss.

After obtaining the random variable Z_t, we will need to develop an effective strategy for updating the classifier. Following the PA learning principle, we propose the following updating method:

$$f_{t+1} = \arg\min_{f \in \mathcal{H}_\kappa} P_t(f) := \frac{1}{2}\|f - f_t\|^2_{\mathcal{H}_\kappa} + \frac{Z_t}{\rho_t}\eta\ell_t(f) \tag{3.1}$$

Note when $\rho_t = 0$, then $Z_t = 0$ and we set $Z_t/\rho_t = 0$. We adopt the above update because its objective is an unbiased estimate of that of PA update, that is,

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

Passive Aggressive based algorithms enjoy lots of advantages compared to gradient based ones. For instance, PA updating strategy is more effective because of the adaptive step size. And PA is less sensitive to the variance of parameter setting. Finally, we summarize the proposed Sparse Passive Aggressive algorithm in Algorithm 14.

Algorithm 7 Sparse PA learning with kernels (**SPA**)

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

3.3.3 Application to Binary Classification

The proposed SPA method is a generic online learning framework that can be applied to various online learning tasks. Examples include online classification, regression, and uniclass prediction tasks. Without loss of generality, we focus on the discussion on the application of the proposed algorithm for online binary classification task.

Specifically, we consider an online classification task with label set $\mathcal{Y} = \{-1, +1\}$, and adopt the widely used hinge loss function:

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

where $[z]_+ = \max(0, z)$. The hinge loss function is $\kappa(\mathbf{x}, \mathbf{x})$-Lipschitz with respect to f:

$$|\ell(f; (\mathbf{x}, y)) - \ell(g; (\mathbf{x}, y))| = |[1 - yf(\mathbf{x})]_+ - [1 - yg(\mathbf{x})]_+|$$
$$\leq |yf(\mathbf{x}) - yg(\mathbf{x})| \leq \sqrt{\kappa(\mathbf{x}, \mathbf{x})}\|f - g\|_{\mathcal{H}_\kappa},$$

where we used the fact $[1-z]_+$ is 1-Lipschitz with respect to z. This further implies the corresponding $\ell_t(f)$s are X-Lipschitz, since $\kappa(\mathbf{x}_t, \mathbf{x}_t) \leq X^2$. This fact will be used in the theoretical analysis in the next section.

After choosing the hinge loss, the rest is how to solve the optimization (3.1). Fortunately, we can derive a closed-form solution, as shown in the following proposition.

Proposition 1. *When $\ell_t(f) = [1 - y_t f(\mathbf{x}_t)]_+$, the optimization* (3.1) *enjoys the following closed-form solution*

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

This proposition is easy to prove, so we omit its proof.

3.4 Theoretical Analysis

SPA algorithm will be analyzed for binary classification case. To facilitate the discussion, we denote

$$f_* = \arg\min_f \sum_{t=1}^{T} \ell_t(f)$$

The following theorem analyzes the regret of SPA, i.e., $\sum_{t=1}^{T} \ell_t(f_t) - \sum_{t=1}^{T} \ell_t(f_*)$, which is a main performance measure of online algorithms.

Theorem 1. *Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of examples where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ for all t. If we assume $\kappa(\mathbf{x}, \mathbf{x}) = 1$ and the hinge loss function*

$\ell_t(\cdot)$ *is* 1*-Lipschitz, then for any* $\beta \geq \alpha > 0$*, and* $\eta > 0$*, the proposed SPA algorithm satisfies the following inequality*

$$\mathbb{E}[\sum_{t=1}^{T}(\ell_t(f_t) - \ell_t(f_*))] < \frac{1}{2\eta}\|f_*\|_{\mathcal{H}_\kappa}^2 + \frac{\eta\beta}{\min(\alpha, \sqrt{\beta\eta})}T$$

where η *is the aggressiveness parameter. When setting* $\eta = \|f_*\|_{\mathcal{H}_\kappa}\sqrt{\frac{\alpha}{2\beta T}}$ *and* $\alpha^3 \leq \frac{\beta}{2T}\|f_*\|_{\mathcal{H}_\kappa}^2$*, we will have*

$$\mathbb{E}[\sum_{t=1}^{T}(\ell_t(f_t) - \ell_t(f_*))] < \|f_*\|_{\mathcal{H}_\kappa}\sqrt{2\beta T/\alpha}$$

Proof. Firstly, the $P_t(f)$ defined in the equality

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

is 1-strongly convex. Further, f_{t+1} is the optimal solution of $\min_f P_t(f)$, we thus have the following inequality according the definition of strongly convex

$$\begin{aligned}&\frac{1}{2}\|f - f_t\|_{\mathcal{H}_\kappa}^2 + \frac{1}{\rho_t}Z_t\eta\ell_t(f)\\ &\geq \frac{1}{2}\|f_{t+1} - f_t\|_{\mathcal{H}_\kappa}^2 + \frac{1}{\rho_t}Z_t\eta\ell_t(f_{t+1}) + \frac{1}{2}\|f - f_{t+1}\|_{\mathcal{H}_\kappa}^2\end{aligned}$$

where the inequality used $\nabla P_t(f_{t+1}) = 0$. After rearranging the above inequality, we get

$$\begin{aligned}&\frac{1}{\rho_t}Z_t\eta\ell_t(f_{t+1}) - \frac{1}{\rho_t}Z_t\eta\ell_t(f)\\ &\leq \frac{1}{2}\|f - f_t\|_{\mathcal{H}_\kappa}^2 - \frac{1}{2}\|f - f_{t+1}\|_{\mathcal{H}_\kappa}^2 - \frac{1}{2}\|f_{t+1} - f_t\|_{\mathcal{H}_\kappa}^2\end{aligned}$$

Secondly, since $\ell_t(f)$ is 1-Lipshitz with respect to f

$$\ell_t(f_t) - \ell_t(f_{t+1}) \leq \|f_t - f_{t+1}\|_{\mathcal{H}_\kappa}.$$

Combining the above two inequalities, we get

$$\begin{aligned}&\frac{1}{\rho_t}Z_t\eta[\ell_t(f_t) - \ell_t(f)]\\ \leq&\frac{1}{2}\|f - f_t\|_{\mathcal{H}_\kappa}^2 - \frac{1}{2}\|f - f_{t+1}\|_{\mathcal{H}_\kappa}^2 - \frac{1}{2}\|f_{t+1} - f_t\|_{\mathcal{H}_\kappa}^2 + \frac{1}{\rho_t}Z_t\eta\|f_t - f_{t+1}\|_{\mathcal{H}_\kappa}\end{aligned}$$

Summing the above inequalities over all t leads to

$$\sum_{t=1}^{T} \frac{1}{\rho_t} Z_t \eta \left[\ell_t(f_t) - \ell_t(f)\right] \leq \frac{1}{2}\|f - f_1\|_{\mathcal{H}_\kappa}^2 \tag{3.2}$$
$$+ \sum_{t=1}^{T} \left[-\frac{1}{2}\|f_{t+1} - f_t\|_{\mathcal{H}_\kappa}^2 + \frac{1}{\rho_t} Z_t \eta \|f_t - f_{t+1}\|_{\mathcal{H}_\kappa}\right]$$

We now take expectation on the left side. Note, by definition of the algorithm, $\mathbb{E}_t Z_t = \rho_t$, where we used $\mathbb{E}_t$ to indicate conditional expectation give all the random variables $Z_1, \ldots, Z_{t-1}$. Assuming $\rho_t > 0$, we have

$$\begin{aligned} &\mathbb{E}\big[\frac{1}{\rho_t} Z_t \eta \left[\ell_t(f_t) - \ell_t(f)\right]\big] \\ &= \mathbb{E}\big[\frac{1}{\rho_t} \mathbb{E}_t Z_t \eta \left[\ell_t(f_t) - \ell_t(f)\right]\big] = \eta \mathbb{E}\left[\ell_t(f_t) - \ell_t(f)\right] \end{aligned} \tag{3.3}$$

Note that in some iterations, $\rho_t = 0$, in that case, we have $\ell_t(f_t) = 0$, thus:

$$\eta[\ell_t(f_t) - \ell_t(f)] \leq 0 \tag{3.4}$$

As mentioned before, $\rho_t = 0$ indicates $Z_t = 0$ and $Z_t/\rho_t = 0$, we get

$$\frac{1}{\rho_t} Z_t \eta[\ell_t(f_t) - \ell_t(f)] = 0 \tag{3.5}$$

Combining (3.3), (3.4) and (3.5) and summarizing over all t leads to

$$\eta \mathbb{E} \sum_{t=1}^{T} [\ell_t(f_t) - \ell_t(f)] \leq \mathbb{E} \sum_{t=1}^{T} \frac{1}{\rho_t} Z_t \eta \left[\ell_t(f_t) - \ell_t(f)\right]$$

We now take expectation on the right side of (3.2)

$$\begin{aligned} &\mathbb{E}\left[\frac{1}{2}\|f - f_1\|_{\mathcal{H}_\kappa}^2\right] + \mathbb{E}\left[\sum_{t=1}^{T}\left[-\frac{1}{2}\|f_{t+1} - f_t\|_{\mathcal{H}_\kappa}^2 + \frac{1}{\rho_t} Z_t \eta \|f_t - f_{t+1}\|_{\mathcal{H}_\kappa}\right]\right] \\ &\leq \frac{1}{2}\|f\|_{\mathcal{H}_\kappa}^2 + \sum_{t=1}^{T} \mathbb{E}\left[-\frac{1}{2}\tau_t^2 + \frac{1}{\rho_t} Z_t \eta \tau_t\right] \end{aligned}$$

Given all the random variables $Z_1, \ldots, Z_{t-1}$, we now calculate the conditional expectation of the variable $M_t = -\frac{1}{2}\tau_t^2 + \frac{1}{\rho_t} Z_t \eta \tau_t$: In probability ρ_t, $Z_t = 1$ and $\tau_t = \tau_t' = \min(\frac{\eta}{\rho_t}, \ell_t(f_t))$. We have $M_{t\ (Z_t=1)} = -\frac{1}{2}\tau_t'^2 + \frac{1}{\rho_t}\eta\tau_t'$. And in probability $1 - \rho_t$, $Z_t = 0$ and $\tau_t = 0$. We have $M_{t\ (Z_t=0)} = 0$. Considering the two cases, the

conditional expectation is:

$$\mathbb{E}_t[M_t] = \rho_t M_{t\ (Z_t=1)} + (1-\rho_t) M_{t\ (Z_t=0)} = \rho_t \left[-\frac{1}{2}\tau_t'^2 + \frac{1}{\rho_t}\eta\tau_t' \right] < \eta\tau_t'$$

In the case when $\alpha \leq \ell_t$ and $\rho_t = \frac{\alpha}{\beta}$, $\tau_t' = \min(\frac{\eta\beta}{\alpha}, \ell_t(f_t)) \leq \frac{\eta\beta}{\alpha}$, thus $\eta\tau_t' \leq \frac{\eta^2\beta}{\alpha}$.

And in the case $\alpha > \ell_t$ and $\rho_t = \frac{\ell_t}{\beta}$, $\tau_t' = \min(\frac{\eta\beta}{\ell_t(f_t)}, \ell_t(f_t)) \leq \sqrt{\eta\beta}$. Thus, $\eta\tau_t' \leq \frac{\eta^2\beta}{\sqrt{\beta\eta}}$.

Considering both of the cases leads to

$$\mathbb{E}_t[M_t] < \frac{\eta^2\beta}{\min(\alpha, \sqrt{\beta\eta})}$$

Summing the above inequality over all t and combining with (3.6), we get

$$\eta\mathbb{E}\sum_{t=1}^{T}[\ell_t(f_t) - \ell_t(f)] < \frac{1}{2}\|f\|_{\mathcal{H}_\kappa}^2 + \frac{\eta^2\beta}{\min(\alpha, \sqrt{\beta\eta})}T$$

Setting $f = f_*$, and multiplying the above inequality with $1/\eta$ will conclude the theorem. □

Remark 1. The theorem indicates that the expected regret of the proposed SPA algorithm can be bounded by $\|f_*\|_{\mathcal{H}_\kappa}\sqrt{2\beta T/\alpha}$ in expectation. In practice, β/α is usually a small constance. Thus, we can conclude that the proposed algorithm can achieve a strong regret bound in expectation.

Remark 2. The expected regret has a close relationship with the two parameters α and β. As indicated above, to avoid assigning too high probability on a noisy instance, the parameter α can not be too large. Assuming $\alpha \leq \sqrt{\beta\eta}$ (which is accessible in the practical parameter setting), the expected regret bound is proportion to the ratio β/α. This consists with the intuition that larger chances of adding SVs leads to smaller loss. Further more, for $\alpha > \sqrt{\beta\eta}$, the expected regret bound is less tight than the above case, which consists with the analysis before that too large α involves large number of noisy instances and might be harmful.

Next, we would give a bound on the number of support vectors of the final classifier $\bar{f}_T$ in expectation. Since we never delete an existing support vector, it should be the same with the number of support vectors of the final intermediate

classifier f_T, which equals to the random number $\sum_{t=1}^{T} Z_t$.

Theorem 2. *Let* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *be a sequence of examples where* $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ *for all* t. *If we assume* $\kappa(\mathbf{x}, \mathbf{x}) = 1$ *and the hinge loss function* $\ell_t(\cdot)$ *is* 1*-Lipschitz, then for any* $\beta \geq \alpha > 0$, *and* $\eta > 0$, *the proposed SPA algorithm satisfies the following inequality*

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

Especially, when $\eta = \|f_*\|_{\mathcal{H}_\kappa} \sqrt{\frac{\alpha}{2\beta T}}$ *and* $\alpha^3 \leq \frac{\beta}{2T} \|f_*\|_{\mathcal{H}_\kappa}^2$, *we have*

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

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

$$\begin{aligned}
&\mathbb{E}[\sum_{t=1}^{T} Z_t] = \mathbb{E}[\sum_{t=1}^{T} \mathbb{E}_t Z_t] = \mathbb{E}[\sum_{t=1}^{T} \rho_t] = \\
&\mathbb{E}[\sum_{t=1}^{T} \min(\frac{\alpha}{\beta}, \frac{\ell_t(f_t)}{\beta})] \leq \min(\frac{\alpha}{\beta} T, \frac{1}{\beta} \mathbb{E} \sum_{t=1}^{T} \ell_t(f_t)) \leq \\
&\min \left\{ \frac{\alpha}{\beta} T, \frac{1}{\beta} [\sum_{t=1}^{T} \ell_t(f_*) + \frac{1}{2\eta} \|f_*\|_{\mathcal{H}_\kappa}^2 + \frac{\eta\beta}{\min(\alpha, \sqrt{\beta\eta})} T] \right\}
\end{aligned}$$

which concludes the first part of the theorem. The second part of the theorem is trivial to be derived. □

Remark. First, this theorem indicates the expected number of support vectors is less than $\alpha T/\beta$. Thus, by setting $\beta \geq \alpha T/n$ $(1 < n \leq T)$, we guarantee that the expected number of support vectors of the final classifier is bounded by a budget n. Second, this theorem also implies that, by setting $\beta \geq [\sum_{t=1}^{T} \ell_t(f_*) + \|f_*\|_{\mathcal{H}_\kappa} \sqrt{2\beta T/\alpha}]/n$ $(1 < n \leq T)$, the expected number of support vectors is always less than n, no matter how is the value of α.

3.5 Experiments

In this section, we conduct extensive experiments to evaluate the empirical performance of the proposed SPA algorithm for online binary classification tasks.

3.5.1 Experimental Testbed

Table 5.1 summarizes details of some binary classification datasets in our experiments. The first five can be downloaded from LIBSVM[1]or KDDCUP competition site [2]. The original SUSY dataset contains 5,000,000 instances, we randomly sampled a subset.

Table 3.1: Summary of Binary Classification Datasets Used in SPA Evaluation.

DATA SET	# INSTANCES	#FEATURES
KDD08	102,294	117
A9A	48,842	123
W7A	49,749	300
CODRNA	59,535	8
COVTYPE	581,012	54
SUSY	1,000,000	18

3.5.2 Compared Algorithms and Setup

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

- "Perceptron": the kernelized Perceptron [22];
- "OGD": the kernelized OGD algorithm [33];
- "PA-I": the kernelized passive aggressive algorithm with soft margin [10].

Further, we compare our SPA algorithm with a variety of budget online kernel learning algorithms:

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

- "RBP": the Randomized Budget Perceptron by random removal [3];
- "Forgetron": the Forgetron by discarding oldest support vectors [17];
- "Projectron": the Projectron algorithm using the projection strategy [45];
- "Projectron++": the aggressive version of Projectron algorithm [45];
- "BPA-S": the Budget Passive-Aggressive algorithm [71], we only adopt the BPA-Simple algorithm since the other two variants are too computational expensive to scale to large datasets;
- "BOGD": the Bounded Online Gradient Descent algorithm [82, 69];
- "OSKL":the Online Sparse Kernel Learning algorithm [80].

To make a fair comparison, we adopt the same experimental setup for all the algorithms. We use the hinge loss for gradient-based algorithms (OGD and BOGD). For all the algorithms, we adopt a gaussian kernel $\exp(-\gamma||x_1 - x_2||^2)$ with parameter γ set to 0.4 for all the datasets. The learning rate parameter η for OGD, BOGD, OSKL and SPA is automatically chosen by searching from $\{10^3, ..., 10^{-3}\}$ based on a random permutation of each dataset. We adopt the PA-I algorithm for comparison since it was proved to be robust to noise and achieved better performance than PA without soft margin. The soft margin parameter C is optimized by searching from range $\{0.25,\ 0.5,\ 1,\ 2\}$. In the proposed algorithm, the parameter $\alpha = 1$ for all datasets. We choose a proper β parameter so that the resulting support vector size is roughly a proper fraction of that of the non-budget OGD algorithm (specifically, we set $\beta = 20$ for three small datasets, and $\beta = 200$ for three large datasets). Note that the OSKL algorithm also adopts a stochastic approach to sample the support vectors, which indicates that there is no ideal method for setting the same number of support vectors and thus absolutely fair comparison between OSKL and our proposed algorithm. We tune the sampling probability parameter G in the OSKL algorithm so that the averaged number of support vectors is close to that of the proposed algorithm. Finally, to ensure that all budget algorithms adopt the same budget

size, we choose the budget size B of all the other compared algorithms according to the average number of support vectors generated by the proposed SPA algorithm.

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

3.5.3 Evaluation of Online Learning Performance

The first experiment is to evaluate the performance of kernel-based online learning algorithms for online classification tasks. Table 5.2 shows the experimental results. To demonstrate the superiority of averaging classifier in pure online setting, the reported accuracy of the proposed SPA algorithm and OSKL is obtained by the averaged classifier $\frac{1}{t}\sum_{i=1}^{t} f_i$. We can draw some observations as follows.

We first examine the time cost comparisons. First of all, all the budget algorithms are significantly more efficient than the non-budget algorithms (Perceptron, OGD, and PA-I) for most cases, especially on large-scale datasets. This validates the importance of studying bounded kernel-based online learning algorithms. Second, by comparing different budget algorithms, Projectron++ is the least efficient due to its expensive projection strategy, and the proposed SPA algorithm is the most efficient. The other budget algorithms (RBP, Forgetron, BPAS, BOGD) are in general quite efficient as they all are based on simple SV removal strategy for budget maintenance. The reason that our SPA algorithm is even more efficient than these algorithms is because our algorithm adds the support vectors incrementally while the other algorithms perform the budget maintenance only when the support vector size reaches the budget. This encouraging result validates the high efficiency advantage of our stochastic support vector sampling strategy.

Table 3.2: Evaluation of Online Kernel Classification on Six Datasets in SPA Evaluation. Time in seconds

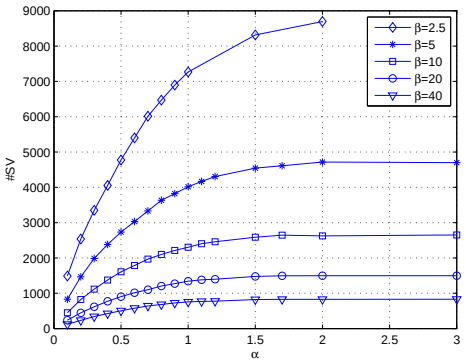
	KDD08, $\beta = 20$			a9a, $\beta = 20$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	99.04$_{\pm 0.01}$	11.58	0.9k	79.40$_{\pm 0.11}$	19.93	9.9k
OGD	99.44$_{\pm 0.01}$	14.54	1.2k	83.41 $_{\pm 0.05}$	54.20	23.5k
PA-I	99.45$_{\pm 0.01}$	39.53	3.2k	84.07$_{\pm 0.08}$	57.91	22.8k
RBP	98.96$_{\pm 0.03}$	3.24	149	78.83$_{\pm 0.38}$	5.34	1,350
Forgetron	98.93$_{\pm 0.03}$	3.28	149	78.08$_{\pm 0.22}$	6.45	1,350
Projectron	99.02$_{\pm 0.01}$	4.19	149	79.25$_{\pm 0.13}$	32.47	1,350
Projcetron++	99.32$_{\pm 0.01}$	4.86	149	79.35$_{\pm 0.13}$	150.63	1,350
BPAS	**99.41$_{\pm 0.01}$**	3.93	149	80.44$_{\pm 0.14}$	7.75	1,350
BOGD	99.39$_{\pm 0.01}$	3.40	149	80.97$_{\pm 0.04}$	6.17	1,350
OSKL	**99.41$_{\pm 0.01}$**	**2.63**	149	82.01$_{\pm 0.32}$	4.08	1,344
SPA	**99.41$_{\pm 0.01}$**	**2.60**	149	**82.04$_{\pm 0.25}$**	**3.84**	1,350
	w7a, $\beta = 200$			codrna, $\beta = 20$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	97.64 $_{\pm 0.04}$	2.50	1.1k	90.79 $_{\pm 0.09}$	6.21	5.4k
OGD	98.06$_{\pm 0.02}$	38.16	10.1k	93.31$_{\pm 0.05}$	10.41	8.8k
PA-I	98.16$_{\pm 0.02}$	63.47	19.6k	93.65$_{\pm 0.05}$	15.82	12.4k
RBP	96.78$_{\pm 0.16}$	0.68	175	86.59$_{\pm 0.22}$	1.68	822
Forgetron	96.36$_{\pm 0.06}$	0.72	175	86.62$_{\pm 0.15}$	1.91	822
Projectron	95.19$_{\pm 0.39}$	0.87	175	83.35$_{\pm 0.23}$	19.38	822
Projcetron++	95.90$_{\pm 0.38}$	3.01	175	84.71$_{\pm 0.19}$	32.99	822
BPAS	96.83$_{\pm 0.23}$	1.77	175	91.23$_{\pm 0.05}$	2.21	822
BOGD	96.75$_{\pm 0.03}$	1.05	175	85.62$_{\pm 0.06}$	1.92	822
OSKL	96.98$_{\pm 0.43}$	0.71	177	**92.07$_{\pm 0.37}$**	1.37	825
SPA	**97.05$_{\pm 0.13}$**	**0.56**	175	91.59$_{\pm 0.35}$	**1.31**	822
	covtype, $\beta = 200$			SUSY, $\beta = 200$		
Algorithm	Accuracy (%)	Time	#SVs	Accuracy (%)	Time	#SVs
Perceptron	73.08$_{\pm 0.03}$	2861	156.0k	71.69$_{\pm 0.04}$	7298	283.2k
OGD	78.34$_{\pm 0.03}$	5113	296.7k	78.97$_{\pm 0.01}$	13648	491.7k
PA-I	78.18$_{\pm 0.05}$	4402	298.2k	78.96$_{\pm 0.01}$	14715	507.2k
RBP	65.63$_{\pm 0.23}$	50.27	1,510	66.44$_{\pm 0.26}$	96.7	1933
Forgetron	65.33$_{\pm 0.22}$	67.60	1,510	66.31$_{\pm 0.33}$	131.5	1933
Projectron	57.82$_{\pm 6.09}$	316.05	1,510	54.46$_{\pm 0.10}$	4523.0	1933
Projcetron++	59.33$_{\pm 5.85}$	470.35	1,510	60.01$_{\pm 1.62}$	3738.6	1933
BPAS	**72.37$_{\pm 0.13}$**	73.48	1,510	74.85$_{\pm 0.07}$	157.4	1933
BOGD	68.75$_{\pm 0.03}$	54.44	1,510	68.75$_{\pm 0.03}$	105.0	1933
OSKL	72.11$_{\pm 0.67}$	29.77	1,508	73.57$_{\pm 0.47}$	74.3	1957
SPA	71.34$_{\pm 0.59}$	**27.53**	1,510	**80.04$_{\pm 0.34}$**	**49.2**	1933

We then compare online classification accuracy of different algorithms. First, the non-budget algorithms have better accuracy than their budget variants. This is not surprising as the non-budget algorithms use a larger SV size. Second, comparing different budget algorithms, we found that PA and OGD based algorithms (BPAS, BOGD, and SPA) generally outperform Perceptron-based algorithms (RBP, Forgetron, Projectron, and Projectron++). Finally, our SPA algorithm achieves the best accuracy among all the budget algorithms most of the cases and is comparable to the OSKL algorithm. These results again validate the effectiveness of the proposed budget learning strategy.

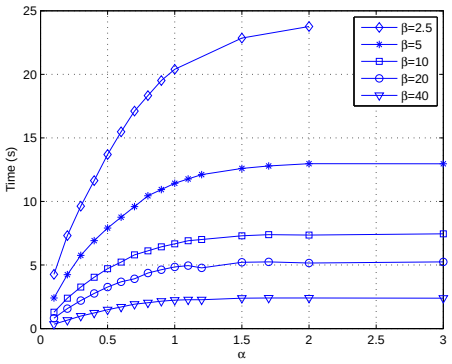
3.5.4 Parameter Sensitivity of α and β in SPA Evaluation

The proposed SPA algorithm has two critical parameters α and β which could considerably affect the accuracy, support vector size and time cost. Our second experiment is to examine how different parameters of α and β affect the learning performance so as to give insights for how to choose them in practice. Figure 3.1 evaluates the performance (support vector size, time and accuracy) of the SPA algorithm on the "a9a" dataset with varied α values ranging from 0.1 to 3, and varied β in the range of $\{2.5, 5, 10, 20, 40\}$. Several observations can be drawn from the experimental results.

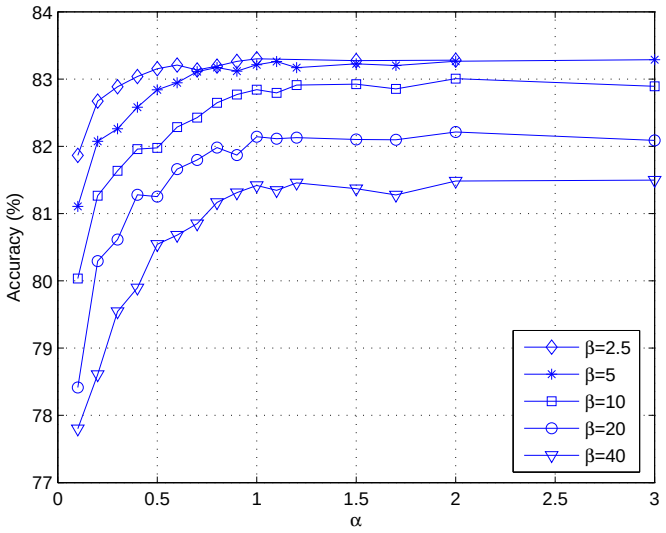
First of all, when β is fixed, increasing α generally results in (i) larger support vector size, (ii) higher time cost, but (iii) better classification accuracy, especially when α is small. However, when α is large enough (e.g., $\alpha > 1.5$), increasing α has very minor impact to the performance.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



(a) Number of support vectors



(b) Time Cost (seconds)



(c) Accuracy (%)

Figure 3.1: The Impact of α and β on #SV, Time Cost and Accuracy by the Proposed SPA Algorithm on "a9a".

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

3.5.5 Evaluation of Output Classifiers on Test Data

Table 3.3: Evaluation of Final Classifiers for Test Data in SPA Evaluation (Time in Seconds) .

	a9a			codrna		
Algorithm	Acc. (%)	Time	#SV	Acc. (%)	Time	#SV
LIBSVM	85.04	97.8	11.5k	96.67	80.2	8.0k
Pegasos	84.66$\pm$0.21	15.8	11.0k	95.63$\pm$0.43	14.9	12.2k
BOGD	82.49$\pm$1.22	5.85	2,079	92.10$\pm$1.32	5.34	2,386
BPAS	82.11$\pm$0.83	7.59	2,079	93.12$\pm$2.01	6.98	2,386
OSKL-last	80.17$\pm$3.91	3.19	2,073	94.15$\pm$1.66	3.42	2,410
OSKL-avg	**84.91$\pm$0.13**	3.19	2,073	95.67$\pm$0.13	3.42	2,410
SPA-last	82.97$\pm$2.59	**3.16**	2,079	91.23$\pm$3.40	**3.03**	2,386
SPA-avg	84.88$\pm$0.10	**3.16**	2,079	**96.05$\pm$0.09**	**3.03**	2,386
	KDD08			w7a		
Algorithm	Acc. (%)	Time	#SV	Acc. (%)	Time	#SV
LIBSVM	99.39	577.3	14.6k	98.31	92.61	8.0k
Pegasos	99.50$\pm$0.01	976.8	81.8k	97.56$\pm$0.01	29.21	24.7k
BOGD	99.37$\pm$0.01	31.84	1,760	97.05$\pm$0.01	8.33	3,710
BPAS	99.22$\pm$0.29	37.48	1,760	97.75$\pm$0.40	13.95	3,710
OSKL-last	99.20$\pm$0.01	22.79	1,752	97.92$\pm$0.20	5.08	3,726
OSKL-avg	99.20$\pm$0.01	22.79	1,752	97.94$\pm$0.07	5.08	3,726
SPA-last	99.41$\pm$0.07	**19.42**	1,760	97.49$\pm$2.05	**5.04**	3,710
SPA-avg	**99.46$\pm$0.01**	**19.42**	1,760	**97.97$\pm$0.04**	**5.04**	3,710

A key advantage of SPA is that it assures the final output averaged classifier is sparse, which is very important when applying the output classifier in real applications. This experiment thus is to examine if the final output classifier of SPA is effective for batch classification. We evaluate two SPA classifiers: "SPA-last" that simply outputs the classifier at the last iteration as the final classifier f_T, and

"SPA-avg" that outputs the average of classifiers at every iteration, $\frac{1}{T}\sum_{t=1}^{T} f_t$.

We compare our algorithms with two state-of-the-art batch algorithms for SVM classifications: (1) LIBSVM: a widely used and the most accurate solution of kernel SVM for batch classification [5]; (2) Pegasos[3][56]: an efficient stochastic gradient descent solver for SVM, for which we adapt it for kernel learning. Moreover, we compare our solutions with the output classifiers by two bounded kernel-based online online algorithms: BOGD and BPAS, as they achieve the best among all the existing algorithms in previous experiments. For these two algorithms, as their average classifier is not sparse, we compare with their last classifiers.

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

Table 4.8 shows the results, where we only report the test set accuracy and training time (we exclude the test time as it is proportional to SV sizes). We can draw some observations. First of all, in terms of training time, the budget algorithms run much faster than LIBSVM and Pegasos, in which our SPA algorithm achieves the lowest training time among all. Specifically, compared to batch algorithms, SPA achieves the speedup of training time for about 20 to 30 times over LIBSVM, and about 5 times over Pegasos.

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

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

Third, we notice that BOGD, BPAS and SPA-last achieve similar test set accuracy, but their standard deviations are in general much larger than that of SPA-avg for most cases. This shows that employing the averaged classifier with SPA for test data classification leads to more accurate and more stable performance than many budget learning algorithms that output the last classifier, which again validates the advantage of our technique.

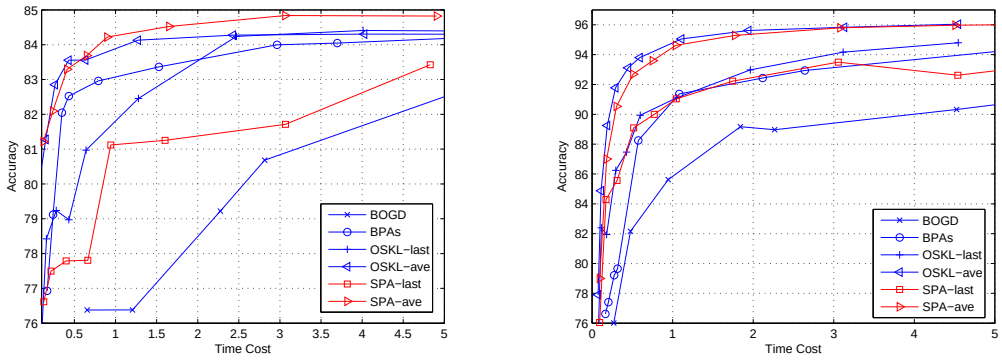
3.5.6 Experiments on Various Budget Sizes

In the previous experiments we compared many state-of-the-art online budget learning algorithms under fixed budget size. To be more convincing, in this subsection, we show the "time vs accuracy" of all compared algorithms under varied SV size setting in Figure 3.2.

The train and test splitting and parameter setting are identical to that used in Table 3.3, except the β and B. We adopt various β and B values to show the different performance of all algorithms under various SV's set size. Finally, we plot the accuracy on the same time axis for fair comparison. Several observation can be drawn as follows:

First, for all algorithms, more support vectors (larger B, smaller β) result in higher accuracy. Consequently, it is important for the learner to choose a proper trade-off between efficiency and accuracy. Second, when comparing the "averaged classifiers" (SPA-ave and OSKL-ave) and their corresponding "last classifiers", we find that the averaged classifiers always achieve higher accuracy which validates our

main motivation. Third, by comparing the accuracy between different algorithms, we find our proposed algorithm, SPA-ave gets better accuracy than most of the compared algorithms and sometimes comparable to that of OSKL. These observations consist with that in Table 3.3.



(a) a9a (b) codrna

Figure 3.2: Evaluation of different budget single kernel algorithms on a9a and codrna data sets. The curves of online mistake rates vs time costs were obtained by choosing varied budget values.

3.6 Discussion

To overcome the curse of kernelization in large-scale kernel methods, this section proposed a novel framework of Sparse Passive Aggressive algorithm for bounded kernel-based online learning. In contrast to traditional budget kernel algorithms that only bound the number of support vectors at each iteration, our algorithm can output a sparse final averaged classifier with bounded support vector size, making it not only suitable for online learning tasks but also applicable to batch classification tasks. The experimental results from both online and batch classification tasks showed that the proposed method achieved a significant speedup over LIBSVM and yielded sparse and more accurate classifiers than existing online kernel methods, validating the efficacy, efficiency and scalability of the proposed technique.

Chapter 4

Online Kernel Learning by Functional Approximation Methods

4.1 Introduction

In this section, 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" [49, 10, 19], 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 only 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" [33, 22], 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 the un-

bounded number of SV's. To address this challenge, a promising research direction is to explore "budget online kernel learning" [11], as introduced in section 2.4. 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.

Unlike the existing budget online kernel learning methods, in this section, 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.

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

4.2 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 [49, 22, 10, 81, 68, 30], 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*. They discard a SV when the number of SV's reaches the budget. Some well-known examples include RBP [3], Forgetron [16], BOGD and BPA-S [71].

- *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 [45], BPA-P, and BPA-NN [71]. 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 [70]. The similar idea of SV merging was also explored to bound the number of SV's in BSGD-M algorithm in [69].

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.

Moreover, our work is also related to kernel methods for classification tasks [27, 29], especially for some studies on large-scale kernel methods [72, 47].

Our approach shares the similar idea with the Low-rank Linearization SVM (LLSVM) [78], 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 [47], which have been successfully explored for speeding up batch kernelized SVMs [78] and kernel-based clustering [7, 8] tasks. Besides, another kernel approximation technique used in our approach is the well-known Nyström method [72, 38], which has been widely applied in machine learning tasks, including Gaussian Processes [72], Kernelized SVMs [78], Kernel PCA, Spectral Clustering [77], and manifold learning [61]. 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 section. Finally, we note that the short version of this work had been published in IJCAI2013 [66]. This version has made significant extension by including substantial amount of new contents and more extensive empirical studies.

4.3 Binary Classification

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

4.3.1 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where instance $\mathbf{x}_t \in \mathbb{R}^d$ and class label $y_t \in \mathcal{Y} = \{+1, -1\}$. We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \rightarrow \mathbb{R}\}$. We will use $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \rightarrow \mathbb{R}$ as the loss function that penalizes the deviation of estimating $f(\mathbf{x})$ from observed labels y.

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

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

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

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

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), α_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.

4.3.2 Fourier Online Gradient Descent

Random Fourier features can be used in shift-invariant kernels [47]. 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:

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u}, \tag{4.1}$$

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

$$p(\mathbf{u}) = (\frac{1}{2\pi})^d \int e^{-i\mathbf{u}^\top(\Delta\mathbf{x})} k(\Delta\mathbf{x}) d(\Delta\mathbf{x}). \tag{4.2}$$

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 [52], the kernel function can be expressed as an expectation of function with a random variable $\mathbf{u}$:

$$\begin{aligned}
&\int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1-\mathbf{x}_2)} d\mathbf{u} = \mathrm{E}_{\mathbf{u}}[e^{i\mathbf{u}^\top\mathbf{x}_1} \cdot e^{-i\mathbf{u}^\top\mathbf{x}_2}] \\
&= \mathrm{E}_{\mathbf{u}}[\cos(\mathbf{u}^\top\mathbf{x}_1)\cos(\mathbf{u}^\top\mathbf{x}_2) + \sin(\mathbf{u}^\top\mathbf{x}_1)\sin(\mathbf{u}^\top\mathbf{x}_2)] \\
&= \mathrm{E}_{\mathbf{u}}[[\sin(\mathbf{u}^\top\mathbf{x}_1), \cos(\mathbf{u}^\top\mathbf{x}_1)] \cdot [\sin(\mathbf{u}^\top\mathbf{x}_2), \cos(\mathbf{u}^\top\mathbf{x}_2)]].
\end{aligned} \tag{4.3}$$

The equality (4.1) can be obtained by only keeping the real part of the complex function. From (4.3), 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 8.

Algorithm 8 FOGD — Fourier Online Gradient Descent for Binary Classification

Input: the number of Fourier components D, step size η, kernel function k;
Initialize $\mathbf{w}_1 = \mathbf{0}$.
Calculate $p(\mathbf{u})$ for kernel k as (4.2).
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 $\widehat{y}_t = \text{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t))$;
  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 9 NOGD — Nyström Online Gradient Descent for Binary Classification

Input: the budget B, step size η, 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 $\widehat{y}_t = \text{sgn}(f_t(\mathbf{x}_t))$;
  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 $\widehat{y}_t = \text{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t))$;
  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

4.3.3 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 [72] 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

$$\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^+\mathbf{C}^\top \approx \mathbf{K}, \tag{4.4}$$

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 (4.4), we could see the kernel value between i-th instance $\mathbf{x}_i$ and j-th instance $\mathbf{x}_j$ 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 9. 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

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

4.3.4 Theoretical Analysis

In this section, we analyze the theoretical properties of the two proposed algorithms.

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 8. 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 [54], 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:

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

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 [47]), 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:

$$\|\mathbf{w}^*\|^2 \leq (1+\epsilon)\|f^*\|_1^2. \tag{4.5}$$

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

$$\begin{aligned}
|\sum_{t=1}^{T}\ell_t(\mathbf{w}^*) - \sum_{t=1}^{T}\ell_t(f^*)| &\leq \sum_{t=1}^{T}|\ell_t(\mathbf{w}^*) - \ell_t(f^*)| \\
&\leq \sum_{t=1}^{T}L\sum_{i=1}^{T}|\alpha_i^*||\mathbf{z}(\mathbf{x}_i)^\top\mathbf{z}(\mathbf{x}_t) - \kappa(\mathbf{x}_i, \mathbf{x}_t)| \\
&\leq \sum_{t=1}^{T}L\epsilon\sum_{i=1}^{T}|\alpha_i^*| = \epsilon LT\|f^*\|_1.
\end{aligned} \tag{4.6}$$

Combining (4.5), (4.5) and (4.6) 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

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

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 $\sigma_p^2 = 2d\gamma$ [47].

The theoretical analysis for the NOGD algorithm follows the similar procedure as used by the FOGD algorithm. We first introduce a lemma to facilitate 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 [75], 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 9. 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:*

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

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,

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

As proven in [75], 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,

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

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

Remark. As shown in Theorem 5 of [75], 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 [9](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.

4.4 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.

4.4.1 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 [10]. 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:

$$\hat{y}_t = \arg\max_{r \in \mathcal{Y}} f_t^r(\mathbf{x}_t). \tag{4.7}$$

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 [10], 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.

4.4.2 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:

$$\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big) = \max(0, 1 - \gamma_t) = \max(0, 1 - \mathbf{w}_t^{y_t} \cdot \mathbf{z}(\mathbf{x}_t) + \mathbf{w}_t^{s_t} \cdot \mathbf{z}(\mathbf{x}_t)) \quad (4.8)$$

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 η 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:

$$\mathbf{w}_{t+1}^{y_t} = \mathbf{w}_t^{y_t} + \eta \mathbf{z}(\mathbf{x}_t); \quad (4.9)$$

$$\mathbf{w}_{t+1}^{s_t} = \mathbf{w}_t^{s_t} - \eta \mathbf{z}(\mathbf{x}_t). \quad (4.10)$$

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 10.

4.4.3 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

$$f_{t+1}^{y_t} = f_t^{y_t} + \eta \kappa(\mathbf{x}_t, \cdot); \tag{4.11}$$

$$f_{t+1}^{s_t} = f_t^{s_t} - \eta \kappa(\mathbf{x}_t, \cdot). \tag{4.12}$$

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 α_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 α_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 (4.9) and (4.10).

We derive the NOGD algorithm for multi-class version, as summarized in Al-

Algorithm 10 MFOGD — Multi-class Fourier Online Gradient Descent

Input: the number of Fourier components D, step size η, kernel function k;
Initialize $\mathbf{w}_1^r = \mathbf{0}$, $r = 1, ..., m$.
Calculate $p(\mathbf{u})$ for kernel k as (4.2).
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 (4.7);
  Receive y_t and suffer loss $\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big)$ (4.8);
  if $\ell\big(\overline{\mathbf{w}}_t, \mathbf{x}_t, y_t\big) > 0$ **then**
    update $\overline{\mathbf{w}}_t$ as (4.9) and (4.10)
  end if
end for

Algorithm 11 MNOGD — Multi-class Nyström Online Gradient Descent

Input: the budget B, step size η, 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 (4.7);
  Update $\overline{f_t}$ by regular Online Gradient Descent (OGD), as (4.11) and (4.12);
  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 (4.7);
  Update $\overline{\mathbf{w}}_t$ as (4.9) and (4.10)
end for

gorithm 11.

4.4.4 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 10. 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 [54], 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 β_t^r is the parameter used to update $\mathbf{w}_t^r$ as (4.9) and (4.10). Thus, it may be η,

$-\eta$, 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:

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

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:

$$\sum_{t=1}^{T} \ell_t(\overline{\mathbf{w}}_t) - \sum_{t=1}^{T} \ell_t(\overline{\mathbf{w}}_*) \leq \eta T D + \frac{\sum_{r=1}^m \|\mathbf{w}_*^r\|^2}{2\eta}. \tag{4.13}$$

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 [47]), 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:

$$\sum_{r=1}^{m} \|\mathbf{w}_*^r\|^2 \leq m\|f_*\|_1^2(1+\epsilon). \tag{4.14}$$

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

$$|\sum_{t=1}^{T} \ell_t(\overline{\mathbf{w}}_*) - \sum_{t=1}^{T} \ell_t(\overline{f_*})| \leq 2T\epsilon \|f_*\|_1. \tag{4.15}$$

Combining (4.13), (4.14) and (4.15) 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 [31]. For consistency, in the following analysis, we adopt the common slack variables for all classes setting [13, 14].

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

$$\max P(\boldsymbol{\alpha}) = -\frac{1}{2\lambda}\sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top \mathbf{K}\boldsymbol{\alpha}_r - \sum_{r=1}^{m}\boldsymbol{\alpha}_r^\top \mathbf{e}_r,$$

$$s.t.\ \sum_{r=1}^{m}\alpha_{t,r} = 0,\ \forall t \in \{1,2,...,T\};$$

$$\alpha_{t,r} \leq 0, \text{ if } y_t \neq r, \quad \alpha_{t,r} \leq \frac{1}{T}, \text{if } y_t = r, \ \ \forall t \in \{1,2,...,T\}, \ \ \forall r \in \{1,2,...m\}$$

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,

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

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 11. 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:*

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

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.

4.5 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 ϵ as the threshold. Update is only performed when the loss exceeds threshold ϵ. We denote the proposed algorithms the FOGD for Regression (FOGD-R) and NOGD (NOGD-R) for regression tasks, as summarized in Algorithm 12 and Algorithm 13.

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

Algorithm 12 FOGD-R — Fourier Online Gradient Descent for Regression

Input: the number of Fourier components D, step size η, threshold ϵ;
Initialize $\mathbf{w}_1 = \mathbf{0}$.
Calculate $p(\mathbf{u})$ as (4.2). 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 13 NOGD-R — Nyström Online Gradient Descent for Regression

Input: the budget B, step size η, rank approximation k, threshold ϵ.
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 ϵ: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \ell\big(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_t); y_t\big)$.
end for

4.6 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 [28].

4.6.1 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.

Experimental Test bed and Setups

Table 4.1 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.

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 4.1: Details of Binary Classification Datasets in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

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

- "LIBSVM": one of state-of-the-art implementation for batch kernel SVM available at the LIBSVM website [5];
- "LLSVM": Low-rank Linearization SVM algorithm that transfers kernel classification to a linear problem using low-rank decomposition of the kernel matrix [78];
- "BSGD-M": The Budgeted Stochastic Gradient Descent algorithm which extends the Pegasos algorithm [55] by exploring the SV Merging strategy for budget maintenance [69];
- "BSGD-R": The Budgeted Stochastic Gradient Descent algorithm which extends the Pegasos algorithm [55] by exploring the SV Random Removal strategy for budget maintenance [69].

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

To make a fair comparison of algorithms with different parameters, all the parameters, including regularization parameter (C in LIBSVM, λ in pegasos), the learning rate (η in FOGD and NOGD) and the RBF kernel width (σ) 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].

Performance Evaluation Results

Table 4.2 shows the experimental results of batch binary classification tasks. We can drawn several observations from the results.

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 to 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

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

Algori	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**
Algori	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**
Algori	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 4.2: Performance Evaluation Results on Batch Binary Classification Tasks in Experiments of Large Scale Online Kernel Learning by Functional Approximation, Accuracy in Percentage.

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.

4.6.2 Experiments for Online Binary Classification Tasks

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

Experimental Test beds and Setup

Table 4.3 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 include the following two popular algorithms for regular online kernel classification without concerning budget:

- "Percept": the kernelized Perceptron [22] without budget;
- "OGD": the kernelized online gradient descent [33] without budget.

[4] `http://www.ics.uci.edu/~mlearn/`

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 4.3: Details of Online Binary Classification Datasets in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

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 [3];
- "Forgetron": the Forgetron by discarding oldest support vectors [16];
- "Project": the Projectron algorithm using the projection strategy [46];
- "Project++": the aggressive version of Projectron algorithm [45, 46];
- "BPA-S": the Budget Passive-Aggressive algorithm with simple SV removal strategy in [71];
- "BOGD": the Budget Online Gradient Descent algorithm by SV removal strategy [82];

To make fair comparisons, all the algorithms follow the same setups. We adopt the hinge loss as the loss function ℓ. 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 sub-gradient 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. For all gradient descent based algorithms, we search for the best η in range $\{2, 0.2, ..., 0.0002\}$ that achieves the highest accuracy in a random permutation of certain datasets. 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.

Performance Evaluation Results

Table 5.2 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.

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.

Algori	german		spambase		w7a	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	Time(s)
Percept	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
Project	35.6 ± 1.5	0.101	30.8 ± 1.2	0.644	5.38 ± 1.15	11.22
Project++	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

Algori	w8a		a9a		ijcnn1	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	Time(s)
Percept	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
Project	5.42 ± 1.10	38.1	21.6 ± 1.9	15.3	12.38 ± 0.09	19.2
Project++	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

Algori	codrna		KDDCUP08		KDDCUP99	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	Time(s)
Percept	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
Project	15.8 ± 0.5	26.0	0.94 ± 0.02	34.1	0.02 ± 0.00	642
Project++	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 4.4: Evaluation of Binary Classification Task in Experiments of Large Scale Online Kernel Learning by Functional Approximation. Mistake Rate in Percentage

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.

4.6.3 Experiments for Multi-class Classification Tasks

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

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 4.5 summarizes the details of these 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 4.5: Details of Multi-class Classification Datasets in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

We adopt the same set of compared algorithms and similar parameter settings in multiclass 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.

Performance Evaluation Results

Table 5.3 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 4.1 and Figure 4.2 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.

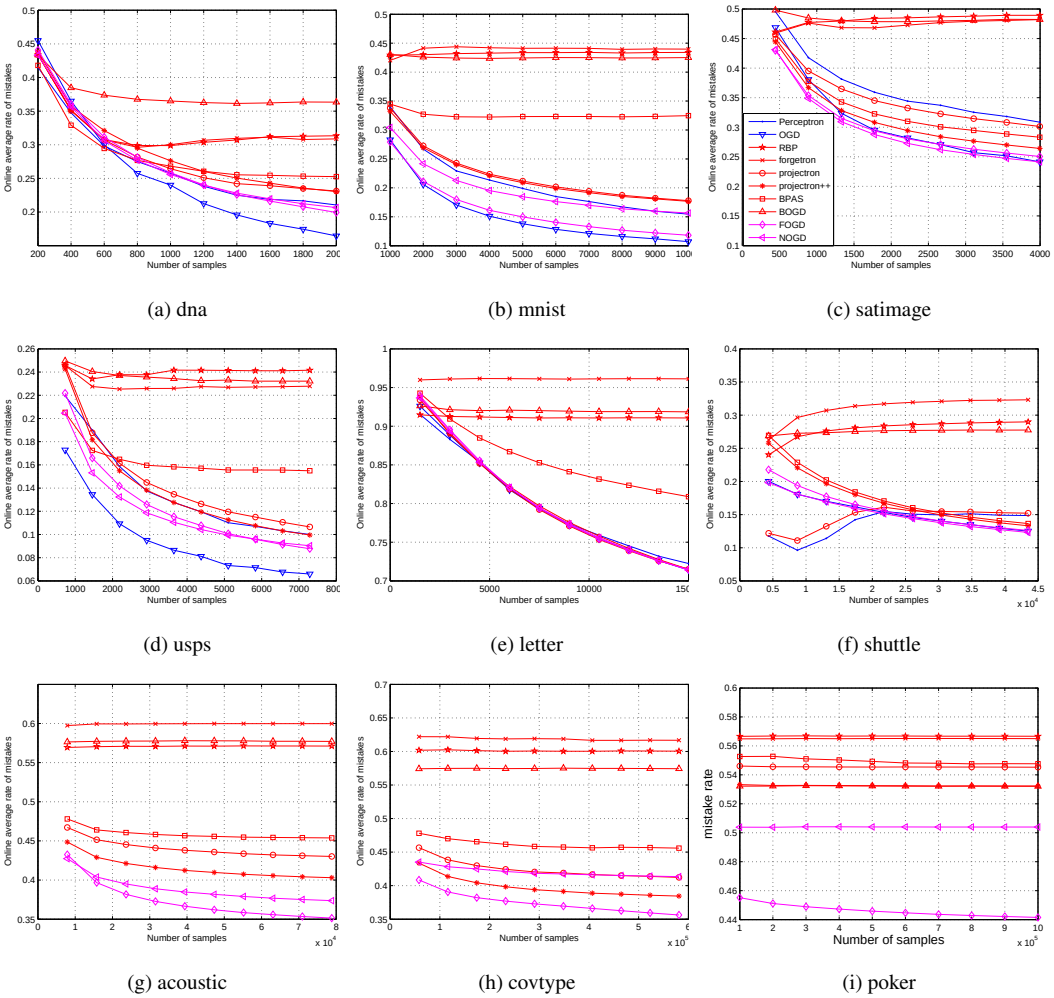
First of all, similar to the binary case, budget online kernel learning algorithms are much more efficient than the regular online kernel learning algorithms without budget, which is more obvious for larger scale datasets. For the three largest dataset-

Algori	dna		mnist		satimage	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	Time(s)
Percept	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
Project	22.7 ±4.4	4.254	17.7 ±0.1	434.53	29.6 ±0.5	3.685
Project++	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

Algori	usps		letter		shuttle	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	Time(s)
Percept	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
Project	10.6 ±0.1	192.347	71.5 ±0.5	26.921	15.3 ±0.3	20.034
Project++	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

Algori	acoustic		covtype		poker	
	Mistake	Time(s)	Mistake	Time(s)	Mistake	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
Project	43.0 ±0.1	46.469	41.1 ±0.2	930.646	54.5 ±0.1	810.421
Project++	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 4.6: Evaluation of Multi-class Classification Task in Experiments of Large Scale Online Kernel Learning by Functional Approximation. Mistake in Percentage



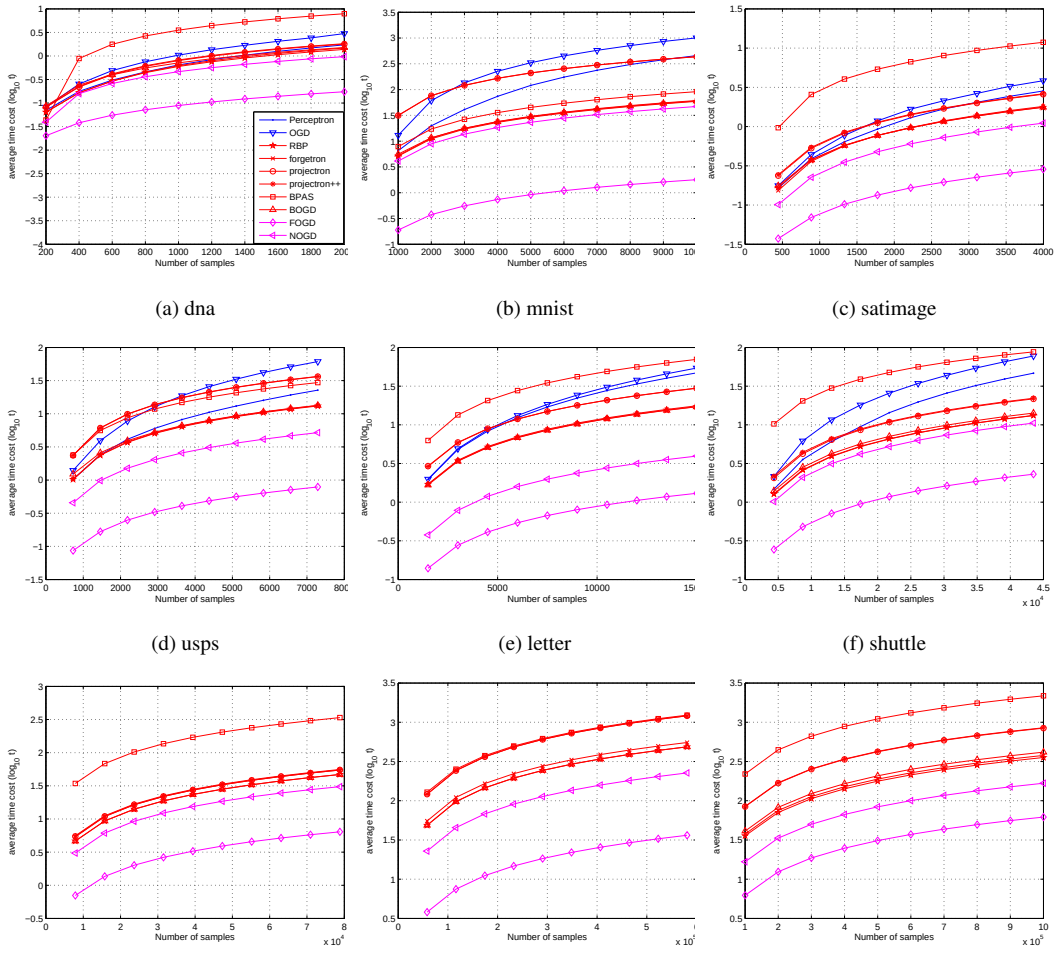
(a) dna (b) mnist (c) satimage

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

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

Figure 4.1: Convergence Evaluation of Multi-class Datasets: Mistake Rate, in Experiments of Large Scale Online Kernel Learning by Functional Approximation

s (*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.



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

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

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

Figure 4.2: Convergence Evaluation of Multi-class Datasets: Time Cost in Experiments of Large Scale Online Kernel Learning by Functional Approximation

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 updat-

ed 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.

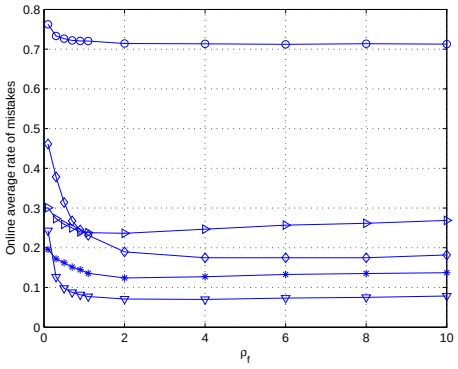
Furthermore, by comparing the two proposed algorithms, FOGD and NOGD, with the existing budget online kernel learning algorithms, we observe that the proposed algorithms achieve the highest accuracy for most cases, and meanwhile run significantly faster than the other algorithms, which again validates 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.

Evaluation for the ρ_f and ρ_n on Multi-class Tasks

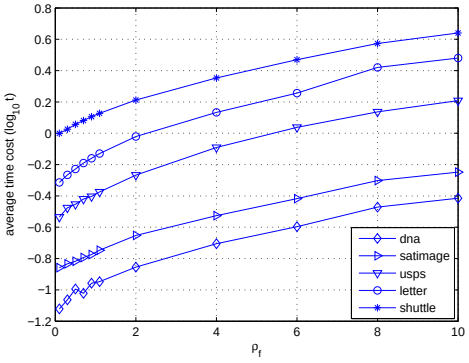
As mentioned in the previous experiments, we set the parameter for the number of Fourier components $D = \rho_f \times B$ and the rank of Nyström matrix approximation $k = \rho_n B$, where different choices of parameters ρ_f and ρ_n could affect the resulting performance of FOGD and NOGD, respectively. 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, ρ_f, and ρ_n) by following the same settings as the previous multi-class classification tasks. Figure 4.3 summarizes the performance evaluation results, including average mistake rates and average time costs. From the results, we can draw some observations as follows.

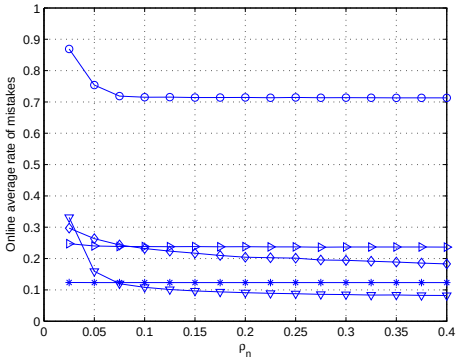
First of all, we observe that increasing the value of ρ_f or ρ_n leads to better clas-



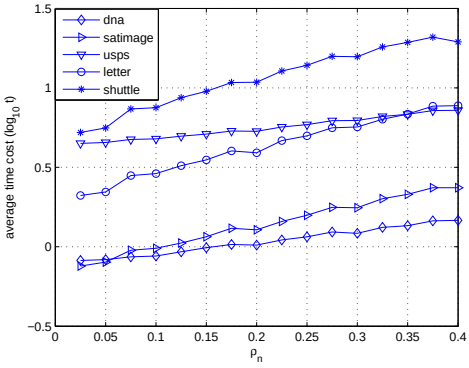
(a) The effect of ρ_f on the mistake rate of FOGD



(b) The effect of ρ_f on the time cost of FOGD



(c) The effect of ρ_n on the mistake rate of NOGD

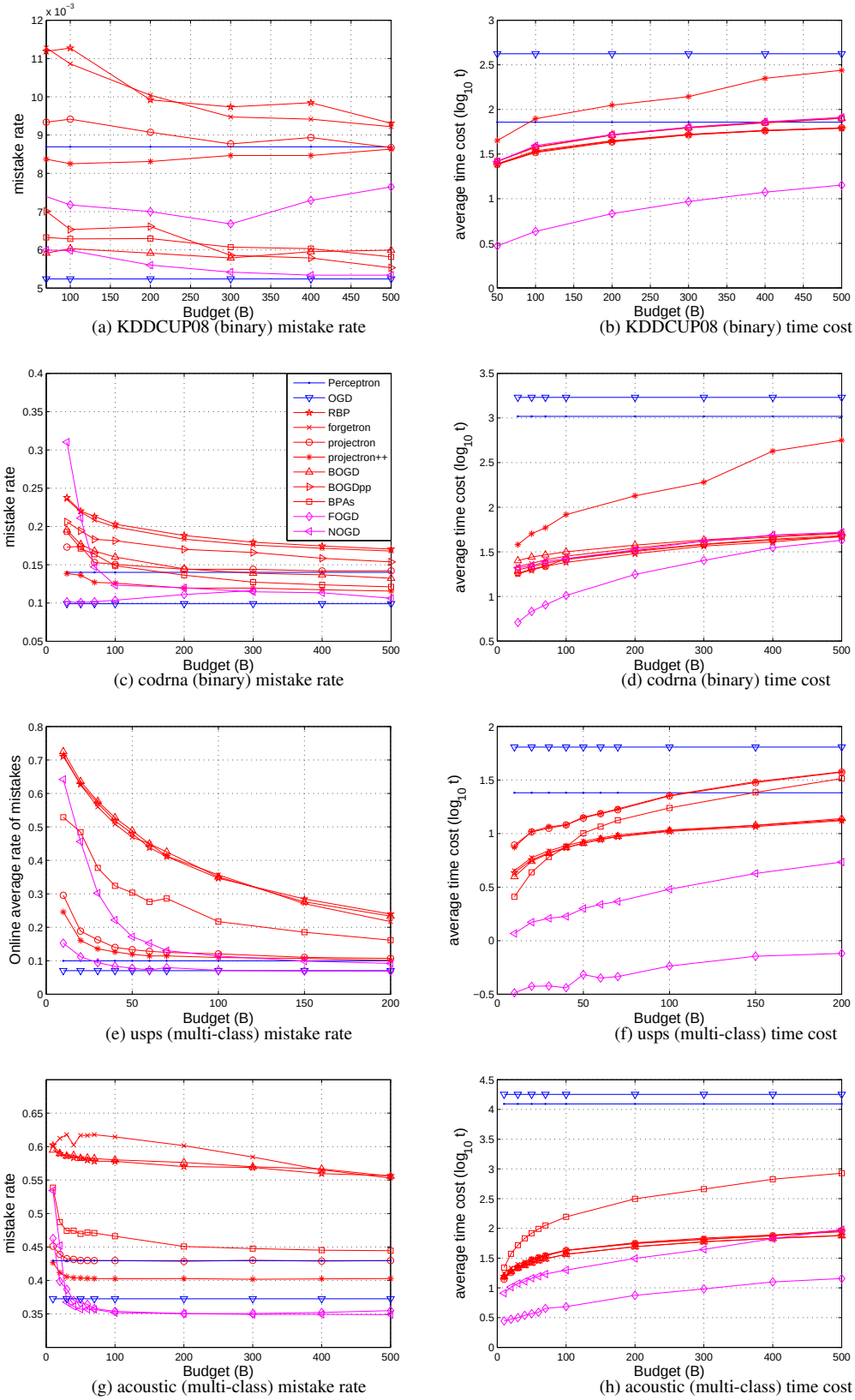


(d) The effect of ρ_n on the time cost of NOGD

Figure 4.3: Performance Evaluation on Different Values of ρ_f and ρ_n in Experiments of Large Scale Online Kernel Learning by Functional Approximation

sification accuracy but higher running time cost. This is not difficult to understand since increasing the value ρ_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 ρ_f. Similarly, for NOGD, the large the value of ρ_n, the more accurate approximation achieved by the Nystrom kernel matrix approximation, and meanwhile the more computational cost required. Thus, the choice of parameter ρ_f or ρ_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



(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 4.4: The Effect of Different Budget Sizes B in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

different results. In particular, we observe that when the value of ρ_f or ρ_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.

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 4.4 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.

4.6.4 Experiments for Online Regression Tasks

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

Experimental Test beds and Setup

Table 4.7 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 [33], and four other existing budget online kernel learning al-

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 4.7: Details of Regression Datasets in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

gorithms 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 ϵ is learned and updated at each iteration. For all the other algorithms, this parameter ϵ 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.

Performance Evaluation Results

Table 4.8 shows the summary of empirical evaluation results on the nine datasets, and Figures 4.6 and 4.6 show the detailed regression 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

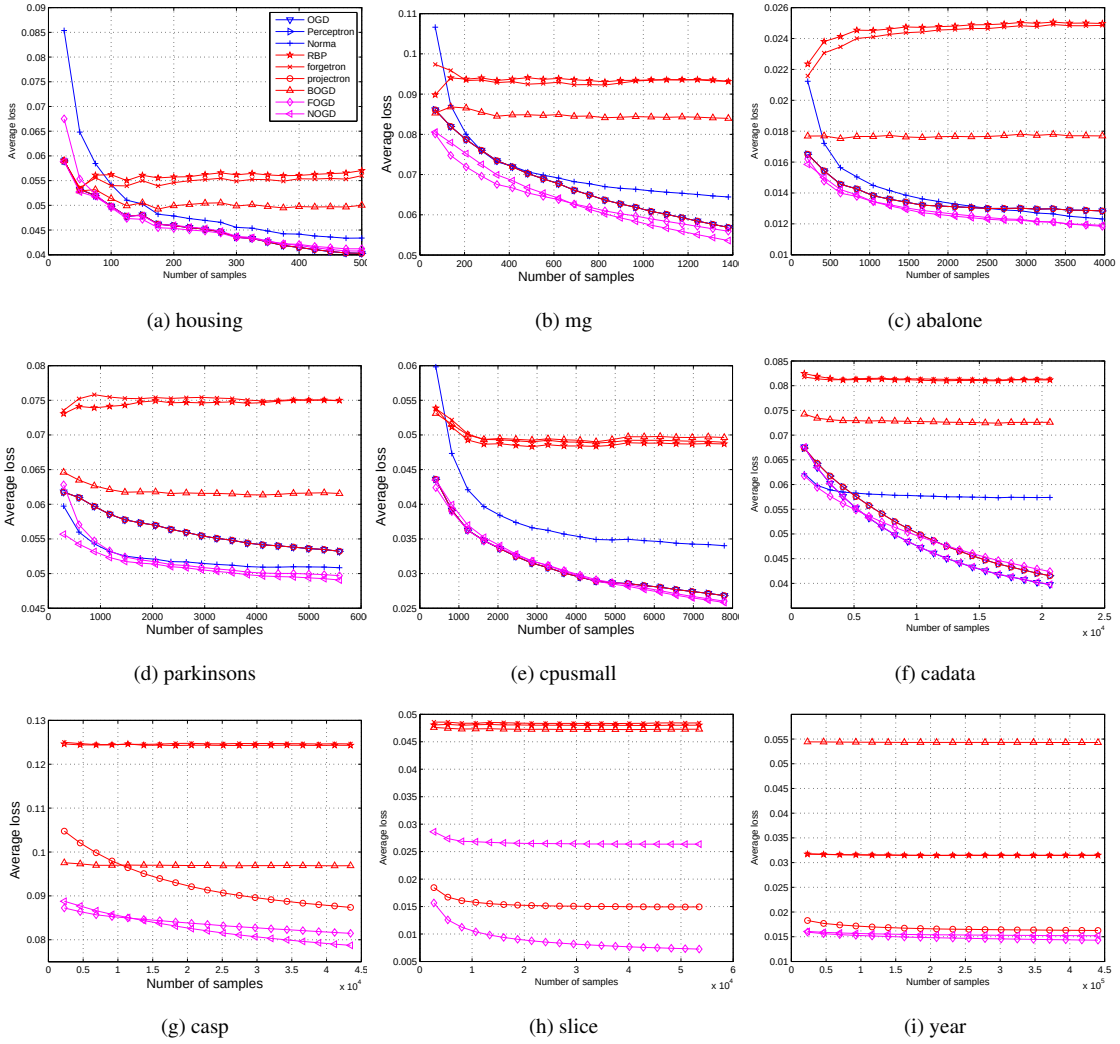
Algori	housing		mg		abalone	
	Squared Loss	Time	Squared Loss	Time	Squared Loss	Time
OGD	0.04017±0.00043	0.028	0.05341±0.00071	0.103	0.01137±0.00007	0.388
Percept	0.04018±0.00080	0.029	0.05682±0.00084	0.103	0.01280±0.00010	0.388
Norma	0.04329±0.00065	0.028	0.06446±0.00073	0.086	0.01224±0.00006	0.448
RBP	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
Project	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

Algori	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
Percept	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
Project	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

Algori	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
Project	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 4.8: Evaluation of Regression Task in Experiments of Large Scale Online Kernel Learning by Functional Approximation,Time in Seconds.

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.



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

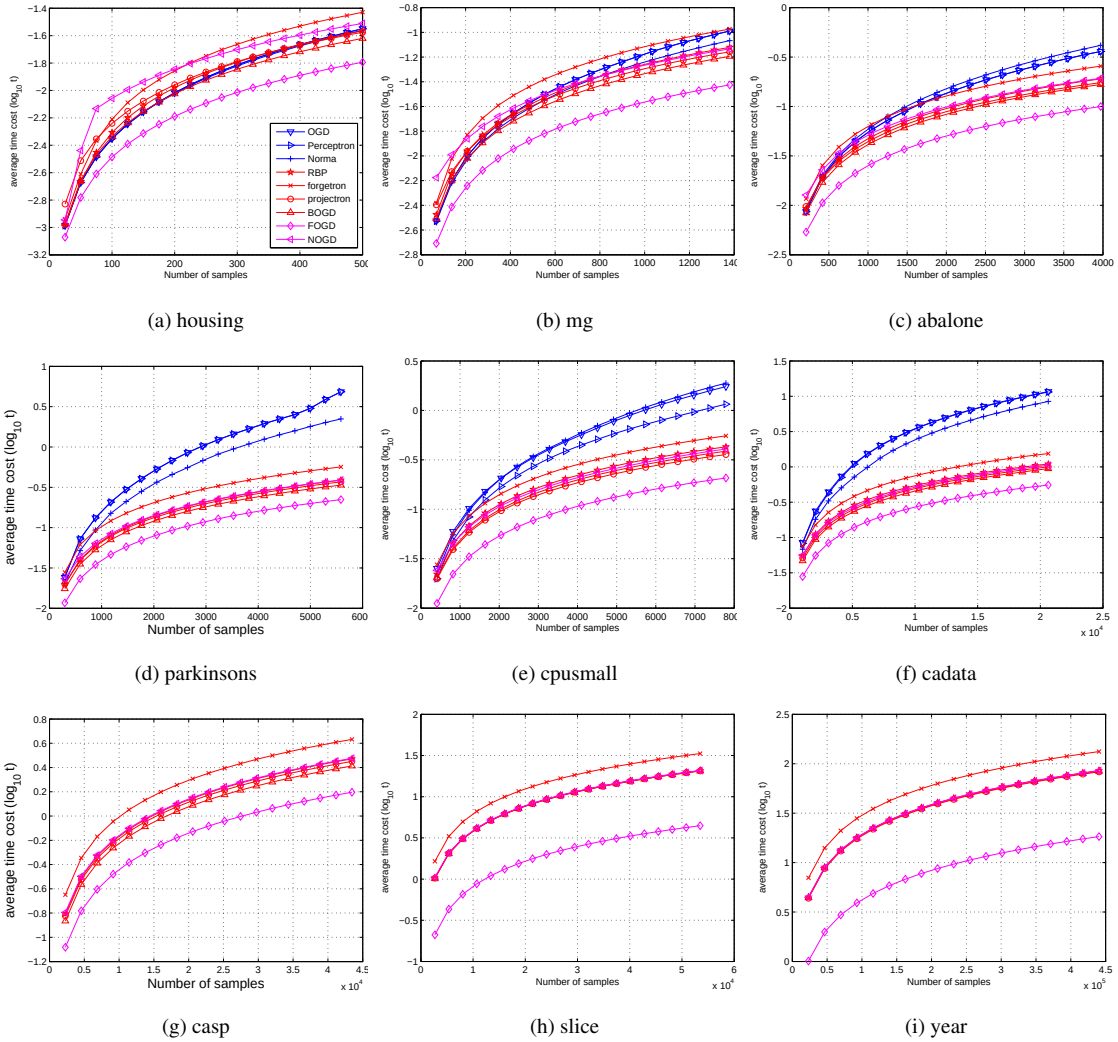
(d) parkinsons (e) cpusmall (f) cadata

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

Figure 4.5: Evaluation of Online Average Squared Loss on the Regression Tasks in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

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 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



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

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

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

Figure 4.6: Evaluation of Online Average Time Cost on Online Regression Tasks in Experiments of Large Scale Online Kernel Learning by Functional Approximation.

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.

4.6.5 Comparison with SPA algorithm

Although the methodologies used in this section is quite different from that used by SPA, all of the three can be used to deal with the online kernel learning problems. It is nature to ask, which of the them is more accurate and which is more efficient. We will answer these questions in the following experiment.

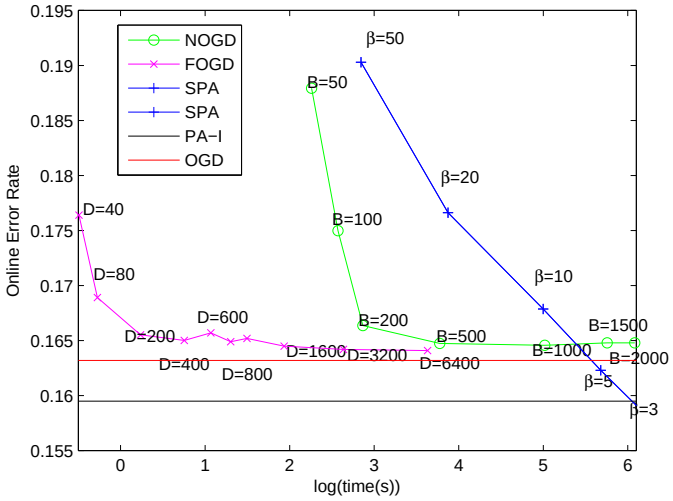


Figure 4.7: Comparison between SPA, FOGD, NOGD with various parameter settings of B, D and β on a9a Dataset.

For fair comparison, the learning rate parameters η of all the three algorithms will be chosen by cross-validation. The Gaussian kernel width parameter is set to $\delta = 8$ for all. The k in NOGD and α are set as the optimal values as stated in the earlier sections.

The only question left is, how to set the FOGD dimension parameter D, the budget size parameters B for NOGD and β SPA, which control the trade off between efficiency and accuracy. As we noticed during our experiments, because of the obvious difference in the methodology, each algorithm has its own optimal trade-off point of parameter settings. First, NOGD is not able to work efficiently when B is large, which is due to the difficult calculation of the SVD for large kernel matrix. Second, the SPA is not accurate when budget size is small (i.e. large β) because of its simple SV selection strategy. Consequently, it is unfair to compare them under the same budget setting. Instead, we find a fair way for comparison. We run the three algorithms with various parameter settings of B, β and D and plot their accuracy on the same time axis, as shown in Figure 4.7.

From the comparison results in the figure, we can draw several observations. First, there is no such an algorithm that can beat all the others in all aspects. Different algorithms have their own optimal trade-off points between efficiency and

effectiveness. Second, when given longer running time (i.e. large B large D, small β), the accuracy of SPA keeps decreasing since it approaches to the non-budget kernel learning algorithm, while the performance of NOGD and FOGD would not improve too much. This indicates that SPA is better in getting high accuracy when time cost is not the main concern. Finally, when running for large scale datasets and the time cost is a major concern, FOGD is preferred, followed by NOGD. This indicates that our proposed functional approximation based algorithms are more suitable for large scale kernel learning compared to SPA.

We also would like to make a discussion about the best accuracy that SPA, FOGD and NOGD could achieve given infinite Fourier components and infinite budget size. First, note that when the D value increases, the mistake rate decreases and approaches to the mistake rate of that in non-budget OGD algorithm while this decrease speed slows down. This is due to the fact that FOGD is an approximation of OGD and the approximation error decreases with the increase of D. This consists with the theoretically analysis in Theorem 1. We infer that if D were set to infinite, the accuracy of FOGD would be identical to that of OGD.

Second, when the B is set to 2000 in NOGD, there is still a gap between the accuracy NOGD and OGD. This due to the error in matrix approximation. As discussed in the remark of Theorem 2, this matrix approximation error is bounded by $\|\mathbf{K} - \mathbf{K}_{0.2B}\|_2 + \frac{T}{\sqrt{B}}\mathbf{K}_{\max}(2 + \log\frac{1}{\epsilon})$ in high probability.

Third, we find that SPA with small β even achieves higher accuracy than that of the non-budget PAI algorithm. This again validates the claim that averaged classifier works better than the last classifier. In addition, in the SPA algorithm if $\ell_t(\mathbf{x})$ is only marginally larger than 0, there will be only a small probability of adding it to the SV set. We infer that this is another reason that SPA extends the performance of PA-I.

In conclusion, the two proposed algorithms in this section, FOGD and NOGD achieve a large improvement from SPA, especially for large datasets in limited time cost.

4.7 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 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, 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, the memory cost of the two algorithm is different. In the FOGD algo-

rithm, we need to store the Fourier components vectors $\mathbf{u}_i \in \mathbb{R}^d, i \in \{1, ..., D\}$, weight vector $\mathbf{w}_t \in \mathbb{R}^{2D}$ and the feature vector $\mathbf{z}(\mathbf{x}_t) \in \mathbb{R}^{2D}$ in memory. Consequently, the total memory cost of FOGD is $D(d+4)$. In the NOGD algorithm, we need to store the B SV's ($\mathbb{R}^{B\times d}$), the mapping matrix $\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}} \in \mathbb{R}^{B\times k}$, the weight vector $\mathbf{w}_t \in \mathbb{R}^k$ and the feature vector $\mathbf{z}(\mathbf{x}_t) \in \mathbb{R}^k$. The total memory of NOGD is $B(d+k)+2k$. As we found in the early experiments (Figure 4.3 etc), for comparable accuracy, the k is usually set to $0.2B$ and the D is $4B$. By comparing the time complexity, we find that for high-dimensional datasets, 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 shift-variant 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.

4.8 Discussion

This section 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.

Chapter 5

Scalable Online Multiple Kernel Learning

Multiple Kernel Learning (MKL) [35, 2] aims to find the optimal (linear) combination of a pool of predefined kernel functions in learning kernel-based predictive models. In comparison to single kernel learning, MKL not only is able to avoid heuristic manual selection of best kernels, but also is able to achieve better performance whenever there are multiple kernels who are complementry for training a better predictive model by combining them, particularly for learning from data with heterogeneous representations. MKL has achieved great successes in many applications, ranging from multimedia [73], signal processing [60], biomedical data fusion [76], mobile app mining [6], and beyond.

Although batch MKL methods have been extensively studied recently [59, 24, 65, 1, 48], unfortunately, the generalization from batch MKL to its online counterpart is far from straightforward. First of all, different from batch MKL that can in principle be solved via cross-validation, online learning with multiple kernels has no foresight on the best kernel function before data arrival but needs to learn kernel classifiers and their combination weights simultaneously from sequential data. Second, online learning with multiple kernels suffers more from the curse of kernelization since more kernel classifiers getting updated would result in even greater

complexity and higher computation cost.

Recent years have witnessed a variety of emerging studies that successfully addressed the first challenge of Online Multiple Kernel Learning (OMKL) and learnt effective multiple kernel classifiers from data stream [32, 40, 43, 44, 26, 53, 73]. Despite the active explorations, it remains an open challenge of making these algorithms scalable for large-scale applications.

In this work, we aim to extend our proposed SPA, FOGD and NOGD algorithms to the OMKL problem. As we demonstrated in the previous chapters, they are effective, efficient and scalable algorithms for online kernel learning in that they solve the problem of unbounded number of SV's. When adopting them to the OMKL setting, the remaining questions are, 1) how to learn the combination weights of different kernels effectively in an online fashion? 2) how to conduct effective kernel selection so that the algorithm is efficient enough when the number of kernels is large.

We address the first problem by adopting the Hedge algorithm. Specially, the weight of each component classifier is reduced by a constant factor whenever this component classifier makes a mistake. For the second question, we adopt a stochastic strategy to select the kernel classifiers according to its historical accuracy. We theoretically prove that the proposed algorithm not only bounds the number of support vectors but also achieves an optimal mistake bound in expectation and conduct an extensive set of empirical studies which show that the proposed algorithm outperforms a variety of budget online kernel learning algorithms.

5.1 Related Work

Our proposed algorithm is closely related to multiple kernel learning, especially online multiple kernel learning. In literature, although MKL has been extensively studied, most of them focus on the batch setting [35, 2] while a few of them studies the OMKL problem.

Existing OMKL algorithms usually adopt one of the two following strategies. The first method is to learn the weights using the Hedge algorithm. Specially, the weight of each kernel is reduced by some factor whenever this single kernel component classifier makes a mistake [32, 26, 53, 73]. Consequently, kernels with higher historical accuracy are assigned with higher weight values. The second method is to add a group sparsity regularizer to the loss function [40, 42, 43, 44]. This then results in a group sparse kernel classifier, where a subset of kernels are assigned with zero weight.

Despite extensive studied, all of these algorithm still suffer from the curse of kernelization due to the lack of effective strategies to bound the number of SV's. And this challenge is even more challenging compared with that in single kernel learning. To the best of our knowledge, no existing work has attempted to scale up the OMKL algorithms with functional approximation methods.

5.2 The Proposed OMKL Algorithms

In this section, we first introduce the problem setting of the online multiple kernel learning algorithm and then propose the three OMKL algorithms. Finally, we theoretically prove the mistake bound of the proposed algorithms.

5.2.1 Problem Setting and Preliminaries

Without loss of generality, we still consider the problem of online learning for binary classification. Following the problem setting introduced in the previous chapters, our goal is to learn a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where instance $\mathbf{x}_t \in \mathbb{R}^d$ and class label $y_t \in \mathcal{Y} = \{+1, -1\}$. We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \rightarrow \mathbb{R}\}$. We will use $\ell(f(\mathbf{x}); y) : \mathbb{R}^2 \rightarrow \mathbb{R}$ as the loss function that penalizes the deviation of estimating $f(\mathbf{x})$ from observed labels y. We denote $\ell_t(f)$ as $\ell(f(\mathbf{x}_t), y_t)$ for

conciseness.

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

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

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

The above convex optimization of regular batch MKL has been resolved using various optimization schemes [24]. Despite the extensive studies in literature, they suffer some common drawbacks of batch learning methods, i.e., poor scalability for large-scale applications, expensive retraining cost for increasing data, and being unable to adapt to fast-changing patterns.

To address the challenges faced by batch MKL methods, Online Multiple Kernel Learning (OMKL) techniques have been proposed to resolve the multiple kernel classification tasks in an online learning manner [26]. In particular, the Online Multiple Kernel Classification (OMKC) method in [26] consists of two major steps. First, it learns a set of single kernel classifiers $f^i_t \in \mathcal{H}_{\kappa_i}, i = 1, ..., m$ using some existing online kernel learning algorithms (such as kernel Perceptron or kernel Passive Aggressive). Second, it learns to find the optimal linear combination of these single kernel classifiers in order to yield an effective final classifier $f_t(\mathbf{x})$:

$$f_t(\mathbf{x}) = \sum_{i=1}^{m} \theta^i_t f^i_t(\mathbf{x}), \tag{5.1}$$

where $\theta^i_t \in [0, 1]$ is the combination weight of the classifier with respect to κ_i at time t. The combination weights can be updated during the online learning process

Algorithm 14 Bounded OMKC Algorithm using SPA

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

by adopting the Hedge algorithm [21].

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

5.2.2 SPA for Online Multiple Kernel Learning

Similar to budget online kernel learning, the goal of bounded online multiple kernel learning is to ensure the total number of support vectors in the final multi-kernel classifier is bounded by a given budget. In order to achieve this, the basic idea is to apply an existing bounded online kernel learning algorithm to bound the support

vector size of each individual single-kernel classifier, which in turn can bound the total SV size given the fixed number of kernels. However, given the number of kernels can be potentially large, the challenge of bounded online multiple kernel learning is not only to bound the number of SV's for each kernel classifier, but also to minimize the overall computational cost and maximize the learning accuracy.

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

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

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

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

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

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

where θ_t^i is the importance weight variable for kernel combination which reflects the historical classification performance of the i-th kernel classifier, and $\delta \in (0, 1)$ is a small constant for smoothing purpose (which ensures every kernel has a certain small probability to be sampled). This sampling strategy

was originally proposed in the SD algorithm of OMKC using the stochastic updating strategy in [26]

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

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

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

The remaining problem is how to learn the appropriate kernel combination weights θ_t^i for the set of single kernel classifiers according to their classification accuracy at each online learning iteration. Specifically, we initialize their weights by a uniform distribution $\theta_1^i = \frac{1}{m}$. Then, the Hedge algorithm [21] is adopted to update the combination weights of the kernel classifiers, during the online learning process, i.e.

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

where $M_t^i = \mathbb{I}(y_t f_t^i(\mathbf{x}_t) < 0)$ indicates if there is a mistake, and $\gamma \in (0, 1)$ is a discount weight parameter. Finally, we summarize the proposed algorithm using the SPA algorithm in Algorithm 14.

In previous chapters, we have demonstrated that the number of mistakes made by a single kernel online classifier is bounded by some constant factor of its batch counterpart in Theorem 1. While in the multiple kernel classification task, the performance of single kernel classifiers varies significantly and we have no foresight to the winner. In the following, we will show that the final multiple kernel classifier in our proposed algorithm achieves almost the same performance as that of

the best online single kernel classifier. We denote $f_*^i = \arg\min_{f \in \mathcal{H}_\kappa^i} \sum_{t=1}^T \ell_t(f)$ as the optimal classifier in $\mathcal{H}_\kappa^i$ space with the assumption of the foresight to all the instances.

Theorem 3. *Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of examples where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y} = \{-1, +1\}$ for all t. If we assume $\kappa_i(\mathbf{x}, \mathbf{x}) = 1$ and $\ell_t(\cdot)$ is the hinge loss function, then for any $\beta \geq \alpha > 0$, $\alpha \leq 1$, and $\eta > 0$, $\gamma \in (0, 1)$, $\delta \in (0, 1)$, the expected number of mistakes made by the multiple kernel classifier generated by our proposed SPA algorithm satisfies the following inequality*

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

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

Proof. Using the analysis in Theorem 2 [21] and the initialization w_1^i with $1/m$, we have,

$$\sum_{t=1}^T \sum_{i=1}^m \theta_t^i M_t^i \leq \frac{\ln m - \ln \gamma \sum_{t=1}^T M_t^i}{1 - \gamma}, \tag{5.2}$$

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

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

we have,

$$\sum_{t=1}^T M_t \leq \frac{2 \ln m - 2 \ln \gamma \sum_{t=1}^T M_t^i}{1 - \gamma}. \tag{5.3}$$

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

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

$$\sum_{t=1}^T \min(\frac{\eta Z_t^i}{\rho_t^i}, 1) M_t^i \leq \sum_{t=1}^T 2 \frac{\eta Z_t^i}{\rho_t^i} \ell_t(f_*^i) + ||f_*^i||_{\mathcal{H}_\kappa^i}^2. \tag{5.4}$$

Here, according to Algorithm (14), $\mathbb{E}[Z_t^i] = \rho_t^i * p_t^i$, where $\delta < p_t^i < 1$. Now, we can take conditional expectation with respect to Z_t^i (given all random variables $Z_1^i, \ldots, Z_{t-1}^i$) to get,

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

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

where the fact $\alpha \leq 1$ is used for the first equality. Plugging the above two equalities into the inequality (5.4), gives

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

Combining the above inequality with inequality (5.3) concludes this proof. □

Remark. This theorem implies that even without any foresight of which kernel would achieve the best accuracy, the performance of our proposed bounded online multiple kernel classifier is still comparable to the best single kernel classifier.

5.2.3 FOGD for Online Multiple Kernel Learning

Here we introduce the detailed steps of our extended FOGD algorithm in OMKL setting. Assuming that all the m kernels are shift-invariant kernels, we calculate their Fourier transform $p_i(\mathbf{u}), i \in \{1, ..., m\}$. For each kernel κ_i, we sample D random Fourier components $\mathbf{u}_{i,j}, j \in 1, ..., D$ according to the probability $p_i(\mathbf{u})$ independently. Following we construct the new feature representation

$$\mathbf{z}_i(\mathbf{x}) = (\sin(\mathbf{u}_{i,1}^\top \mathbf{x}), \cos(\mathbf{u}_{i,1}^\top \mathbf{x}), ..., \sin(\mathbf{u}_{i,D}^\top \mathbf{x}), \cos(\mathbf{u}_{i,D}^\top \mathbf{x}))^\top.$$

whose inner product approximates the corresponding kernel function i.e.

$$\mathbf{z}_i(\mathbf{x}_1)^\top \mathbf{z}_i(\mathbf{x}_2) \approx \kappa_i(\mathbf{x}_1, \mathbf{x}_2), i \in \{1, ..., m\}$$

Algorithm 15 Fourier Online Gradient Descent for Online Multiple Kernel Learning "**OMKL(FOGD)**"

Input: the number of Fourier components D, step size η, kernel function k;
Initialize $\mathbf{w}_1 = \mathbf{0}$.
for $i = 1, ..., m$ **do**
  Calculate $p_i(\mathbf{u})$ for kernel κ_i as (4.2).
  Sample random Fourier components: $\mathbf{u}_{i,1}, ..., \mathbf{u}_{i,D}$ from distribution $p_i(\mathbf{u})$
end for
for $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t$;
  Construct new representation: $\mathbf{z}(\mathbf{x}_t)$ as Equ. (5.5).
  Predict $\hat{y}_t = \text{sign}\Big(\sum_{i=1}^{m} \theta_t^i \cdot \text{sign}[\mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t)]\Big)$;
  Receive the true class label y_t;
  for $i = 1, 2, \ldots, m$ **do**
    Update the i-th classifier using gradient descent
    $\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} - \eta \nabla \ell\big(\mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t); y_t\big)$.
    Update weight $\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i}$, where $M_t^i = \mathbb{I}(y_t \mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t) < 0)$
  end for
  Scale the weights $\theta_{t+1}^i = \frac{\theta_{t+1}^i}{\sum_{j=1}^{m} \theta_{t+1}^j}, i = 1, ..., m$
end for

These m $2D$-dimensional feature vectors are then used as the input to the group sparsity optimization problem. The goal is to train a linear classifier

$$\mathbf{w}_t = (\mathbf{w}_{t,1}, ..., \mathbf{w}_{t,i}, ..., \mathbf{w}_{t,m}) \in \mathbb{R}^{m*2D}$$

from the sequence of instances. During time t, the algorithm receives the feature vector

$$\mathbf{z}(\mathbf{x}_t) = (\mathbf{z}_1(\mathbf{x}_t)^\top, ..., \mathbf{z}_i(\mathbf{x}_t)^\top, ..., \mathbf{z}_m(\mathbf{x}_t)^\top)^\top \in \mathbb{R}^{m \times 2D} \tag{5.5}$$

And we have all the m classifiers $f_t^i(\mathbf{x}_t) = \mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t)$. The Hedge weight update strategy is identical to that introduced in the SPA setting. The classifier update strategy, however, is different from that in the SPA algorithm. In SPA the stochastic classifier update was adopted to reduce the number of support vectors. But in the linear case, we adopt deterministic update since the number of SV's does not grow with the update. We summarize the proposed FOGD for OMKL in Algorithm 15.

We now analysis the mistake bound of the proposed FOGD-OMKL algorithm theoretically. As analyzed in the previous sections,

$$\sum_{t=1}^{T} M_t \leq \frac{2\ln m - 2\ln\gamma \sum_{t=1}^{T} M_t^i}{1-\gamma} \tag{5.6}$$

and for hinge loss, $\sum_{t=1}^{T} M_t^i \leq \sum_{t=1}^{T} \ell_t$, which was bounded by $O(\sqrt{T})$ term. We now conclude that the mistake bound for FOGD for OMKL is comparable to that of the best single kernel classifier, even when we have no foresight for which kernel is the best kernel and it's bounded by a sublinear term.

5.2.4 NOGD for Online Multiple Kernel Learning

We continue to introduce the steps of our extended NOGD algorithm in the OMKL setting. Different from the single kernel setting, we now have m kernel matrixes $\mathbf{K}_i \in \mathbb{R}^{T\times T}, i \in \{1, ..., m\}$. For each of the kernel matrixes, we can do the Nyström kernel approximation and get $\hat{\mathbf{K}}_i$ and obtain the new feature representation $\mathbf{z}_i(\mathbf{x})$ as described in Section 4.3.3. We then learn a linear classifier $\mathbf{w} \in \mathbb{R}^{m\times 2D}$ from the new representation $\mathbf{z}(\mathbf{x}_t) = (\mathbf{z}_1(\mathbf{x}_t)^\top, ..., \mathbf{z}_i(\mathbf{x}_t)^\top, ..., \mathbf{z}_m(\mathbf{x}_t)^\top)^\top \in \mathbb{R}^{m\times 2D}$. We summarizes the proposed NOGD algorithm in Algorithm 16. We get similar mistake bound as previous.

5.3 Experiments

In this section, we conduct extensive experiments to evaluate the empirical performance of the proposed algorithms for online binary classification tasks.

5.3.1 Experiments for SPA OMKL

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

Algorithm 16 OMKL-NOGD

Input: the budget B, step size η, 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 $\hat{y}_t = \text{sign}\Big(\sum_{i=1}^{m} \theta_t^i \cdot \text{sign}[f_t^i(\mathbf{x}_t)]\Big)$;
  for $i = 1, \ldots, m$ **do**
    $f_{t+1}^i = f_t^i - \eta \nabla \ell_t\big(f_t^i\big)$;
    Update $\mathcal{S}_{t+1}^i = \mathcal{S}_t^i \cup \{t\}$ whenever loss is nonzero;
  end for
  $t = t + 1$;
end while
for $i = 1, \ldots, m$ **do**
  Construct the kernel matrix $\hat{\mathbf{K}}_{i,t}$ from $\mathcal{S}_t^i$.
  $[\mathbf{V}_i, \mathbf{D}_i] = \mathsf{eigs}(\hat{\mathbf{K}}_{i,t}, k)$.
  Initialize $\mathbf{w}_{t,i}^\top = [\alpha_{i,1}, ..., \alpha_{i,B}](\mathbf{D}_i^{-0.5}\mathbf{V}_i^\top)^{-1}$.
end for
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 $\mathbf{x}_t$:
  $\mathbf{z}_i(\mathbf{x}_t) = \mathbf{D}_i^{-0.5}\mathbf{V}_i^\top(\kappa_i(\mathbf{x}_t, \hat{\mathbf{x}}_1), ..., \kappa_i(\mathbf{x}_t, \hat{\mathbf{x}}_B))^\top$,
  Predict $\hat{y}_t = \text{sign}\Big(\sum_{i=1}^{m} \theta_t^i \cdot \text{sign}[\mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t)]\Big)$;
  for $i = 1, 2, \ldots, m$ **do**
    Update the i-th classifier using gradient descent
    $\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} - \eta \nabla \ell\big(\mathbf{w}_{t,i}^\top \mathbf{z}_i(\mathbf{x}_t); y_t\big)$.
    Update weight $\theta_{t+1}^i = \theta_t^i \gamma^{M_t^i}$, where $M_t^i = \mathbb{I}(y_t \mathbf{w}_{t,i} \mathbf{z}_i(\mathbf{x}_t) < 0)$
  end for
  Scale the weights $\theta_{t+1}^i = \frac{\theta_{t+1}^i}{\sum_{j=1}^m \theta_{t+1}^j}, i = 1, ..., m$
end for

Experimental Testbed

All datasets used in our experiments are commonly used benchmark datasets and are publicly available from LIBSVM, UCI [1] and KDDCUP competition site. These datasets are chosen fairly randomly to cover a variety of different dataset scales. We summarize the details of the datasets in Table 5.1.

Table 5.1: Summary of binary classification datasets in the OMKC experiments.

Data sets	# instances	#features
German	1000	24
Svmguide3	1,243	21
Madelon	2,000	500
Magic04	19,020	10
A9a	48,842	123
Ijcnn1	49,990	22
Kdd08	102,294	117
Codrna	271,617	8
Susy	1,000,000	18

Kernels

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

Compared Algorithms

First of all, we include an "ideal" baseline algorithm which assumes the best kernel among the pool of kernels can be disclosed prior to the arrival of training data at the beginning of online learning. Specifically, we search for the best single kernel classifier from the set of 16 predefined kernels using one random permutation of all the training examples. Using this best kernel, we then construct an "ideal" kernel classifier using the Perceptron algorithm [50], denoted as "Per(*)" in later discussion.

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

The second group of compared algorithms are the existing OMKC algorithms in [26], including three variants of OMKC:

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

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

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

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

- "RBP": the Random Budget Perceptron algorithm [3];
- "Forgetron": budget Perceptron by discarding the oldest SV [17];
- "BOGD": the Budget OGD algorithm [82];
- "BPAS": the Budget PA algorithm [71].

Due to the highly intensive computational costs, we exclude the comparisons with other insufficient budget online kernel learning algorithms such as Projectron and its variants. Although the procedure of both DD and SD algorithms was clearly

Algorithm 17 The Bounded OMKC-DD using the Deterministic update

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

presented in [26], we still describe its budget variants used in our comparison in Algorithm 17 and 18 for easier understanding, where B is the total budget size of all component classifiers and the "budget maintenance" step denotes any of the four budget algorithms.

Parameter Settings

To make a fair comparison, we adopt the same experimental setup for all the algorithms. The weight discount parameter γ is fixed to 0.99 for all multiple kernel algorithms on all datasets. The smoothing parameter δ for all stochastic update algorithms is fixed to 0.001. The learning rate parameters in PA-based algorithms (SPA, BPAS, DDPA, SDPA) are all fixed to 0.1. The regularization parameter λ and learning rate parameter η in BOGD is searched in the range of $\{1, 0.1, ..., 0.0001\}$. For the proposed SPA algorithm, we set $\alpha = 1$ for all the datasets. In addition, as discussed in the theoretical analysis, the number of SV's is bounded by $\alpha T/\beta$, which indicates that β should vary according to the number of instances T in a dataset. We thus set $\beta = 3$ for smaller datasets ($T < 10^5$) and $\beta = 10$ for other datasets. For fair comparison, we choose the budget size B for all the other compared budget algorithms the same as the total number of SV's yielded by our proposed SPA

Algorithm 18 The Bounded OMKC-SD using the Stochastic update

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

algorithms.

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

Evaluation by Comparing SPA with Unbounded OMKC

The first experiment is to evaluate the performance of SPA for binary classification tasks by comparing it with the existing unbounded OMKC algorithms. Note that we did not report the results on three large-scale datasets in this experiment since it is simply computationally prohibited to run the non-budget algorithms on such large datasets due time and memory limits. Table 5.2 summarizes the results. We can draw some observations below.

First of all, we compare the three unbounded OMKC algorithms using deterministic updates (Perceptron(*), OMKC(U), OMKC(DD)). Obviously, the mistake

Table 5.2: Evaluation of Online Classification on Small-scale and Medium-scale Datasets by comparing SPA with Unbounded OMKC algorithms (time in seconds).

Algorithm	german			madelon		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
Per(*)	31.87$\pm$1.61	318.8$\pm$16.1	0.04	48.65$\pm$1.57	973.0$\pm$31.4	0.81
U	35.75$\pm$1.52	6904.1$\pm$101.3	0.38	50.00$\pm$0.00	26498.2$\pm$92.6	35.27
DD	30.71$\pm$1.25	6904.1$\pm$101.3	0.38	41.15$\pm$0.50	26498.2$\pm$92.6	35.46
SD	34.83$\pm$1.54	5723.3$\pm$95.1	0.34	50.00$\pm$0.00	13872.3$\pm$150.6	20.67
DDPA	28.92$\pm$0.90	12481.5$\pm$60.9	0.70	**37.55$\pm$1.00**	28918.0$\pm$95.7	39.39
SDPA	28.88$\pm$0.63	6794.6$\pm$93.5	0.45	50.00$\pm$0.00	23837.2$\pm$29.7	33.93
SPA	**28.57$\pm$0.51**	2388.2$\pm$95.2	**0.16**	39.39$\pm$1.12	2285.2$\pm$68.8	**3.58**
Algorithm	a9a			magic04		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
Per(*)	20.43$\pm$0.13	9980.1$\pm$67.4	22.5	24.29$\pm$0.13	4620.9$\pm$25.9	1.94
U	20.60$\pm$0.10	234008.7$\pm$373.8	1178.5	28.70$\pm$0.12	157922.7$\pm$164.4	199.5
DD	19.20$\pm$0.12	234008.7$\pm$373.8	1178.6	22.58$\pm$0.46	157922.7$\pm$164.4	199.0
SD	18.93$\pm$0.10	155393.9$\pm$560.9	778.9	22.39$\pm$0.11	73433.4$\pm$320.1	69.4
DDPA	**15.82$\pm$0.07**	491137.2$\pm$237.2	3061.3	**18.91$\pm$0.10**	243950.0$\pm$106.9	352.8
SDPA	20.70$\pm$0.48	214751.2$\pm$474.8	1051.6	34.57$\pm$0.01	92635.9$\pm$560.1	94.6
SPA	16.31$\pm$0.11	14540$\pm$1779.4	**58.0**	19.62$\pm$0.23	7452.3$\pm$869.2	**5.10**
Algorithm	KDD08			svmguide3		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
Per(*)	0.94$\pm$0.01	963.5$\pm$11.0	14.7	25.98$\pm$0.51	323.0$\pm$6.3	0.03
U	0.66$\pm$0.01	32665.1$\pm$199.7	829.7	28.25$\pm$0.80	6166.9$\pm$100.1	0.36
DD	0.72$\pm$0.01	32665.1$\pm$199.7	829.0	25.41$\pm$0.95	6166.9$\pm$100.1	0.37
SD	0.63$\pm$0.01	17218.5$\pm$95.3	450.8	24.65$\pm$0.46	5550.8$\pm$95.8	0.36
DDPA	0.60$\pm$0.01	473444.2$\pm$124.6	12371	21.81$\pm$0.21	13734.9$\pm$61.6	0.87
SDPA	**0.59$\pm$0.01**	409441.0$\pm$7293.0	10834	**21.64$\pm$0.11**	12596.2$\pm$217.9	0.80
SPA	**0.59$\pm$0.01**	13749.1$\pm$858.2	**331.5**	22.21$\pm$0.24	2777.1$\pm$92.50	**0.20**

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

fier. This further validates the significance of exploring multiple kernel learning.

Second, we found that despite generating fewer SV's, the OMKC(SD) algorithms using stochastic updates achieve even lower mistake rates compared with OMKC(DD) using deterministic updates, which is consistent with the previous observations in [26]. This indicates that not all SV's are essentially useful for constructing an accurate final classifier. This supports our main claim that when the added SV's are wisely selected, the accuracy may not be degraded much while the efficiency can be significantly improved.

Finally, we found that most PA based algorithms significantly outperform the Perceptron based ones in terms of accuracy, but paid by requiring much higher time and space costs due to more aggressive updates. We then make a comparison among the three PA based algorithms. Although only a small fraction of SV's are adopted, the proposed SPA algorithm still achieves comparable or sometimes even better accuracy compared with its unbounded counterparts. Moreover, the time cost reduction by the proposed SV sampling strategy is especially more significant on the large datasets (e.g., KDD08 and a9a). In fact, when the data sets are very large (such as "Codrna" and "Susy", as shown in later experiments), it is even impossible for non-budget algorithms to complete the whole online learning process due extremely huge time cost and memory space needed for storing the unbounded SV's which grows continuously in the online learning process for large-scale datasets.

Evaluation of Different Bounded OMKC Algorithms

The last experiment is to test the accuracy and efficiency of our proposed SPA algorithm in comparison to the other variants of OMKC algorithms using traditional budget maintenance strategies. Table 5.3 summarizes the experimental results on several large-scale datasets.

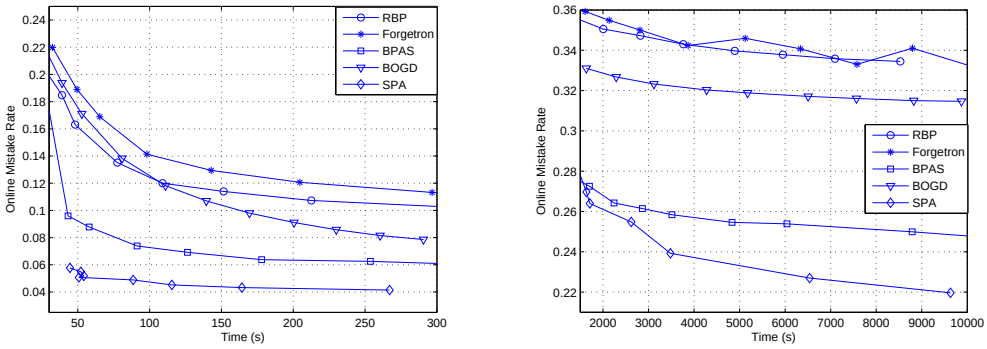
First, we compare the accuracy of the two groups of traditional budget algorithms, bounded OMKC(DD) using the Deterministic updating strategy (denoted as "DD" for short) and bounded OMKC(SD) using Stochastic updating strategy

Table 5.3: Evaluation of OMKC using different budget learning algorithms (time in sec.).

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

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

Second, it is clear that SPA not only achieves the best accuracy among all bounded OMKC algorithms, but also the lowest time cost. There are two major reasons for explaining the promising results: (i) Different from the budget DD that treats each kernel equally, SPA concentrates more effort on updating the classifiers of



(a) codrna (b) susy

Figure 5.1: Evaluation of different Bounded OMKC algorithms on two large-scale data sets. The curves of online mistake rates vs time costs were obtained by choosing varied budget values. As bounded SD algorithms have no significant advantages over Bounded DD (see Table 5.3), we thus only include Bounded DD algorithms in these 2 figures to simplify the presentation.

good kernels and thus the poor kernels will yield fewer SV's, which can greatly reduce the prediction time cost; (ii) Different from the budget SD where the classifier weights θ are also updated in a stochastic manner, the weights in the proposed SPA algorithm are updated always whenever a classification mistake appears, which leads to better SV allocation and more precise prediction combination.

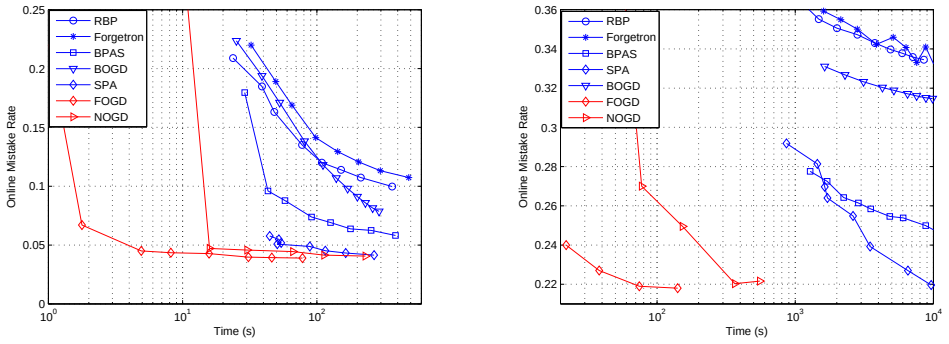
To further examine the trade-off between classification accuracy and computational efficiency, Figure 5.2 shows the evaluation of different Bounded OMKC algorithms on two large-scale data sets, where the curves of online mistake rates vs time costs were obtained by choosing varied budget values (via proper parameter settings). As observed from the results, for the largest dataset "SUSY" with one million instances, we found that when following the previous setting where all the algorithms adopt the same number of SV's, the compared budget algorithms cannot finish processing the dataset in the fixed amount of time. Thus, for a fair comparison, we plot the online mistake rate of different algorithms on the same time axis according to different SV sparsity setting ($\beta = 2, ..., 10$). Obviously, the proposed SPA algorithm always achieves the lowest mistake rate with the given same amount of time cost, which validates the effectiveness and robustness of SPA.

5.3.2 Experiments of Functional Approximation Algorithms for OMKL

In this subsection, we evaluate the performance of the two proposed functional approximation algorithms in terms of accuracy and time cost.

Experiment Setup

The experiment setup is mostly the same as that for the SPA section. We test the FOGD and NOGD algorithm using C++. The η, ρ_n for NOGD and FOGD are set according to the setting of Chapter 4. Since FOGD is only suitable for RBF kernels, in the FOGD we only test on the 13 gaussian kernels.



(a) codrna (b) susy

Figure 5.2: Evaluation of different OMKL algorithms on two large-scale data sets. The curves of online mistake rates vs time costs were obtained by choosing varied budget values. We only include Bounded DD algorithms in these 2 figures to simplify the presentation.

Comparison Under Fixed Budget Size

The first experiment is to evaluate the performance of the two functional approximation algorithms under fixed budget size. We set $B = D = 100$ for all single kernel component classifiers in all algorithms. So in total, each budget algorithm have 1600 SV's and the FOGD has 1300 random fourier vectors for the 13 gaussian kernels. The experiment results are reported in Table 5.4 and 5.5. Note that for the largest scale datasets, we did not report the results of the non-budget algorithms because of the time and memory limits. We can draw some observations as below.

Table 5.4: Evaluation of Online Classification on Small-scale and Medium-scale Datasets (time in seconds). We report the number of SV's for all budget algorithms and the number of Fourier components for FOGD algorithms.

Algorithm	german			w7a		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
Perceptron*	$31.87_{\pm 1.61}$	$318.8_{\pm 16.1}$	0.04	$2.73_{\pm 0.03}$	$674.5_{\pm 8.8}$	0.83
U	$35.75_{\pm 1.52}$	$6904.1_{\pm 101.3}$	0.38	$2.46_{\pm 0.03}$	$67111.3_{\pm 140.1}$	138.31
DD	$30.71_{\pm 1.25}$	$6904.1_{\pm 101.3}$	0.38	$2.28_{\pm 0.06}$	$67111.3_{\pm 140.1}$	138.31
SD	$34.83_{\pm 1.54}$	$5723.3_{\pm 95.1}$	0.34	$2.18_{\pm 0.04}$	$13285.6_{\pm 98.8}$	24.02
RBP-DD	$31.53_{\pm 0.71}$	1600	0.15	$2.99_{\pm 0.11}$	1600	3.93
Forget-DD	$33.00_{\pm 1.13}$	1600	0.21	$3.39_{\pm 0.15}$	1600	5.24
BOGD-DD	$30.00_{\pm 0.49}$	1600	0.18	$3.00_{\pm 0.00}$	1600	4.91
BPAS-DD	$30.08_{\pm 0.40}$	1600	0.18	$3.00_{\pm 0.00}$	1600	4.93
RBP-SD	$32.02_{\pm 1.01}$	1600	0.15	$3.08_{\pm 0.15}$	1600	4.31
Forget-SD	$33.48_{\pm 1.79}$	1600	0.19	$3.39_{\pm 0.10}$	1600	4.66
BOGD-SD	$29.34_{\pm 0.69}$	1600	0.17	$3.00_{\pm 0.00}$	1600	5.04
BPAS-SD	$30.07_{\pm 0.56}$	1600	0.18	$3.00_{\pm 0.00}$	1600	4.70
FOGD	$30.22_{\pm 0.11}$	1300	0.08	$3.00_{\pm 0.00}$	1300	1.52
NOGD	$30.24_{\pm 0.32}$	1600	0.24	$3.00_{\pm 0.00}$	1600	4.89
Algorithm	a9a			KDD08		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
Perceptron*	$20.43_{\pm 0.13}$	$9980.1_{\pm 67.4}$	22.5	$0.94_{\pm 0.01}$	$963.5_{\pm 11.0}$	14.7
U	$20.60_{\pm 0.10}$	$234008.7_{\pm 373.8}$	1178.5	$0.66_{\pm 0.01}$	$32665.1_{\pm 199.7}$	829.7
DD	$19.20_{\pm 0.12}$	$234008.7_{\pm 373.8}$	1178.6	$0.72_{\pm 0.01}$	$32665.1_{\pm 199.7}$	829.0
SD	$18.93_{\pm 0.10}$	$155393.9_{\pm 560.9}$	778.9	$0.63_{\pm 0.01}$	$17218.5_{\pm 95.3}$	450.8
RBP-DD	$22.39_{\pm 0.57}$	1600	8.80	$0.68_{\pm 0.00}$	1600	50.10
Forget-DD	$22.48_{\pm 0.37}$	1600	13.19	$0.86_{\pm 0.00}$	1600	50.60
BOGD-DD	$20.46_{\pm 0.07}$	1600	9.37	$0.61_{\pm 0.00}$	1600	51.51
BPAS-DD	$18.73_{\pm 0.30}$	1600	10.14	$0.61_{\pm 0.00}$	1600	51.60
RBP-SD	$20.62_{\pm 0.39}$	1600	9.28	$0.70_{\pm 0.00}$	1600	50.10
Forget-SD	$21.64_{\pm 0.19}$	1600	11.63	$0.77_{\pm 0.00}$	1600	50.07
BOGD-SD	$21.52_{\pm 0.076}$	1600	9.54	$0.61_{\pm 0.00}$	1600	54.20
BPAS-SD	$18.72_{\pm 0.21}$	1600	9.96	$0.61_{\pm 0.00}$	1600	54.48
FOGD	$16.52_{\pm 0.17}$	1300	3.35	$0.61_{\pm 0.00}$	1300	20.27
NOGD	$17.24_{\pm 0.26}$	1600	9.66	$0.61_{\pm 0.00}$	1600	51.57

Table 5.5: Evaluation of Online Classification on large-scale Datasets time in seconds).We report the number of SV's for all budget algorithms and the number of Fourier components for FOGD algorithms.

Algorithm	codrna			ijcnn1		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
RBP-DD	$20.71_{\pm 0.61}$	1600	23.41	$12.54_{\pm 0.33}$	1600	5.22
Forgetron-DD	$21.95_{\pm 0.08}$	1600	30.14	$13.26_{\pm 0.39}$	1600	6.26
BOGD-DD	$24.29_{\pm 0.05}$	1600	28.48	$9.71_{\pm 0.00}$	1600	6.33
BPAS-DD	$15.45_{\pm 1.09}$	1600	28.86	$9.71_{\pm 0.00}$	1600	6.15
RBP-SD	$17.48_{\pm 0.51}$	1600	22.70	$9.47_{\pm 0.18}$	1600	5.25
Forgetron-SD	$19.45_{\pm 0.37}$	1600	27.59	$10.28_{\pm 0.12}$	1600	6.10
BOGD-SD	$32.75_{\pm 0.03}$	1600	27.36	$9.71_{\pm 0.00}$	1600	6.33
BPAS-SD	$21.88_{\pm 0.62}$	1600	25.83	$9.71_{\pm 0.00}$	1600	5.98
FOGD	$4.33_{\pm 0.15}$	1300	15.85	$6.36_{\pm 0.27}$	1300	3.21
NOGD	$4.57_{\pm 0.04}$	1600	30.27	$9.35_{\pm 0.45}$	1600	6.89
Algorithm	susy			covtype		
	Error(%)	#SV's	Time	Error(%)	#SV's	Time
RBP-DD	$42.10_{\pm 0.31}$	1600	141.59	$39.02_{\pm 0.32}$	1600	68.21
Forgetron-DD	$42.29_{\pm 0.03}$	1600	198.58	$39.48_{\pm 0.05}$	1600	95.40
BOGD-DD	$42.18_{\pm 0.01}$	1600	151.71	$37.61_{\pm 0.04}$	1600	77.86
BPAS-DD	$35.90_{\pm 0.43}$	1600	154.08	$30.51_{\pm 0.19}$	1600	79.56
RBP-SD	$42.47_{\pm 0.71}$	1600	134.10	$36.43_{\pm 0.39}$	1600	67.79
Forgetron-SD	$43.30_{\pm 0.53}$	1600	180.55	$38.13_{\pm 0.23}$	1600	90.27
BOGD-SD	$44.01_{\pm 0.04}$	1600	149.05	$39.31_{\pm 0.06}$	1600	75.39
BPAS-SD	$37.21_{\pm 0.35}$	1600	142.89	$30.68_{\pm 0.21}$	1600	76.31
FOGD	$21.95_{\pm 0.14}$	1300	74.37	$23.95_{\pm 0.31}$	1300	37.28
NOGD	$24.95_{\pm 0.49}$	1600	152.99	$32.14_{\pm 0.46}$	1600	77.05

First of all, we compare the the 4 existing non-budget OMKL algorithms and draw similar observations as that in the previous section. The accuracy of "DD" significantly outperforms that of uniform combination ("U") and the best single kernel classifier, which validates the significance of exploring multiple kernel learning.

Second, we find that the proposed NOGD and FOGD usually achieve higher accuracy than most of the budget learning algorithms, which is more obvious in larger datasets. This validates our claim that the two functional approximation algorithms are more powerful than traditional budget maintenance strategies. This conclusion consists with that in the single kernel experiments. Specially, the FOGD algorithm is extremely fast.

Comparison Under Variant Budget Size

In the first experiment we compare the accuracy and time cost of different OMKL algorithms under fixed budget size. This is sometimes not comprehensive since slower algorithms might achieve higher accuracy. Note that the B and D parameter control the trade-off between efficiency and effectiveness. It is thus more straight forward to compare the algorithms under variant B and D values. We plot their accuracy on the same time axis, in Figure 5.2.

We can draw several observations. First, the three proposed algorithms all outperform the existing budget algorithms in most of the cases. Second, FOGD is able to achieve higher accuracy when the time cost is very limited, which is better than NOGD and SPA. But when we allow higher time cost (larger B and D or smaller β), SPA can achieve higher accuracy compared to these two since it approaches to the original non-budget PA algorithms.

5.4 Single Kernel vs Multiple Kernel

In the previous sections, we have demonstrated that the proposed online multiple kernel learning algorithms, SPA, FOGD and NOGD outperformed the other budget

Table 5.6: Evaluation of Online Classification on large-scale Datasets (time in seconds). All algorithms adopt the same number of SVs. B=1600, decreasing factor $\gamma = 0.999$

Algorithm	codrna		a9a		susy	
	Error(%)	Time	Error(%)	Time	Error(%)	Time
RBP-single	$7.58_{\pm 0.07}$	14.63	$21.03_{\pm 0.05}$	6.05	$34.26_{\pm 0.24}$	82.16
BOGD-single	$8.19_{\pm 0.06}$	16.66	$17.01_{\pm 0.17}$	6.88	$30.35_{\pm 0.35}$	90.51
SPA-multiple	$10.92_{\pm 1.03}$	13.01	$18.71_{\pm 1.35}$	5.72	$35.53_{\pm 1.08}$	68.90
FOGD-multiple	$4.33_{\pm 0.15}$	15.85	$16.52_{\pm 0.17}$	3.35	$21.95_{\pm 0.14}$	74.37
NOGD-multiple	$4.57_{\pm 0.04}$	30.27	$17.24_{\pm 0.26}$	9.66	$24.95_{\pm 0.49}$	152.99

OMKL algorithms under same budget size (Table 5.3, 5.4, 5.5) and under the same time cost (Figure 5.1, 5.2). While there is still an open question: given same number of SV's, is their performance better than a single kernel classifier?

Obviously, the performance online multiple kernel classifiers should be better than lots of single kernel classifiers since OMKC is designed to filter out bad kernels. Consequently, in this section, we only compare our proposed OMKC algorithms with the best single kernel classifier, assuming that the single kernel classifier had foresight for the best kernel. The results are demonstrated in Figure 5.6.

We find that FOGD and NOGD can even outperform the single kernel classifier although the single kernel classifier is with the unrealistic assumption that it knows the best classifier. The SPA algorithm is not accurate when using so few SV's, this consists with that in Figure 4.7. But note that this weaker performance when compared with single kernel classifiers does not indicate the ineffective of the proposed algorithms, since OMKL algorithms is designed to avoid manual kernel selection.

5.5 Discussion

To make large-scale kernel methods practical, we proposed a novel framework for bounded online multiple kernel learning. We extended the proposed SPA, FOGD and NOGD algorithms to Bounded Online Multiple Kernel Learning tasks. Our promising experimental results shows that our algorithms outperform a variety of state-of-the-art budget online learning algorithms.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Kernel learning plays an important role in the machine learning field due to its ability to learn complicated nonlinear patterns. Online Kernel Learning, which learns a kernel based model from sequentially arriving data, is a powerful tool in many real-world, real-time applications, such as classification, regression and ranking. In this paper, we address the key challenge in Online Kernel Learning, the curse of kernelization and make OKL scalable to large scale applications.

In our survey of online kernel learning algorithms, we find that there are still some open challenges. First, most of the existing budget online kernel learning algorithms can not bound the number of SV's for the averaged classifier over the learning process, which makes them not suitable for online-to-batch conversion. Second, existing budget maintenance strategies are usually either too simple to achieve satisfactory accuracy or too complicated and time consuming, making them not scalable to large scale applications. Third, for the online multiple kernel problem, an extension of the traditional single kernel learning problem by learning a linear combination of multiple kernel functions, there is no effective and efficient algorithm for large scale applications. Our research aims to address these challenges.

For the first challenge, we propose the Sparse Passive Aggressive Online Learn-

ing algorithm (SPA). The main idea is to adopt a stochastic updating strategy and add SV's wisely. Since no SV is removed during the learning process, we not only bound the number of SV's for classifiers in all iterations, but also bound the number of SV's in the averaged classifier, making it applicable to batch testing.

For the second challenge, we propose two functional approximation 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. These two algorithms achieve high efficiency by approximating the kernel learning problem with a linear learning problem.

For the third challenge, we propose a group of online multiple kernel learning algorithms. We extend the three proposed online kernel learning algorithms to scale up each of the component classifiers and learn the combination weights by the Hedge algorithm.

For all the proposed algorithms, we conduct theoretical analysis and empirical experiments in comparison with many existing algorithms. The encouraging results valid the advantages of these proposed algorithms.

6.2 Future Work

In this section, we try to list some potential problems for future research.

First, the existing Nyström kernel matrix approximation are built on the first B support vectors and will be fixed along the whole learning process. This is somehow sub-optimal since the first a few support vectors might not be the optimal choice of the most representative support vectors. A promising future direction is to develop an online algorithm that can update the Nyström SV set when possible. This should improve the accuracy in the case when the first a few SV's are not very representa-

tive. Currently, only a few works tried to address this problem [51, 37]

Second, in the OMKL section, we currently pay equal attention to all the kernels since it is impossible to determine which kernel is the optimal one before training. A possible direction is to update the number of fourier vectors D and the Nyström budget B according to the performance of certain component classifier along the learning process. So we can allocate more resources to more powerful kernels.

Third, in this work, we mainly focus on binary classification, though multi-class classification and regression are also considered. While the proposed algorithms are not limited to these applications. In the future, we wish to address more real world applications such as ranking and recommendation system.

Appendix: The Open Source Tool-box

To facilitate other researchers in the online kernel learning field. We provide an open source C++ tool-box for all the proposed algorithms and all the compared OKL algorithms in this dissertation, including:

- Online Kernel Learning algorithms: Perceptron, OGD, PA, PA-I and PA-II
- Budget Online Kernel Learning algorithms: RBP, Forgetron, Projectron, Projectron++, BOGD, BPA-S, SPA, FOGD and NOGD
- Online Multiple Kernel Learning algorithms: U, DD, SD,
- Budget Online Multiple Kernel Learning algorithms: RBP, Forgetron, BOGD, BPA-S, SPA, FOGD and NOGD.

See `https://github.com/LIBOL/KOL` and `http://lsokl.stevenhoi.org/` for details.

Bibliography

[1] F. R. Bach. Consistency of the group lasso and multiple kernel learning. *The Journal of Machine Learning Research*, 9:1179–1225, 2008.

[2] F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[3] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3):143–167, 2007.

[4] N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

[5] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

[6] N. Chen, S. C. Hoi, S. Li, and X. Xiao. Mobile app tagging. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 63–72. ACM, 2016.

[7] R. Chitta, R. Jin, T. C. Havens, and A. 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.

[8] R. Chitta, R. Jin, and A. Jain. Efficient kernel clustering using random fourier features. In *IEEE International Conference on Data Mining*, 2012.

[9] C. Cortes, M. Mohri, and A. Talwalkar. On the impact of kernel approximation on learning accuracy. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[10] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[11] K. Crammer, J. S. Kandola, and Y. Singer. Online classification on a budget. In *Neural Information Processing Systems*, volume 2, page 5, 2003.

[12] K. Crammer, A. Kulesza, and M. Dredze. Adaptive regularization of weight vectors. In *Advances in neural information processing systems*, pages 414–422, 2009.

[13] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[14] K. Crammer and Y. Singer. On the learnability and design of output codes for multi-class problems. *Machine Learning*, 47(2-3):201–233, 2002.

[15] O. Dekel. From online to batch learning with cutoff-averaging. In *Advances in Neural Information Processing Systems*, pages 377–384, 2009.

[16] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The forgetron: A kernel-based perceptron on a fixed budget. In *Neural Information Processing Systems*, 2005.

[17] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput.*, 37(5):1342–1372, 2008.

[18] O. Dekel and Y. Singer. Data-driven online to batch conversions. In *Advances in Neural Information Processing Systems*, pages 267–274, 2005.

[19] M. Dredze, K. Crammer, and F. Pereira. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine Learning*, pages 264–271, 2008.

[20] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

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

[22] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Maching Learning.*, 37(3):277–296, 1999.

[23] C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2(Dec):213–242, 2001.

[24] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268, 2011.

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

[26] S. C. Hoi, R. Jin, P. Zhao, and T. Yang. Online multiple kernel classification. *Machine Learning*, 90(2):289–316, 2013.

[27] S. C. Hoi, M. R. Lyu, and E. 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.

[28] S. C. Hoi, J. Wang, and P. Zhao. Libol: A library for online learning algorithms. *Journal of Machine Learning Research*, 15:495–499, 2014.

[29] S. C. H. Hoi, R. Jin, and M. R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *Proceedings of the International Conference on Machine Learning*, pages 361–368, Corvalis, Oregon, 2007.

[30] S. C. H. Hoi, R. Jin, P. Zhao, and T. Yang. Online multiple kernel classification. *Machine Learning*, 90(2):289–316, 2013.

[31] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[32] R. Jin, S. C. Hoi, and T. Yang. Online multiple kernel learning: Algorithms and mistake bounds. In *Algorithmic Learning Theory*, pages 390–404. Springer, 2010.

[33] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In *Neural Information Processing Systems*, pages 785–792, 2001.

[34] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *Signal Processing, IEEE Transactions on*, 52(8):2165–2176, 2004.

[35] G. R. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004.

[36] B. Li, P. Zhao, S. C. Hoi, and V. Gopalkrishnan. Pamr: Passive aggressive mean reversion strategy for portfolio selection. *Machine learning*, 87(2):221–258, 2012.

[37] H. Li and L. Zhang. Dynamic subspace update with incremental nyström approximation. In *Asian Conference on Computer Vision*, pages 384–393. Springer, 2010.

[38] M. Li, W. Bi, J. T. Kwok, and B.-L. Lu. Large-scale nyström kernel matrix approximation using randomized svd. *IEEE transactions on neural networks and learning systems*, 26(1):152–164, 2015.

[39] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1-3):361–387, 2002.

[40] J. Luo, F. Orabona, M. Fornoni, B. Caputo, and N. Cesa-Bianchi. Om-2: An online multi-class multi-kernel learning algorithm. In *In Proceeding of CVPR 2010, Online Learning for Computer Vision Workshop*, number EPFL-CONF-192505, 2010.

[41] J. Ma, L. K. Saul, S. Savage, and G. 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.

[42] A. F. Martins, M. A. Figueiredo, P. M. Aguiar, N. A. Smith, and E. P. Xing. Online multiple kernel learning for structured prediction. *arXiv preprint arXiv:1010.2770*, 2010.

[43] F. Orabona, L. Jie, and B. Caputo. Online-batch strongly convex multi kernel learning. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 787–794. IEEE, 2010.

[44] F. Orabona, L. Jie, and B. Caputo. Multi kernel learning with online-batch optimization. *The Journal of Machine Learning Research*, 13(1):227–253, 2012.

[45] F. Orabona, J. Keshet, and B. Caputo. The projectron: a bounded kernel-based perceptron. In *Proceedings of the International Conference on Machine Learning*, pages 720–727, 2008.

[46] F. Orabona, J. Keshet, and B. Caputo. Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10:2643–2666, 2009.

[47] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Neural Information Processing Systems*, 2007.

[48] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782. ACM, 2007.

[49] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

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

[51] A. Rudi, R. Camoriano, and L. Rosasco. Less is more: Nyström computational regularization. In *Advances in Neural Information Processing Systems*, pages 1657–1665, 2015.

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

[53] D. Sahoo, S. C. Hoi, and B. Li. Online multiple kernel regression. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 293–302. ACM, 2014.

[54] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Conference on Learning Theory*, pages 416–426, 2001.

[55] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[56] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, 127(1):3–30, 2011.

[57] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *arXiv preprint arXiv:1212.1824*, 2012.

[58] A. J. Smola and B. Schölkopf. *Learning with kernels*. Citeseer, 1998.

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

[60] N. Subrahmanya and Y. C. Shin. Sparse multiple kernel learning for signal processing applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):788–798, 2010.

[61] A. Talwalkar, S. Kumar, and H. A. Rowley. Large-scale manifold learning. In *Computer Vision and Pattern Recognition*, 2008.

[62] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

[63] T. van Erven and W. M. Koolen. Metagrad: Multiple learning rates in online learning. In *Advances in Neural Information Processing Systems*, pages 3666–3674, 2016.

[64] V. N. Vapnik and V. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[65] M. Varma and B. R. Babu. More generality in efficient multiple kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1065–1072. ACM, 2009.

[66] J. Wang, S. C. Hoi, P. Zhao, J. Zhuang, and Z.-y. Liu. Large scale online kernel classification. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1750–1756. AAAI Press, 2013.

[67] J. Wang, P. Zhao, and S. C. Hoi. Exact soft confidence-weighted learning. *arXiv preprint arXiv:1206.4612*, 2012.

[68] J. Wang, P. Zhao, and S. C. H. Hoi. Cost-sensitive online classification. In *IEEE International Conference on Data Mining*, pages 1140–1145, 2012.

[69] Z. Wang, K. Crammer, and S. Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13:3103–3131, 2012.

[70] Z. Wang and S. Vucetic. Twin vector machines for online learning on a budget. In *International Conference on Data Mining*, pages 906–917. SIAM, 2009.

[71] Z. Wang and S. Vucetic. Online passive-aggressive algorithms on a budget. In *International Conference on Artificial Intelligence and Statistics*, pages 908–915, 2010.

[72] C. K. I. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Neural Information Processing Systems*, pages 682–688, 2000.

[73] H. Xia, S. C. Hoi, R. Jin, and P. Zhao. Online multiple kernel similarity learning for visual search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(3):536–549, 2014.

[74] H. Xia, P. Wu, and S. 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.

[75] T. Yang, Y. Li, M. Mahdavi, R. Jin, and Z. hua Zhou. Nystrom method vs random fourier features: A theoretical and empirical comparison. In *Neural Information Processing Systems*, 2012.

[76] S. Yu, T. Falck, A. Daemen, L.-C. Tranchevent, J. A. Suykens, B. De Moor, and Y. Moreau. L 2-norm multiple kernel learning and its application to biomedical data fusion. *BMC bioinformatics*, 11(1):1, 2010.

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

[78] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel svm on limited resources: A low-rank linearization approach. In *AISTATS*, volume 22, pages 1425–1434, 2012.

[79] L. Zhang, R. Jin, C. Chen, J. Bu, and X. He. Efficient online learning for large-scale sparse kernel logistic regression. In *AAAI*, 2012.

[80] L. Zhang, J. Yi, R. Jin, M. Lin, and X. He. Online kernel learning with a near optimal sparsity bound. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 621–629, 2013.

[81] P. Zhao, S. C. H. Hoi, and R. Jin. Double updating online learning. *Journal of Machine Learning Research*, 12:1587–1615, 2011.

[82] P. Zhao, J. Wang, P. Wu, R. Jin, and S. C. Hoi. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. In *ICML*, 2012.

[83] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning(ICML 2003)*, pages 928–936, 2003.