Singapore Management University
Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

1-2019

Large scale online multiple kernel regression with application to time-series prediction

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

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

Bin LIN

DOI: https://doi.org/10.1145/3299875

Citation

SAHOO, Doyen; HOI, Steven C. H.; and LIN, Bin. Large scale online multiple kernel regression with application to time-series prediction. (2019). *ACM Transactions on Knowledge Discovery from Data*. 13, (1), 9:1-33. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4383

Large Scale Online Multiple Kernel Regression with Application to Time-Series Prediction

DOYEN SAHOO and STEVEN C. H. HOI, Singapore Management University
BIN LI, Wuhan University

Kernel-based regression represents an important family of learning techniques for solving challenging regression tasks with non-linear patterns. Despite being studied extensively, most of the existing work suffers from two major drawbacks as follows: (i) they are often designed for solving regression tasks in a batch learning setting, making them not only computationally inefficient and but also poorly scalable in real-world applications where data arrives sequentially; and (ii) they usually assume that a fixed kernel function is given prior to the learning task, which could result in poor performance if the chosen kernel is inappropriate. To overcome these drawbacks, this work presents a novel scheme of Online Multiple Kernel Regression (OMKR), which sequentially learns the kernel-based regressor in an online and scalable fashion, and dynamically explore a pool of multiple diverse kernels to avoid suffering from a single fixed poor kernel so as to remedy the drawback of manual/heuristic kernel selection. The OMKR problem is more challenging than regular kernel-based regression tasks since we have to on-the-fly determine both the optimal kernel-based regressor for each individual kernel and the best combination of the multiple kernel regressors. We propose a family of OMKR algorithms for regression and discuss their application to time series prediction tasks including application to AR, ARMA, and ARIMA time series. We develop novel approaches to make OMKR scalable for large datasets, to counter the problems arising from an unbounded number of support vectors. We also explore the effect of kernel combination at prediction level and at the representation level. Finally, we conduct extensive experiments to evaluate the empirical performance on both real-world regression and times series prediction tasks.

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

Additional Key Words and Phrases: Online learning, multiple kernel regression, large-scale kernel learning, time-series prediction

ACM Reference format:
Doyen Sahoo, Steven C. H. Hoi, and Bin Li. 2019. Large Scale Online Multiple Kernel Regression with Application to Time-Series Prediction. *ACM Trans. Knowl. Discov. Data* 13, 1, Article 9 (January 2019), 33 pages.
https://doi.org/10.1145/3299875

This research is supported by the National Research Foundation Singapore under its AI Singapore Programme [AISG-RP-2018-001], and the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative. This research is also supported by the MOE project of Humanities and Social Science (18YJCZH072), the Academic Team Building Plan for Young Scholars from Wuhan University (Whu2016012), and the Fundamental Research Funds for the Central Universities (1203-410500077/413000031).

Authors' addresses: D. Sahoo and S. C. H. Hoi, School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902; emails: {doyens, chhoi}@smu.edu.sg; B. Li, Economics and Management School, Wuhan University, Wuhan 430072, P. R. China; email: binli.whu@whu.edu.cn.

1 INTRODUCTION

Kernel methods have been extensively studied for regression tasks and found successes in many real-world applications [53, 54]. In contrast to linear regression methods, kernel-based regression methods are able to tackle challenging non-linear regression tasks using the kernel trick that implicitly maps data from the original space to a high or even infinite dimensional space by means of a kernel function. Although a variety of kernel methods have been proposed for regression tasks [53], most conventional kernel methods suffer from two major drawbacks. First of all, they are often designed for solving regression tasks in a batch learning setting. This often results in a high re-training cost when new training data becomes available, making them poorly scalable in many real-world applications where data arrives sequentially. Second, they usually assume that prior to the learning task, a fixed kernel function is given either by manual selection or via cross validation. This could result in poor performance if the chosen kernel is inappropriate in a new environment, which happens commonly for some real-world applications, such as time series prediction where data observations can be non-stationary and the optimal kernel function may change over time.

To overcome the above drawbacks, we propose a novel scheme of Online Multiple Kernel Regression (OMKR), which sequentially learns a kernel-based regressor with multiple kernels in an online fashion for regression tasks. On one hand, the proposed OMKR technique, as an online learning method that often makes simple incremental update for a new training data example, avoids the expensive re-training cost of conventional batch kernel methods, and thus significantly improves the efficiency and scalability, especially when handling data stream applications. On the other hand, OMKR explores a pool of multiple diverse kernels to remedy the drawback of using a single fixed kernel by existing kernel-based regression methods that often suffer considerably when the single kernel is inappropriate. The proposed OMKR problem is however very challenging since we not only need to sequentially learn the optimal kernel-based regressor for each individual kernel in the pool, but also need to simultaneously decide the best way of combining the multiple kernel regressors on the fly at every learning round. We tackle the challenges by (i) exploring two online kernel regression algorithms, Widrow-Hoff learning [60] and NORMA learning [26], for online regression tasks with each individual kernel; and (ii) determining the best combination of the multiple kernel regressors by applying two online learning techniques: *Hedge* algorithm [13] that can track the best kernel regressor, and *Online Gradient Descent*(OGD)[68] that can find the optimal linear combination. To validate the efficacy of the proposed method, we conduct extensive experiments by evaluating the proposed algorithms on both real-world regression and time series datasets, in which our empirical results show that OMKR outperforms conventional single kernel online regression approaches for most cases, especially for time series prediction tasks.

Due to the *curse of kernelization* [32], methods that perform online learning with kernels suffer from an unbounded number of support vectors (SVs). This problem is more severe in the case of multiple kernels, especially if there are some poor performing kernels. Further, existing work in Online Multiple Kernel Learning (MKL) attempts to combine predictive power of multiple kernels only at the prediction level, and does not try to exploit multiple kernel combination at representation level. To address these issues, we develop algorithms for large scale OMKR, which are based on kernel approximation techniques [43, 61, 62]. We demonstrate through extensive experiments the scalability of the proposed methods, often coupled with improved performance. We also evaluate kernel combination strategies, and empirically study the behavior of combining multiple kernels at a prediction level and at the representation level.

We discuss a natural extension of the OMKR algorithms to the process of online learning for time-series prediction. In particular we explore how OMKR can automatically determine the appropriate window size to be considered for the learning procedure, and show how it can be applied for Autoregressive (AR), Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA) time-series modeling. Further, we present how OMKR can also be useful for online learning for non-linear time-series prediction.

We note that a short version of this work appeared in ACM SIGKDD 2014 [47]. We offer substantially new content and empirical studies.

The rest of the article is organized as follows: In Section 2, we review the related work, specifically focussing on contributions in literature in the domain of online learning and MKL. In Section 3, we present the main framework for OMKR. This is followed by the presentation OMKR algorithms for large-scale learning in Section 4. We discuss the application time-series prediction in Section 5. In Section 6, we present our detailed experimental results, and finally conclude in Section 7.

2 RELATED WORK

Our work is related to the fields of *online learning* and *learning with kernels*, which we review in the context of OMKR. We also discuss the computational challenges associated with using kernel-based models in the online setting. We also review work associated with time-series prediction.

2.1 Online Learning

Online learning refers to a class of scalable algorithms that learn sequentially from streamlining data [7, 20, 21, 52, 63], and they have been extensively explored indifferent contexts and applications [9, 37, 65]. The general problem setting is to receive instances one at a time, make a prediction for each instance, and based on the feedback available, update the model. In this work, our focus is on supervised online learning tasks. Many early supervised online learning algorithms were designed for linear models, starting with the Perceptron [44]. More recently a series of *First -Order* online learning algorithms were developed based on the maximum-margin principle, including OGD [68] and Passive Aggressive Algorithms [9, 10]. The first-order algorithms were further advanced by *second -order algorithms* that improved convergence by considering second-order information [12]. Another closely related area is in supervised online learning is *prediction with expert advice* [13, 29, 59], where predictions from multiple experts are weighted and update in every online iteration. One of the most well-known algorithms is the Hedge Algorithm [13], which was a direct generalization of Weighted Majority Algorithm [29].

A major drawback of these approaches is the inability to learn non-linear patterns in the data, thus limiting there real-world applications. A solution for this was developed through online learning with kernels [26] (and more recently Online Learning with Deep Neural Networks [48]). Many extensions to this paradigm were proposed, specifically in the context of regression: Naive Online Regret Minimization (NORMA) [26], Online Passive Aggressive Regression [9], Sparse Implicit Online Learning with Kernels (ILK and SILK) [51], Kernel Least Mean Squares [31], Primal Online Algorithm (PRIONA) [5], and Quantized Kernel Least Mean Squared Algorithm [8]. While these methods provide with promising directions to learn non-linearity, their empirical performance is heavily reliant on the choice of the kernel, a difficult task, which is specifically more challenging in the online setting where traditional validation data is not available, and the appropriate kernel selection has to be done on the fly. Furthermore, usage of a single kernel function may restrict the complexity of the pattern that is learnable, in particular if different kernels can offer complementary information. Also, in scenarios with multimodal data, using a single kernel to combine all the data sources may not be feasible.

Another problem with existing online kernel methods is the *curse of kernelization*, where the number of SVs is unbounded. In the online setting, every instance that suffers a non-zero loss becomes a SV, which results in updating the prediction function such that every new prediction requires a kernel function computation with all the SVs in memory. In particular, in the context of regression, this problem is more severe as usage of popular regression loss functions (e.g., squared loss) would result in a non-zero loss on every instance, which would result in every instance becoming a SV. To address the scalability concerns of kernel-based online learning, many studies focus on the budget issue [6, 10]. These speed up the algorithms by bounding the number of SVs. Some well-known example algorithms include Forgetron [11], Projectron [41], and the Bounded Online Gradient Descent (BOGD) [66], Sparse passive aggressive online learning with kernels [33, 34]. Recently, kernel approximation strategies have been proposed to address the scalability concerns [32], and kernel approximation via random features with re-parameterization [39].

2.2 Multiple Kernel Learning

Kernel methods have gained popularity due to their ability to learn non-linear patterns in the data [50]. Several Kernel methods have been applied to regression tasks [54]. Most kernel methods often assume that a predefined parametric kernel is given a priori, where the parameters are chosen either manually or via cross validation. Kernel learning aims to learn an effective kernel from data automatically. Some studies have attempted to learn kernel functions or matrices from labeled and unlabeled data. Examples include marginalized kernels [25], idealized kernel learning [27], graph-based spectral kernel learning [4, 19], and non-parametric kernel learning [17, 67]. These methods often follow a batch (and transductive) learning setting and thus are difficult to be applied in an online learning scenario.

A prevalent kernel learning technique is MKL [28], which aims to find the optimal combination of multiple kernels. Unlike most existing MKL techniques that are batch learning [15, 28, 55], our work focuses on online regression tasks, and is related to existing online MKL studies that focus on classification tasks [18, 23, 46] and that address structured prediction [36]. Existing studies in Online Learning with Multiple Kernels have several limitations as follows: (i) the kernel combination techniques explored may not be suitable for regression tasks, as they are designed for tracking the best kernel, and not identifying the best kernel combination; (ii) they can be computationally very expensive, in particular for regression tasks, where every instance becomes a SV. This is even more serious for the MKL scenario, where there are many kernel functions, and the poor performing kernels would tend to accrue more SVs; and (iii) existing approaches do not account for differences in addressing stationary and concept-drifting applications, or time-series prediction. In our work, we address all these challenges, in a unified framework for OMKR.

2.3 Learning for Time-Series Prediction

Time-Series prediction is a problem of predicting future values based on current and past values [24, 42, 49]. Traditionally, this prediction has been done through the usage of AR Models or Kalman Filters [24]. Recent years have witnessed the popularization of the usage of SV regression for this task which dates back to 1997 [38], and has been extended to several applications of time-series prediction [49] (e.g., finance [56], air quality [35], utility load [16], machine reliability[22]). Unlike the traditional approaches for time-series prediction, kernel-based SV regression have gained popularity as they can generalize to non-linear processes. These approaches suffer from the following two major limitations: (i) all of these approaches are designed for the batch setting, and are not suitable for online settings; and (ii) selection of the appropriate kernel must be done empirically via some validation setting. Recently, there have been efforts in addressing Online Learning for Time-Series Prediction [1], which focused on AR and ARMA models. This was followed up with approaches for ARIMA models [30], and for time-series prediction in scenarios with missing data [2]. However, these approaches considered only linear models. In contrast, we consider not only kernels to induce non-linearity, but also use multiple kernels in order to automatically perform kernel selection.

3 ONLINE MULTIPLE KERNEL REGRESSION

In this section, we present the proposed OMKR scheme. We will first motivate the problem by introducing the formulation of batch MKL. We then present our OMKR framework, the detailed algorithms for addressing different challenges, and finally theoretical analysis of OMKR.

3.1 Problem Setting

Consider a set of training examples $\{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$, where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ and a collection of m kernel functions $\mathcal{K} = \{\kappa_i : \chi \times \chi \to \mathbb{R}, i = 1, \ldots, m\}$. MKL aims to learn a kernel-based prediction model by identifying the best linear combination of the m kernels, that is, a weighted combination $\theta = (\theta_1, \ldots, \theta_m)$. The learning task can be cast into the following optimization [28]:

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

where $\Delta = \{\theta \in \mathbb{R}^m_+ | \theta^T \mathbf{1}_m = 1\}$, $K(\theta)(\cdot,\cdot) = \sum_{i=1}^{T} \theta_i \kappa_i(\cdot,\cdot)$ and $\ell(f(\mathbf{x}_t), y_t)$ is a convex loss function.

The above convex optimization problem of regular batch MKL can be solved by different schemes [15, 55, 64]. Despite being studied extensively, it remains very challenging when solving the batch MKL for large-scale applications. Besides, similar to most batch kernel methods, regular MKL has some drawbacks as follows: (i) the trained model, if it is not re-trained with new data, may work poorly for non-stationary data in a new environment; but (ii) the re-training cost is extremely expensive for data streams, making it non-scalable.

3.2 OMKR Framework

To overcome the limitations of MKL for a regression task, we propose a new scheme of OMKR by applying the emerging online MKL principle [18] for tackling regression tasks, which attempts to sequentially learn the online multiple-kernel regressor given a new data example using a two-step updating scheme as follows: (i) update the set of kernel-based regressors for each individual kernel; and (ii) update the weights for combining the multiple kernel regressors. In the following, we discuss the details of the proposed algorithms for tackling online regression tasks at each of the two steps.

3.2.1 Learning Online Kernel-Based Regressors. The goal of this task is to learn a regression function $f_t \in \mathcal{H}_\kappa$ in an online setting, where $\mathcal{H}_\kappa$ a reproducing kernel Hilbert space (RKHS) induced by a given specific kernel $\kappa \in \mathcal{K}$. We solve this task by exploring two online regression solutions: Kernel Widrow-Hoff [60] and NORMA [26], which follows the same principle of OGD [68] for online convex optimization and but optimizes two slightly different objective functions.

Kernel Widrow-Hoff Learning. Given a sequence of data instances $(\mathbf{x}_i, y_i), i = 1, \ldots, T$, the goal of kernelized Widrow-Hoff learning is to minimize the total cumulative loss over the whole regression task $\mathcal{L}$ defined as follows:

$$\mathcal{L} = \Sigma_{t=1}^{T} \ell(f_t(x_t), y_t) \triangleq \Sigma_{t=1}^{T} \mathcal{L}_t(f_t), \tag{2}$$

where $f_t(x_t)$ is the prediction made by a kernel regressor on the tth instance, $\ell(f_t(\mathbf{x}_t), y_t)$ denoted by $\mathcal{L}_t(f_t)$ for short, is a convex loss function. Following OGD [68], we have the following online update rule given a data instance $(\mathbf{x}_t, y_t)$:

$$f_{t+1} \leftarrow f_t - \eta_t \nabla \mathcal{L}_t(f_t), \tag{3}$$

where $\eta_t > 0$ is a learning rate parameter that can be either a small constant $\eta_t = \eta$ used in Widrow-Hoff [60] or a factor depending on t. When choosing the squared loss for ℓ:

$$\ell(f_t(\mathbf{x}_t), y_t) = (f_t(\mathbf{x}_t) - y_t)^2,$$

we have the online updating rule expressed explicitly as

$$f_{t+1}(\cdot) \leftarrow f_t(\cdot) - \eta_t(f_t(\mathbf{x}_t) - y_t)\kappa(\mathbf{x}_t, \cdot). \tag{4}$$

NORMA. The above method has two potential drawbacks. First, it may lead to overfitting when dealing with noisy data. Second, due to the use of squared loss, almost every training instance will be added as SVs (unless $f_t(x_t)$ is identical to y_t), making the prediction function computationally intensive when handling large-scale datasets. To overcome these drawbacks, we explore another online regression scheme by following the idea of NORMA [26], which replaces $\mathcal{L}_t(f_t)$ by the following regularized loss:

$$\mathcal{L}_t(f_t) = \frac{\lambda}{2}||f||^2_{\mathcal{H}_\kappa} + \ell(f_t(\mathbf{x}_t), y_t). \tag{5}$$

By the OGD principle, we have the online updating rule as

$$f_{t+1} \leftarrow (1 - \eta_t\lambda)f_t - \eta_t \nabla \ell(f_t(\mathbf{x}_t), y_t), \tag{6}$$

where $\eta_t > 0$ is the learning rate parameter. Instead of using the square loss, we exploit the ϵ-insensitive loss function which is defined as

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

where ϵ represents the width of the insensitivity zone. We can further modify the loss function by making ϵ as a variable of the optimization:

$$\ell_t(f_t, \mathbf{x}_t) = \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon) + \nu\epsilon_t, \tag{7}$$

where $\nu > 0$ is a parameter, and ϵ_t is a variable to be updated in online learning process. Using the above loss function, we can derive the online updating rule for NORMA:

$$f_{t+1} \leftarrow \begin{cases} (1 - \eta_t\lambda)f_t + \eta_t * sgn(d)\kappa(\mathbf{x}_t, \cdot) & \text{if } |d| > \epsilon_t \\ (1 - \eta_t\lambda)f_t & \text{otherwise} \end{cases} \tag{8}$$

$$\epsilon_{t+1} \leftarrow \begin{cases} \epsilon_t + (1 - \nu)\eta_t & \text{if } |d| > \epsilon_t \\ \epsilon_t - \eta_t\nu & \text{otherwise}' \end{cases} \tag{9}$$

where we denote $d = y_t - f(\mathbf{x}_t)$.

Remark. For both of the above methods, at the end of each online learning round, we can express the prediction function of the regressor as a kernel expansion [50]:

$$f_{t+1}(\mathbf{x}) = \Sigma_{i=1}^t \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}),$$

where the α_i coefficients are computed based on the updating rules in (4) or (8). When $\alpha_i \neq 0$, the ith instance is often called as a SV. Thus, the time complexity for prediction is linear with respect to the number of SV's. When using the squared loss, we will have $\alpha_i \neq 0$ for almost every instance, leading to a large number of SVs. By contrast, when using the ϵ-insensitive loss, whenever the difference between the prediction on the ith instance $f_i(x_i)$ and y_i is small enough, i.e., within the ϵ tube, we have $\alpha_i = 0$, which thus generates a much smaller SV size and significantly improves the prediction efficiency.

3.2.2 Learning the Best Kernel Combination. The previous online kernel regression method allows us to learn a set of kernel regressors $f_t^i \in \mathcal{H}_{\kappa_i}, i = 1, \ldots, m$ with respect to the pool of multiple diverse kernels $\mathcal{K}$. The idea of OMKR is to learn an effective regressor $F_t(\mathbf{x})$ by combining the set of multiple kernel regressors:

$$F_t(\mathbf{x}) = \sum_{i=1}^{m} w_t^i f_t^i(\mathbf{x}), \tag{10}$$

where $w_t^i \in \mathbb{R}$ denotes the combination weight for the i-th kernel regressor. The remaining problem then is to determine the appropriate combination weights $\mathbf{w}_t$ for the set of kernels. We note that this is a very challenging task since we may not have prior knowledge for empirical performance of each kernel, and the optimal combination weights may even change over time in the online learning process especially when dealing with non-stationary data.

One naive solution is to simply adopt a uniform combination for all the kernels, i.e., $w_t^i = 1/m$, which does explore all the kernels, but often results in sub-optimal performance, as observed in our empirical studies. In this section, we attempt to learn the best kernel combination weights by exploring two different online learning algorithms: the Hedge algorithm [13] and the OGD algorithm [68]. We will first present each algorithm in detail and finally discuss their strengths and weaknesses for different scenarios.

Hedge Algorithm. The Hedge algorithm is the most popular online algorithm for solving the problem of decision-theoretic online learning or known as prediction with expert advice [7, 59]. Specifically, by treating each online kernel regressor as an expert, the Hedge algorithm aims to minimize the regret of the learner for the regression task, which is the difference between the learner's cumulative loss and the cumulative loss of the best kernel regressor. In theory, Hedge can achieve an optimal upper bound of regret $O(T \ln m)$ with T learning rounds and m kernel regressor experts. It is thus an ideal online learning algorithm for tracking the best online kernel regressor especially when there is some kernel regressor significantly dominates the rest.

Specifically, the Hedge algorithm runs in a fairly simple way. Consider the OMKR problem, at the beginning, the combination weights $\mathbf{w}_t$ are initialized as a uniform distribution, i.e., $w_1^i = 1/m, i = 1, \ldots, m$. At the end of each learning round, according to the performance of the multiple kernel regressors, the weights are updated by

$$w_{t+1}^i = w_t^i \beta^{\ell_t^i}, i = 1, \ldots, m, \tag{11}$$

where $\beta \in (0, 1)$ is a discounting (learning rate) parameter, and ℓ_t^i denotes the loss suffered by the ith kernel regressor at round t, i.e., $\ell_t^i = \ell(f_t^i(x_t), y_t)$. Finally, we normalize all w_{t+1}^i's to ensure the combination weights as a distribution.

We refer to the proposed OMKR algorithm that adopts the Hedge algorithm as the *Deterministic OMKR* (Hedge) algorithm, as shown in Algorithm 1. In the algorithm, we can update each kernel regressor f_{t+1}^i by adopting either the Widrow-Hoff learning in (4) or NORMA in (8).

Although Hedge is ideal for tracking the best kernel regressor, it is not always perfect for solving a practical OMKR problem since our goal is to learn the best combination of multiple kernels. In the following, we present an OGD based algorithm that attempts to learn the optimal linear combination of multiple kernel regressors.

OGD Algorithm. Our goal is to learn the optimal combination weight vector $\mathbf{w}_t \in \mathbb{R}^m$ for combining the multiple kernel regressors. It can be cast into the following online optimization:

$$\mathbf{w}_{t+1} \leftarrow \arg\min_{\mathbf{w}} \ell(\mathbf{w}^\top \mathbf{f}_t(\mathbf{x}_t), y_t) \triangleq (\mathbf{w}^\top \mathbf{f}_t(\mathbf{x}_t) - y_t)^2, \tag{12}$$

where $\mathbf{f}_t(\mathbf{x}_t)$ is a vector representing the predictions made by all the kernel regressors on instance $\mathbf{x}_t$, and ℓ is a loss function denoting the loss suffered by the OMKR. We simply adopt the squared

ALGORITHM 1: Deterministic OMKR (Hedge)

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1, \dots, m$
  - Discounting Parameter: $\beta \in (0, 1)$
  - Step size parameter for each kernel: $\eta$
  - Regression parameters: $\lambda$ and $\nu$ for OMKR(NORMA)
Initialization: $\mathbf{f}_1 = \mathbf{0}$, $\mathbf{w}_1 = \frac{1}{m}\mathbf{1}$
  for t = 1, ..., T do
    Receive instance: $x_t$
    Predict $\hat{y_t} = \sum_{i=1}^{m} w_t^i f_t^i(x_t)$
    Reveal true value $y_t$
    for i = 1, ..., m do
      Set $\ell_t^i = \ell(f_t^i(x_t), y_t)$;
      Update $f_{t+1}^i$ = Equation (4) OR (8)
      Update $w_{t+1}^i = w_t^i \beta^{\ell_t^{*i}}$ where $\ell_t^{*i} = (f_t^i(x_t) - y_t)^2$;
    end for
    Set $w_{t+1}^i = \frac{w_t^i}{W_t}$ where $W_t = \sum_{i=1}^{m} w_t^i$, $i = 1, \dots, m$
  end for
```

loss in our solution (though it may also include a regularizer). Following the OGD, we can derive the updating rule as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_w(\hat{y_t} - y_t)\mathbf{f}_t(\mathbf{x}_t), \tag{13}$$

where η_w is a learning rate parameter, and $\hat{y_t} = \mathbf{w}^\top \mathbf{f}_t(\mathbf{x}_t)$.

Using the above OGD algorithm for learning the optimal combination weights, we propose another OMKR scheme, called *Deterministic OMKR*(OGD), as shown in Algorithm 2. Like in OMKR(Hedge), we can also update each kernel regressor by either Widrow-Hoff in (4) or NORMA in (8).

ALGORITHM 2: Deterministic OMKR (OGD)

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1, \dots, m$
  - Learning rate parameter: $\eta_w$
  - Step size parameter for each kernel: $\eta$
  - Regression parameters: $\lambda$ and $\nu$ for OMKR(NORMA)
Initialization: $\mathbf{f}_1 = \mathbf{0}$, $\mathbf{w}_1 = \mathbf{0}$
  for t = 1, ..., T do
    Receive instance: $x_t$
    Predict $\hat{y_t} = \sum_{i=1}^{m} w_t^i f_t^i(x_t)$
    Reveal true value $y_t$
    for i = 1, ..., m do
      Set $\ell_t^i = \ell(f_t^i(x_t), y_t)$;
      Update $f_{t+1}^i$ = Equation (4) OR (8)
    end for
    Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_w(\hat{y_t} - y_t) f_t(x_t)$
  end for
```

Remark. In online MKL work related to classification [18, 23], Hedge algorithm was used to combine multiple predictions. In contrast, our proposed OGD approach interprets the kernel predictions as new rich features which can be combined linearly. In terms of update rules, Hedge makes multiplicative updates while OGD makes additive updates. Further, for the combination weight vector $\mathbf{w}_t$, Hedge always keep $\mathbf{w}_t$ a distribution ($w_t^i \geq 0$ and $\sum_i w_t^i = 1$) while OGD is able to learn any real-valued vector for $\mathbf{w}_t$. In general, both Hedge and OGD have their different merits. Hedge is good at tracking the best kernel regressor, while OGD is good at learning the optimal combination of multiple kernel regressors. However, OGD often suffers from slow convergence rate. In practice, the empirical performance of OMKR(Hedge) and OMKR(OGD) may vary a lot in different scenarios. Due to the nature of multiplicative update of Hedge, it converges quickly, and in an online setting, may tend to achieve better performance than OGD, if the dataset is small, or if the pattern changes due to non-stationarity. We conduct more in-depth analysis through our extensive experimental studies in Section 6.

3.3 Theoretical Analysis

Without loss of generality we assume that $\forall\, i,\ \forall\, t,\ \kappa^i(x_t \cdot x_t) \leq 1$, and $\ell_t(f_t^i(\mathbf{x}_t), y_t) \leq 1$. We will demonstrate how the loss suffered by the OMKR algorithms has sub-linear regret with respect to the best fixed kernel model.

Theorem 3.1. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (Hedge) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \frac{\ln(\frac{1}{\beta})}{1-\beta} \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\ln(m)}{1-\beta}, \tag{14}$$

where

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^T (f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta} \right]. \tag{15}$$

Here, $\mathcal{L}_{OMKR}$ is the total loss suffered at each prediction, and due to the convexity of the loss function, we have

$$\mathcal{L}_{OMKR} = \Sigma_{t=1}^T \ell(\Sigma_{i=1}^m w_t^i f_t^i(x_t), y_t) \leq \Sigma_{t=1}^T \Sigma_{i=1}^m w_t^i \ell(f_t^i(x_t), y_t),$$

and by choosing $\beta = \frac{\sqrt{T}}{\sqrt{T}+\sqrt{\ln m}}$, we get

$$\mathcal{L}_{OMKR} \leq \left(1 + \sqrt{\frac{\ln m}{T}} \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \ln m + \sqrt{T \ln m}\right),$$

where $\mathcal{D}$ is a sequence of instances.

Proof. The proof follows from combining the proof of Hedge Algorithm and the Widrow-Hoff Regression. Let $\phi_t^i = ||f_t^i - f||_2^2$ for any $f \in \mathcal{H}_{\kappa_i}$. Also, let Δ_t denote the change in f during each update, such that $\Delta_t = \eta(f_t(x_t) - y_t)\kappa(x_t, \cdot)$. We also define $\ell_t = f_t(x_t) - y_t$ as the signed error suffered by f_t, and $\ell_t^* = f(x_t) - y_t$ be the signed error suffered by f:

$$\begin{aligned} \phi_{t+1}^i - \phi_t^i &= ||f_{t+1}^i - f||_2^2 - ||f_t^i - f||_2^2 \\ &= ||\Delta_t||_2^2 - 2(f_t^i - f) \cdot \Delta_t \\ &= \eta^2 \ell_t^{i2} \kappa(x_t \cdot x_t) - 2\eta \ell_t \kappa(x_t, \cdot) \cdot (f_t^i - f) \\ &\leq \eta^2 \ell_t^{i2} - 2\eta \ell_t^2 + 2\eta \ell_t \ell_t^* \\ &= \eta^2 \ell_t^{i2} - 2\eta \ell_t^{i2} + 2\eta \left[\left(\ell_t^i \sqrt{1-\eta}\right) \left(\frac{\ell_t^*}{\sqrt{1-\eta}}\right) \right]. \end{aligned}$$

The inequality follows from the assumption $\kappa(x_t \cdot x_t) \leq 1$.

$$\begin{aligned}\phi_{t+1}^i - \phi_t^i &\leq \eta^2 \ell_t^{i2} - 2\eta\ell_t^{i2} + \eta(1-\eta)\ell_t^{i2} + \frac{\eta}{1-\eta}\ell_t^{*2} \\ &= -\eta\ell_t^{i2} + \frac{\eta}{1-\eta}\ell_t^{*2}.\end{aligned} \tag{16}$$

In the above equation, we use the algebraic inequality $ab \leq (a^2 + b^2)/2$. From this, by assuming $f_1 = \mathbf{0}$, and using a telescoping sum, it is very simple to prove that

$$\mathcal{L}_{WH}^i \leq \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^T (f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta} \right], \tag{17}$$

where $\mathcal{L}_{WH}^i$ is the cumulative loss suffered by the regression function learnt by the Widrow-Hoff Algorithm in the RKHS by the ith kernel. As number of instances T grows large, the average loss per instance vanishes. Plugging this result into the Hedge Algorithm gives us the bound, which shows that the regret of OMKR is in $O(\sqrt{T})$. The choice of β maybe overestimated because of the assumption that the loss suffered by the algorithm is T. □

Theorem 3.2. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (OGD) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\eta G T}{2} + \frac{C}{2\eta}, \tag{18}$$

where G is constant that upper bounds the gradient of the loss function, and C is a constant that upper bounds the distance between any two weight vectors $\mathbf{w}$*, and*

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^T (f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta} \right]. \tag{19}$$

By choosing $\eta = \sqrt{\frac{C}{GT}}$*, we get:*

$$\mathcal{L}_{OMKR} \leq \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \sqrt{CGT},$$

where $\mathcal{D}$ *is a sequence of instances.*

Proof. The proof follows from combining the proof of OGD and the Widrow-Hoff Regression. OGD provides a sub-linear regret with respect to the best linear combination of its input features, which in this case is the output of individual kernel experts. If the loss of the optimal combination is denoted by $\ell(\mathbf{w}^*)$, it follows the $\ell(\mathbf{w}) < \ell(\mathbf{w}^*)\ \forall \mathbf{w}$.

We define the optimal kernel expert as $\tilde{\mathbf{w}}$ which corresponds to a one-hot vector, i.e., it is an indicator vector corresponding to the optimal expert. Since $\ell(\tilde{\mathbf{w}}) < \ell(\mathbf{w}^*)$, we get the result in the theorem. □

Next, we present the analysis of OMKR based on NORMA. The loss function of NORMA in the tth iteration for the ith kernel is denoted as

$$\ell_t(f_t^i) = \frac{\lambda}{2}||f||_{\mathcal{H}_\kappa}^2 + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t. \tag{20}$$

There are two sets of parameters to be updated as follows: f and ϵ. The loss function is convex in both these parameters. Since the update rule takes the form of OGD [68], both f and ϵ are learnt via the same update rule. Thus, we incorporate ϵ into the prediction function f.

Lemma 3.3. *After receiving a sequence of T instances, the cumulative loss suffered by NORMA regression is bounded as*

$$\mathcal{L}_{NORMA} \leq \min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^{T} \left(\frac{\lambda}{2} ||f||^2_{\mathcal{H}_\kappa} + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t \right) + \sqrt{C_2 G_2 T}, \tag{21}$$

where G_2 is constant that upper bounds the gradient of the loss function, and C_2 is a constant that upper bounds the distance between any two weight vectors in $\mathcal{H}_{\kappa_i}$.

Proof. Let $\phi_t^i = ||f_t^i - f||_2^2$ for any $f \in \mathcal{H}_{\kappa_i}$.

$$\begin{aligned} \phi_{t+1}^i - \phi_t^i &= ||f_{t+1}^i - f||_2^2 - ||f_t^i - f||_2^2 \\ &= ||f_t^i - \eta\nabla\ell_t^i(f_t^i)||^2 - ||f_t^i - f||_2^2 \\ &= \eta^2 ||\nabla\ell_t^i(f_t^i)||^2 - 2\eta\nabla\ell_t^i(f_t^i)(f_t^i - f). \end{aligned}$$

Using a telescoping sum of all terms over time, we get

$$\begin{aligned} \phi_T^i - \phi_0^i &= -2\eta \sum_{t=1}^{T} (f_t^i - f)\nabla\ell_t^i(f_t^i) + \eta^2 \sum_{t=1}^{T} ||\nabla\ell_t^i(f_t^i)||^2 \\ &\leq -2\eta \sum_{t=1}^{T} (f_t^i - f)\nabla\ell_t^i(f_t^i) + \eta^2 G_2 T. \end{aligned}$$

The last inequality holds due to the assumption that $||\nabla\ell_t^i(f_t^i)||^2 < G_2$. Thus, we can get

$$\begin{aligned} \mathcal{L}_{NORMA} &\leq \mathcal{L}_f + ||f_0^i - f||^2 - ||f_T^i - f||^2 + \eta^2 G_2 T \\ &\leq \mathcal{L}_f + \frac{||f_0^i - f||^2}{2\eta} + \frac{\eta G T}{2} \\ &\leq \mathcal{L}_f + \frac{\eta G T}{2} + \frac{C_2}{2\eta}, \end{aligned}$$

where $\mathcal{L}_{NORMA}$ is the total loss suffered by NORMA algorithm, and $\mathcal{L}_f$ is the cumulative loss suffered by any function f in $\mathcal{H}\kappa_i$. The last inequality holds due to the assumption that $||f_0^i - f||^2 \leq C_2$. Setting $\eta = \frac{C_2}{G_2 T}$, we get the result in the lemma. □

Theorem 3.4. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (Hedge) using the NORMA Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \frac{\ln(\frac{1}{\beta})}{1 - \beta} \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\ln(m)}{1 - \beta}, \tag{22}$$

and by choosing $\beta = \frac{\sqrt{T}}{\sqrt{T} + \sqrt{\ln m}}$, we get

$$\mathcal{L}_{OMKR} \leq \left(1 + \sqrt{\frac{\ln m}{T}} \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \ln m + \sqrt{T \ln m} \right),$$

where $\mathcal{D}$ is a sequence of instances, and

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^{T} \left(\frac{\lambda}{2} ||f||^2_{\mathcal{H}_\kappa} + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t \right) + \sqrt{C_2 G_2 T}$$

Proof. The proof follows from combining the results of Hedge and Lemma 1, in a similar way as done in Theorem 1. □

Theorem 3.5. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (OGD) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\eta G T}{2} + \frac{F}{2\eta}, \tag{23}$$

where G is constant that upper bounds the gradient of the loss function, and C is a constant that upper bounds the distance between any two weight vectors $\mathbf{w}$*, and*

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^{T} (f(x_t) - y_t)^2}{1 - \eta} + \frac{||f||^2}{\eta} \right]. \tag{24}$$

By choosing $\eta = \sqrt{\frac{F}{GT}}$, we get

$$\mathcal{L}_{OMKR} \leq \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \sqrt{FGT},$$

where $\mathcal{D}$ is a sequence of instances and

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^{T} \left(\frac{\lambda}{2} ||f||^2_{\mathcal{H}_\kappa} + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu \epsilon_t \right) + \sqrt{C_2 G_2 T}.$$

Proof. The proof follows by combining the result of Lemma 1 and OGD, in similar fashion as Theorem 2. □

3.4 Speed-Ups by Reducing Number of Support Vectors

A major short coming of the above approach is the quadratic time-complexity in running time of the algorithm, i.e., for a dataset with T instances, the running time is in $O(T^2)$. This is because of the curse of kernelization, where every new prediction requires a kernel function computation with all the older SVs in the dataset (and when squared loss is used, all instances invariably become SVs). This can often be infeasible for large datasets, which restricts the ability to use the above algorithms for online learning on large datasets. To speed this process up, we propose faster approximation schemes. Specifically we propose (i) budgeting strategies to limit the number of SVs; and (ii) functional approximation schemes to approximate the kernel functions without explicitly computing the kernel function with all the SVs.

Even though a non-zero loss is suffered often, particularly when the squared loss is used, many of these instances could potentially be noisy, or the loss suffered would be so small that the α coefficient assigned to the SV would be insignificant, which would lead to an insignificant impact on the prediction function. Moreover, as often observed in online learning applications, the data could exhibit concept drift [14], in which case many of the old SVs may actually harm the prediction function performance, in addition to adding to the computational cost. Further, not all kernels are good candidates for prediction, especially when their weights are low. In addition, not all the historical instances are good candidates for making the prediction, particularly in a non-stationary setting. With this motivation, we propose stochastic update and budget online kernel learning strategies.

3.4.1 Stochastic Update for OMKR. An update to a kernel regressor involves adding a new SV. If SVs are not added to less important kernels, the time taken for prediction by these kernels is significantly reduced. The intuition is if there is only one good kernel or a small subset of good performing kernels, it is only these should be given more data to learn the function, and the poor kernels are still allowed to make predictions (but with limited data), which takes much lesser computational time. We define a probability sampling denoted by q_t^i, which determines the probability of a kernel being selected for updates.

$$q_t^i = \frac{|w_t^i|}{\max_{1\le j\le m} |w_t^j|}. \tag{25}$$

This indicates that higher the absolute weight, the higher is the probability, and the best kernel has a probability of 1. When OMKR(Hedge) is used the weights can never be negative. In case of OGD updates in weights, there is a theoretical possibility for the weights to become negative, and hence we take absolute values to compute q_t^i, so as to account for weights having the maximum impact on the prediction. To prevent kernels with low weights, that do not have a significant impact to the prediction, from completely losing out, we introduce a smoothing parameter $\delta \in (0,1)$. The idea is to add a small component of uniform weights. The new probability of a kernel being selected for update is denoted by

$$p_t^i = (1-\delta)q_t^i + \frac{\delta}{m}. \tag{26}$$

Here, δ is a small value. A similar idea was used in [3], to tradeoff between exploration and exploitation. Using p we sample a subset of kernels based on Bernoulli Sampling, i.e., $m_t^i = Bernoulli(p_t^i)$. Only those kernels that are selected will be chosen for an update. The steps are described in Algorithm 3.

ALGORITHM 3: Stochastic OMKR scheme

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1,\ldots,m$
  - Update Parameter: $\beta \in (0,1)$ if Hedge or $\eta_w$ for OGD
  - Smoothing Parameter: $\delta \in (0,1)$
  - Step Size Parameter for each kernel: $\eta$
  - Regression parameters: $\lambda, \nu$ for OMKR(NORMA) NORMA)
Initialization: $f_1 = 0$, $\mathbf{w}_1 = \frac{1}{m}\mathbf{1}$
  for t = 1, ..., T do
    Receive instance: $x_t$
    Predict $\hat{y_t}$ based on Hedge or OGD combination
    Reveal true value $y_t$
    $q_t^i = \frac{|w_t^i|}{\max\limits_{1\le j\le m} |w_t^j|},\ i = 1,\ldots,m$
    $p_t^i = (1-\delta)q_t^i + \frac{\delta}{m},\ i = 1,\ldots,m$
    Sample $m_t^i = BernoulliSampling(q_t^i),\ i = 1,\ldots,m$
    for i = 1, ..., m do
      Set $\ell_t^i = \ell(f_t^i(x_t), y_t)$
      if $m_t^i == 1$ then
        Update $f_{t+1}^i$ = Equation (3) OR (6)
      end if
    end for
    Update $\mathbf{w}_{t+1}$ based on Hedge or OGD
  end for
```

3.4.2 *Budget OMKR.* As the number of SVs grows in an unbounded manner, it significantly increases the computational cost, particularly in the case of multiple kernels. There have been several approaches in literature to address the issue of setting a budget in the context of online learning with kernels (mostly for single kernel methods). A budget τ is specified, and the number of SVs is not allowed to exceed this number. These are broadly classified into three categories:

Removal, Projection, and Merging. Removal refers to replacing old SVs with new ones when the budget is exceeded based on a certain criteria. Projection refers to finding a common space to project new support vectors on when budegt is exceeded. Merging refers to merging a new support vector with an existing one when the budget is exceed.

Often, a subset of instances can explain the data as well as the entire data. Further, in a non-stationary time series setting, it is common to introduce a sliding window so as to give importance to only the most recent instances. In our case, we have a sliding window of the most recent SVs that explain the data. This is also particularly helpful in the case of NORMA, where in each iteration, the old SVs get reduced by a factor of $(1 - \eta\lambda)$ due to the regularization term. As t grows, the α values of the old SVs get reduced to almost zero. Such SVs can be ignored without any significant impact to the prediction. Therefore, we propose a parameter τ that restricts the total number of SVs that are allowed to be stored by each regressor. The older SVs are deleted.

4 LARGE-SCALE ONLINE MULTIPLE KERNEL REGRESSION VIA FUNCTIONAL APPROXIMATION

The proposed OMKR scheme follows the usage of traditional Online Learning with kernels [26], and independently learns each kernel regressor for a pool of m different kernel functions. These predictions are obtained in each online learning iteration and their weighted combination gives us the final prediction. During the update, first the weights of each kernel predictor updated by using the Hedge Algorithm [13], and each kernel predictor is also updated. This scheme suffers from the following two drawbacks:

- They suffer from the curse of kernelization, which means that the number of SVs is unbounded. This adds a significant computational burden for the algorithm. In particular poor performing kernels have a tendency to acquire a lot of SVs, thus taking up computational resources but not contributing to the final prediction. While the budgeting techniques may speed up the process, they can still be computationally expensive, as several kernel function computations may still be required to get reasonable performance. Moreover, budgeting techniques are heavily reliant on the selected SVs, which can be noisy or insufficient to make accurate predictions.
- The design of the original batch MKL is such that it aims to learn the optimal combination of kernel functions, in order to obtain a single unified kernel function. This means that the kernel combination is at a feature representation level, rather than at a prediction level. Unlike this, OMKR simplifies the learning process to two steps where the diverse kernels learn independently, and the combination of the kernels happens at a prediction level. Currently, there are no approaches in literature that perform Online Multiple Kernel Leading by combining kernels at the representation level.

To address these issues, we propose to apply functional approximation techniques to learn the kernel function [43, 61].

4.1 Functional Approximation for Kernels

The main idea is to construct a kernel induced feature representation $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^D$, where D is the new feature dimension, such that the dot product of instances in this new feature space is able to approximate the kernel function:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j). \tag{27}$$

Following this approximation, the single kernel prediction function takes the following form:

$$f_{t+1}(\mathbf{x}) = \Sigma_{i=1}^{t} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \Sigma_{i=1}^{t} \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}_\kappa^\top \mathbf{z}(\mathbf{x}), \tag{28}$$

where $\mathbf{w}_\kappa$ denotes the weight vector to be learnt in the new feature space that has been induced by the kernel κ.

Applying OGD [68], and performing online learning on this new feature space allows us to perform online learning with a single kernel. To extend this to the multiple kernel setting, we obtain m new feature representations induced by different kernels:

$$\mathbf{z}_{\kappa_i}(\mathbf{x}) \text{ for } i = 1, \ldots, m, \tag{29}$$

and correspondingly, we need to estimate m different weight vectors $\mathbf{w}_i$ for $i = 1, \ldots, m$, in order to learn the prediction function:

$$f^i(\mathbf{x}) = \mathbf{w}_{\kappa_i}^\top \mathbf{x}. \tag{30}$$

4.1.1 Fourier Approximation. First, we consider a class of kernels called shift-invariant kernels. A shift invariant kernel function is one whose result is a function of the distance between the two input instances, i.e.,

$$\begin{aligned} \kappa(\mathbf{x}_1, \mathbf{x}_2) &= k(\Delta\mathbf{x}) \quad \text{where} \\ \Delta\mathbf{x} &= \mathbf{x}_1 - \mathbf{x}_2. \end{aligned}$$

Popular examples of such kernels include Gaussian Kernels, Laplacian Kernels, Cauchy Kernels, and so on. For this class of kernels, random Fourier Features can be obtained to approximate the kernel function [32, 43]. Applying inverse Fourier transform to a shift-invariant kernel function, we get

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u}. \tag{31}$$

Here, $p(\mathbf{u})$ is the probability density function, which is obtained from the Fourier transform of $k(\Delta\mathbf{x})$. Consider the Gaussian Kernel function $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{||\mathbf{x}_1 - \mathbf{x}_2||_2^2}{2\sigma^2})$. The random Fourier component for this is $\mathbf{u}$ with the distribution $p(\mathbf{u}) = \mathcal{N}(0, \sigma^2 I)$. This kernel is continuous and positive definite, and applying Bochner's Theorem, we the kernel function can be expressed as an expectation of the random variable $\mathbf{u}$ [43, 45], such that

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

From the above equation, we can observe that the kernel function, can be represented as a dot product of the instances in the new representation, in expectation, where the new representation is

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

However, using only one Fourier component may lead to a large variance. To reduce the variance, we can sample more random Fourier components. Specifically, we sample D random Fourier components $\mathbf{u}_1, \ldots, \mathbf{u}_D$ and obtain the new feature representation as

$$\mathbf{z}(\mathbf{x}) = [\sin(\mathbf{u}_1^\top\mathbf{x}), \cos(\mathbf{u}_1^\top\mathbf{x}), \ldots, \sin(\mathbf{u}_D^\top\mathbf{x}), \cos(\mathbf{u}_D^\top\mathbf{x})]^\top. \tag{32}$$

4.1.2 Nyström Approximation. The random Fourier features have two limitations. First, they are designed for fixed kernel functions, and are not data dependent, which may cause loss of useful information that could be exploited. Second, they can be used for only shift-invariant kernels, and not any generalized kernel function. In order to address these issues, we can use the Nyström Method to perform Singular Value Decomposition (SVD) on the kernel matrix, to obtain the approximate features.

For a dataset with T instances, consider the full kernel matrix denoted by $\mathbf{K} \in \mathbb{R}^{T\times T}$, with rank r. Applying SVD to this matrix gives us $\mathbf{K} = \mathbf{VDV}^\top$, where the columns in $\mathbf{V}$ are orthogonal, and $\mathbf{D}$ is a diagonal matrix $\mathbf{D} = \mathrm{diag}(\sigma_1, \ldots, \sigma_r))$. For $k < r, \mathbf{K}_k = \Sigma_{i=1}^{k}\sigma_i V_i V_i^\top = \mathbf{V}_k\mathbf{D}_k\mathbf{V}_k^\top$ is the best rank-k approximation of $\mathbf{K}$. Given a large kernel matrix, $\mathbf{K} \in \mathbb{R}^{T\times T}$, Nyström method randomly samples a small subset of B columns, where $B \ll T$, and constructs a new matrix $\mathbf{C} \in \mathbb{R}^{T\times B}$, and using this derives a much smaller kernel matrix $\mathbf{W} \in \mathbb{R}^{B\times B}$. Thus, the large kernel matrix can be approximated as:

$$\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^{+}\mathbf{C}^\top \approx \mathbf{K}, \tag{33}$$

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

Using this Nystöm approach we can obtain the feature representation $\mathbf{z}(\mathbf{x})$ in the induced kernel space. Instead of considering all instances T as SVs, we consider a budget of B, where we are given a maximum of B SVs, where $B \ll T$. From Equation (33), we can see that the kernel value of two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ can be approximated as

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

Consequently, we can construct a new representation for instance $\mathbf{x}$ as

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

4.2 OMKR (Hedge) via Functional Approximation

Next, we briefly describe the OMKR learning strategies based on the new kernel induced feature representation. We discuss the corresponding algorithm for kernel combination using Hedge Algorithm [13]. Following the approach described in Section 3, the OMKR process can be split into the following two steps: (i) learning each kernel regressors; and (ii) learning the kernel combination.

Learning Online Kernel-based Regressors. In each online iteration, given an instance $\mathbf{x}$, first we obtain the new feature representation $\mathbf{z}(\mathbf{x})$. This can be obtained using either the Fourier Approximation strategy or the Nyström Approximation strategy described in the previous section. For the Nyström approach, Online Kernel Learning is performed in the usual manner, till B SVs are obtained. This is followed by using these B SVs and applying the Nyström method to obtain the representation $\mathbf{z}(\mathbf{x})$. Once the representation $\mathbf{z}(\mathbf{x})$ is obtained, the process of learning a single kernel regressor gets reduced to learning a linear model via online learning. Like before, we adopt learning via OGD [68].

Consider a squared loss function

$$\ell(f_t(\mathbf{x}_t), y_t) \quad = \quad (f_t(\mathbf{x}_t) - y_t)^2,$$

where

$$f(\mathbf{x}) \quad = \quad \mathbf{w}_\kappa^\top \mathbf{z}(\mathbf{x}).$$

Here, $\mathbf{w}_\kappa$ is the linear model to be learnt using the new kernel induced representation $\mathbf{z}(\mathbf{x})$. Following, OGD [68], we get the following update rule:

$$\mathbf{w}_{\kappa_t+1} \leftarrow \mathbf{w}_{\kappa_t} - \eta_t(\hat{y_t} - y_t)f_t(\mathbf{x}_t), \tag{34}$$

where η_t is a learning rate parameter, and $\hat{y_t} = \mathbf{w}_{\kappa_t}^\top f_t(\mathbf{x}_t)$. Using similar steps, we can derive the update rule for NORMA [26] based learning, using an ϵ-insensitive loss function.

Learning Online Kernel Combination. The next step is to learn the optimal kernel combination. The final prediction in each online learning iteration using m different diverse kernel functions is given by

$$F_t(\mathbf{x}) = \sum_{i=1}^{m} w_t^i f_t^i(\mathbf{x}),$$

where $f_t^i(\mathbf{x})$ represents the output of the kernel function, where the representation $\mathbf{z}_i(\mathbf{x})$ is induced by kernel i. Applying Hedge [13], and following the update rule as described in Section 3.2.2, we get

$$w_{t+1}^i = w_t^i \beta^{\ell_t^i}, i = 1, \ldots, m.$$

This entire scheme is outlined in Algorithm 5.

4.3 OMKR (OGD) via Functional Approximation

For learning OMKR using functional approximation with OGD combination of kernels, the learned model becomes fundamentally different to all the OMKR approaches described above. The above approaches split the MKL procedure into two steps: learning the kernel regressors indecently, followed by combining the multiple predictions using Hedging. However, following the principle of the original batch MKL optimization:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_K(\theta)} \frac{1}{2} |f|^2_{\mathcal{H}_{K(\theta)}} + C \sum_{t=1}^{T} \ell(f(\mathbf{x}_t), y_t). \tag{35}$$

Here, we can see that the original intention is to learn the optimal kernel function, as a combination of multiple kernels. This means that the combination of multiple kernels is to be at the representation level, and not the prediction level. While the above approaches offer a resembling effect to exploit the representation powers of different kernels, they do not combine the kernels at a representation level to learn the ideal kernel function.

We propose OMKR (OGD) via a Functional Approximation, that enables learning the combination of multiple kernels at a representation level. First we obtain the new feature representation

ALGORITHM 4: OMKR(Hedge) via Functional Approximation

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1, \ldots, m$
  - Update Parameter: $\beta \in (0,1)$ if Hedge
  - Step Size Parameter for each kernel: $\eta$
Initialization: $f_1 = 0, \mathbf{w}_1 = \frac{1}{m}\mathbf{1}$
  for t = 1, . . . , T do
    Receive instance: $x_t$
    Obtain new feature representation $\mathbf{z}_i(\mathbf{x}_t)$ using Fourier or Nyström approach $\forall i = 1, \ldots, m$
    Predict $\hat{y_t} = \sum_{i=1}^{m} w_t^i f_t^i(x_t) = \sum_{i=1}^{m} w_t^i(\mathbf{w}_{\kappa_i t}^\top \mathbf{z}_i(\mathbf{x}_t))$
    Reveal true value $y_t$
    for i = 1, . . . , m do
      Set $\ell_t^i = \ell(f_t^i(x_t), y_t)$
      Update $\mathbf{w}_{\kappa_i t+1} \leftarrow \mathbf{w}_{\kappa_i t} - \eta_t(\hat{y_t} - y_t) f_t(\mathbf{x}_t)$
      Update $w_{t+1}^i = w_t^i \beta^{\ell_t^{*i}}$ where $\ell_t^{*i} = (f_t^i(x_t) - y_t)^2$
    end for
  end for
```

ALGORITHM 5: OMKR(OGD) via Functional Approximation

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1, \ldots, m$
  - Update Parameter: $\eta$ for OGD
Initialization: $f_1 = 0$, $\mathbf{w}_1 = \frac{1}{m}\mathbf{1}$
  for t = 1, ..., T do
    Receive instance: $x_t$
    Obtain new feature representation $\mathbf{Z}(\mathbf{x}) = [\mathbf{z}_1(\mathbf{x}), \ldots, \mathbf{z}_m(\mathbf{x})]^\top$ using Fourier or Nyström approach
    Predict $\hat{y_t} = F_t(\mathbf{x}_t) = \mathbf{w}_t^\top \mathbf{Z}(\mathbf{x}_t)$
    Reveal true value $y_t$
    Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta(\hat{y_t} - y_t)F_t(\mathbf{x}_t)$
  end for
```

for each instance $\mathbf{x}$ by applying the Fourier or the Nyström approximation. Then, we concatenate all these feature representations from each kernel as

$$\mathbf{Z}(\mathbf{x}) = [\mathbf{z}_1(\mathbf{x}), \ldots, \mathbf{z}_m(\mathbf{x})]^\top. \tag{36}$$

$\mathbf{Z}(\mathbf{x})$ represents the approximated feature representation induced by the kernel function, which is a combination of multiple kernels. The aim now is to learn appropriate weights for each of the features in this new representation. We do so by applying OGD in the new feature representation to learn the weight vector $\mathbf{w}$, which has the same dimensionality as $\mathbf{Z}(\mathbf{x})$.

The multiple kernel prediction function is given by

$$F(\mathbf{x}) = \mathbf{w}^\top \mathbf{Z}(\mathbf{x}). \tag{37}$$

Using the squared loss function $\ell(F_t(\mathbf{x}_t), y_t) = (F_t(\mathbf{x}_t) - y_t)^2$, by applying OGD[68], we get the update rule as

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t(\hat{y_t} - y_t)F_t(\mathbf{x}_t). \tag{38}$$

5 APPLICATION TO TIME-SERIES PREDICTION

OMKR can be applied a variety of online regression tasks, especially for mining data streams. A natural application of OMKR is Time Series Prediction, which is the task of predicting the future value based on given past values. Kernel methods have been commonly used for solving such problems [49, 57]. For our problem setting, the aim is to perform online learning for time series prediction [1]. We first introduce the popular time series prediction models: AR models and ARMA models. Then, we present how to apply the OMKR framework to online learn for time-series prediction. This is followed by discussing the capturing of non-linearity for time-series prediction through kernelized models.

5.1 Time Series Models

AR model is used for a univariate time series where the value of the series at a particular time is linearly dependent on its own previous values. An $AR(p)$ model denotes an AR process of order p, i.e., y_t is described by a noisy linear combination of $[y_{t-1}y_{t-2}\ldots y_{t-p}]$:

$$y_t = c + \Sigma_{i=1}^{p}\zeta_i y_{t-i} + \epsilon_t, \tag{39}$$

where c is a constant, ϵ_t is white noise, and ζ_i. are the parameters describing the dependency. We denote by Y_{t-1}^{p} the set of p past values, i.e., $Y_{t-1}^{p} = \{y_{t-1}, y_{t-2}, \ldots, y_{t-p}\}$, and thus the equation simplifies to

$$y_t = c + \zeta^p Y_{t-1}^p + \epsilon_t, \tag{40}$$

where ζ^p and Y_{t-1}^p are both p-dimensional vectors.

Using such a model for online learning faces two main challenges as follows:

— In the real world setting, it is non-trivial to determine the order p, as it is hard to determine how many past variables the target would be dependent on.

— If we arbitrarily chose a very large p and expected the model to automatically learn 0 coefficients for very old values which the model deems to be irrelevant, then the learning procedure could converge slowly, and give noisy results.

To address this issue, we consider a pool of k different values p. For example, $p \in \{p_1, p_2, \dots p_k\}$. Using these we construct a pool of kernels before applying OMKR. This pool of kernels is given as

$$\mathcal{K} = \left\{\kappa(Y_t^{p_1}, \cdot), \dots, \kappa(Y_t^{p_k}, \cdot)\right\}. \tag{41}$$

Since we consider linear time series modeling, we consider only linear kernels, i.e., $\kappa(Y_1^p, Y_2^p) = (Y_1^{p\top} Y_2^p)$. These are effectively k different kernel functions for the learning task, where each kernel function corresponds to a different order of the AR time-series process. This can be directly plugged into the OMKR framework, to perform Online Learning for Time-Series Prediction, which also allows us to obtain sub-linear regret with respect to the best performing order p. Apart from the obvious benefit of not having to manually select the order p, another advantage is that if the order of the process p is very high, it can first be approximated by a low-order p in the initial few iterations, leading to faster convergence, followed by slowly adapting to the higher order via hedging, which could give an improved performance. Thus, this procedure can exploit the faster convergence due to fewer parameters in the initial stages of online learning, and at the same time, it enjoys improved performance in the long run, in case higher order processes describe the time-series better.

We now discuss the extension of this to ARMA time-series modeling. ARMA model is more sophisticated, and involves a term for the moving average (MA) of the time series. The MA model is similar to AR model, except that the linear dependence is not on the past values, but the past errors. An MA(q) model is given by $y_t = \mu + \Sigma_{i=1}^q \xi_i \epsilon_{t-j} + \epsilon_t$. Combining $AR(p)$ and $MA(q)$ gives us an $ARMA(p,q)$ process is given by

$$y_t = c + \Sigma_{i=1}^p \zeta_i y_{t-i} + \Sigma_{i=1}^q \xi_i \epsilon_{t-j} + \epsilon_t. \tag{42}$$

Since the error terms are not directly observable for the MA component, making it very difficult to estimate the model parameters in the online setting. To alleviate this issue, [1] showed that an $ARMA(p,q)$ model could be learned online by learning an $AR(m+p)$ model, where m was set as $m = q \cdot \log_{1-\epsilon}((TLM_{\max})^{-1})$. m controls the level of approximation. Under certain assumptions (discussed in [1]), Online Learning of an $AR(m+p)$ model could achieve sub-linear regret compared to the best $ARMA(p,q)$ model. Thus, applying OMKR, to AR models can also obtain sub-linear regret with respect to the best $ARMA(p,q)$ model.

This approach can further be extended to ARIMA time-series modeling. While ARMA is designed for stationary settings, ARIMA is used for modeling non-stationary series. ARIMA does so by finding patterns in the differentials of the time series. Consider for example the first-order differential $\nabla y_t = y_t - y_{t-1}$, and similarly, the second-order differential as $\nabla^2 y_t = \nabla y_t - \nabla y_{t-1}$. If the sequence of the differentials $\nabla^d y_t$ satisfies an $ARMA(p,q)$ model, then the sequence y_t satisfies and $ARIMA(p, d, q)$ model. Thus, an $ARIMA(p, d, q)$ model takes the following form:

$$\nabla^d y_t = c + \Sigma_{i=1}^{p} \zeta_i \nabla^d y_{t-i} + \Sigma_{i=1}^{q} \xi_i \epsilon_{t-j} + \epsilon_t. \tag{43}$$

Following [30], the above model can also be learnt online using only the $AR(p)$ process, except the time-series here is the differentials. Like before, the optimal value of p can automatically determined by learning this time-series online through the OMKR framework.

5.2 Online Non-Linear Time Series Prediction

An assumption made by time-series models is that the dependence on the historical signals is linear. This assumption may not hold true, and motivates the need for having non-linear time series modeling. Consider the AR model described in Section 5, where

$$y_t = c + \zeta^p Y_{t-1}^p + \epsilon_t. \tag{44}$$

The above model assumes linear dependency on the previous p values. We use kernels to explore non-linear dependencies. The kernelized $AR(p)$ model is given by

$$y_t = c + f(y_{t-1}, y_{t-2}, \dots, y_{t-p}) + \epsilon_t = c + f(Y_{t-1}^p) + \epsilon_t,$$

where $f(Y_{t-1}^p) \in \mathcal{H}_\kappa$ is the prediction of the regression function using a kernel κ.

Next, we propose to construct a pool of multiple kernels for varying values of parameter $p \in [p_1, p_2, \dots, p_k]$, and m kinds of diverse kernels for each p. This gives us the following pool of mk kernel functions:

$$\mathcal{K} = \left\{\kappa^i(Y_t^{p_1}, \cdot), \dots, \kappa^i(Y_t^{p_k}, \cdot) \text{ for } i = 1, \dots, m.\right\}. \tag{45}$$

The above can now be directly plugged into the OMKR framework for solving time series prediction tasks. In comparison to existing kernel methods for times series prediction, the proposed OMKR solution enjoys the important advantages of avoiding tedious kernel selection and parameter selection and exploiting the power of combining multiple kernels for more accurate prediction.

6 EXPERIMENTS

6.1 Evaluation of OMKR on Stationary Datasets and Non-Linear Time-Series

6.1.1 *Datasets.*

We use five regular regression datasets and seven time series datasets. The data is from different applications, with a wide range of data size and dimensionality. All data attributes including the target were scaled to $[0, 1]$. The algorithms were run on 10 random permutations of the regular regression datasets to establish robustness. Such permutations are not applicable in the case of time series. The details of the datasets used can be seen in Table 1.

Datasets D1, and D2 were taken from the UCI repository[1], D3-D4 from StatLib,[2] D5 is a synthetic dataset obtained from Delve.[3] D6–D10 are datasets from the Santa Fe Time Series Competition Data[4]. D6 is stationary, D7 is non-stationary, and unlike other time series data, is not univariate, but is dependent on 2 attributes, D8 and D9's stationarity property is unknown, and D10 is characterized by noise. For univariate time series data the attribute column having 20|10 indicates the choice of 2 kernelized $AR(p)$ process with $p = 10, 20$ each having its own m kernel functions.

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

[2] http://lib.stat.cmu.edu/.

[3] http://www.cs.toronto.edu/~delve/data/datasets.html.

[4] http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html.

Table 1. List of Datasets

ID	Name	# Instances	# Attributes
Regression datasets			
D1	Abalone	4177	8
D2	Parkinsons	5875	20
D3	Spacega	3107	6
D4	Cadata	20640	8
D5	Add10	9792	11
Time series datasets			
D6	Laser	10073	20|10
D7	Physiological	17000	2
D8	Currency Exch. 1	3000	20|10
D9	Currency Exch. 2	3000	20|10
D10	Astrophysical	598	20|10

6.1.2 Kernels. We evaluate the performance of OMKR by using a pool of 24 predefined kernels. These include four polynomial kernels $\kappa(x, y) = (x^T y)^p$ of degree parameter $p = 1, 2, 3, 4$, 13 RBF kernels ($\kappa(x, y) = e^{(\frac{-||x-y||^2}{2\sigma^2})}$) of kernel width parameter σ in $[2^{-6}, 2^{-5}, \ldots, 2^6]$, 5 Cauchy kernels ($\kappa(x, y) = \frac{1}{1+\frac{||x-y||^2}{\sigma^2}}$) with parameter σ in $[2^{-2}, 2^{-1}, \ldots, 2^2]$, one sigmoid kernel($\kappa(x, y) = \tanh(xy)$) and a Chi-Square Kernel ($\kappa(x, y) = 1 - \Sigma_{i=1}^{n} \frac{(x_i-y_i)^2}{\frac{1}{2}(x_i+y_i)}$). Since all our data is scaled to $[0, 1]$, we clip the kernel prediction to this range, i.e., $\hat{y_t} = \max(0, \min(1, \hat{y_t}))$.

6.1.3 Baselines and Experimental Setting. We conduct two sets of experiments. First, in which we compare the proposed OMKR scheme against several baselines, and in the second we evaluate whether the OMKR scheme can generalize its good performance to different learning strategies for the individual kernel. For the first set of experiments we consider the following baselines: (i) Passive Aggressive Online Learning [9] for linear online regression; (ii) Kernel Least Mean Square Algorithm (KNLMS) [31]; (iii) Quantized Kernel Least Mean Squared Algorithm (Q-NLMS) [8]. For both of these, we evaluate the performance of the algorithms across a variety of kernel functions, and report the performance of the best (1st) and the second best (2nd) performing kernel function. Note that these best and second best kernel functions can only be determined in hindsight; (iv) L2-space MKL [40] for different sets of 3 Gaussian Kernels; (v) *Regression(V)*: Best Kernel by validation, where the best kernel function is determined on the basis of best performance on few of the initial instances in the data stream; (vi) *Regression(H)*: Best Kernel in hindsight; (vii) *Uniform OMKR*: Uniform weight distribution over kernels (to see if this can eliminate the impact of a poor kernel choice); (viii) *Deterministic OMKR (Hedge)*; and (ix) *Deterministic OMKR (OGD)*. In the first set of experiments, OMKR based on WH regression is considered.

For the next set of experiments we evaluate both OMKR(WH) and OMKR(Norma) algorithms, and compare the following: Reg(V), Reg(H), Uniform, OMKR(Hedge), and OMKR(OGD). We then analyze the performance of efficiency enhancing variants of OMKR and study the tradeoff between accuracy and efficiency. For the large datasets (D11 and D12), we compare only the budget versions of all algorithms.

All parameters for the regression tasks (if any), and the best kernel for Regression(V) were chosen by online validation technique. We performed a grid search and evaluated the performance

Table 2. Performance of OMKR as Compared to Baselines for Kernel-Based Online Regression Tasks

	D1		D2		D3		D4		D5	
Algorithms	Kernel	MSE	Kernel	MSE	Kernel	MSE	Kernel	MSE	Kernel	MSE
PA	Linear	0.0150	Linear	0.0589	Linear	0.0285	Linear	0.0358	Linear	0.0096
K-NLMS(1st)	RBF(0.5)	0.0075	RBF(0.5)	0.0239	RBF(4)	0.0030	RBF(0.5)	0.0207	RBF(1)	0.0064
K-NLMS(2nd)	RBF(1)	0.0075	RBF(1)	0.0326	RBF(2)	0.0031	RBF(0.25)	0.0224	RBF(2)	0.0070
Q-NLMS(1st)	RBF(0.25)	0.0091	RBF(0.25)	0.0460	RBF(0.25)	0.0032	RBF(0.25)	0.0288	RBF(0.25)	0.0074
Q-NLMS(2nd)	RBF(0.5)	0.0093	RBF(0.5)	0.0495	RBF(5)	0.0033	RBF(0.5)	0.0306	RBF(0.5)	0.0081
L2-S-MKL	RBF(0.5,1,2)	0.0080	RBF(0.5,1,2)	0.0277	RBF(0.5,1,2)	0.0048	RBF(0.5,1,2)	0.0233	RBF(0.5,1,2)	0.0071
L2-S-MKL	RBF(1,2,4)	0.0083	RBF(0.5,1,4)	0.0295	RBF(0.25,1,4)	0.0094	RBF(0.5,1,4)	0.0233	RBF(0.25,1,4)	0.0061
Reg(V)	Linear	0.0085	Linear	0.0512	RBF(32)	0.0041	RBF(32)	0.0594	Linear	0.0096
Reg(H)	Cauchy(0.25)	**0.0072**	Cauchy(0.25)	**0.0199**	Cauchy(1)	**0.0028**	Cauchy(0.25)	**0.0201**	RBF(1)	**0.0051**
OMKR(Uniform)	All	0.0092	All	0.0402	All	0.0160	All	0.0286	All	0.0273
OMKR(Hedge)	All	**0.0073**	All	**0.0201**	All	**0.0029**	All	**0.0202**	All	**0.0053**
OMKR(OGD)	All	0.0082	All	0.0230	All	0.0035	All	0.0216	All	0.0060

Algorithms	D6		D7		D8		D9		D10	
	Kernel	MSE	Kernel	MSE	Kernel	MSE	Kernel	MSE	Kernel	MSE
PA	Linear	0.0167	Linear	0.0117	Linear	0.0095	Linear	0.0056	Linear	0.0206
K-NLMS(1st)	RBF(1)	0.0030	RBF(4)	0.0026	RBF(4)	0.0004	RBF(4)	0.0013	RBF(4)	0.0085
K-NLMS(2nd)	RBF(0.5)	0.0033	RBF(2)	0.0028	RBF(2)	0.0006	RBF(2)	0.0017	RBF(2)	0.0122
Q-NLMS(1st)	RBF(0.25)	0.0062	RBF(4)	0.0028	RBF(0.25)	0.0007	RBF(0.25)	0.0023	RBF(0.25)	0.0106
Q-NLMS(2nd)	RBF(0.5)	0.0067	RBF(2)	0.0028	RBF(0.5)	0.0008	RBF(0.5)	0.0026	RBF(0.5)	0.0122
L2-S-MKL	RBF(0.5,1,2)	0.0027	RBF(0.5,1,2)	0.0035	RBF(0.5,1,2)	0.0029	RBF(0.5,1,2)	0.0048	RBF(0.5,1,2)	0.0213
L2-S-MKL	RBF(0.5,1,4)	0.0028	RBF(0.25,1,4)	0.0039	RBF(0.25,1,4)	0.0054	RBF(0.25,1,4)	0.0083	RBF(0.25,1,4)	0.0841
Reg(V)	Poly(4)	0.0161	RBF(32)	0.0027	Poly(2)	**0.0002**	Poly(2)	0.0738	Poly(2)	0.0088
Reg(H)	Cauchy(0.25)	0.0024	RBF(4)	0.0026	Linear	**0.0002**	Linear	**0.0004**	Linear	0.0087
OMKR(Uniform)	All	0.0082	All	0.0044	All	0.0035	All	0.0075	All	0.0277
OMKR(Hedge)	All	**0.0023**	All	0.0025	All	**0.0002**	All	0.0009	All	0.0075
OMKR(OGD)	All	0.0024	All	**0.0010**	All	0.0003	All	**0.0004**	All	**0.0069**

The numbers in bold denote the best performance.

of the parameters on the of first 100 instances or first 10% of the instances, whichever was lesser. The value of Hedge parameter β was fixed to 0.5 in all cases, and the learning rate η_w was fixed to 0.025 for OGD update of weights). We also conducted sensitivity analysis for the weight update parameters. The learning rate η for each kernel regression was fixed at 0.1. Since η is the same for both single kernel and multi-kernel versions, its choice does not affect the comparison between Single Kernel Regression and OMKR. For budget strategies, we fixed the budget size $\tau = 500$ SVs. In stochastic OMKR, the smoothing parameter δ was set to 0.05 in all cases. The baselines were implemented using a learning rate of 0.1, and default parameter settings of the toolbox in [58].

6.1.4 *Results and Discussion.*

Table 2 shows the result of OMKR algorithms in comparison to the baselines. The Kernel column identifies the kernel function whose performance has been reported. There are several critical observations that are made here. In general, we see there is a clear advantage of using kernel-based regression over linear regression approaches. Second, we observe that a variety of different kernels get selected for different datasets and algorithms. This emphasizes the problem of kernel selection in the online setting. In contrast, our proposed approach (OMKR) is able to in most cases match the performance of the best kernel function (in hindsight), and often even beat the this kernel.

The detailed results of single kernel regression against OMKR can be seen in Table 3. Columns Reg(V) and Reg(H) represent single kernel regression by validation and in hindsight. Columns

Table 3. Single Kernel Regression vs. Multiple Kernel Regression

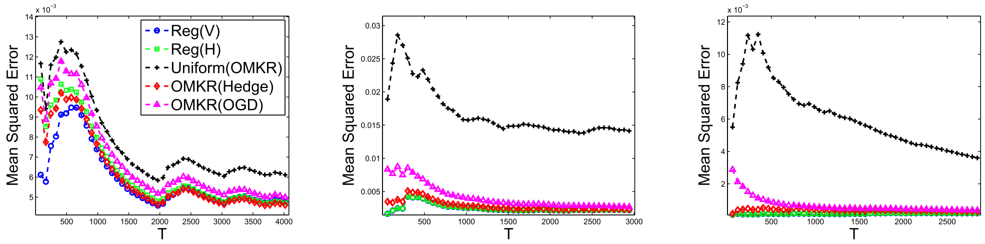
	Widrow Hoff					Norma				
ID	Reg(V)	Reg(H)	Uniform	Hedge	OGD	Reg(V)	Reg(H)	Uniform	Hedge	OGD
Regression Datasets										
D1	1.00	**0.85**	1.06	0.89	0.98	1.00	0.36	0.29	0.31	**0.26**
D2	1.00	**0.39**	0.79	**0.39**	0.45	1.00	0.40	0.49	0.40	**0.39**
D3	1.00	**0.69**	3.90	**0.69**	0.82	1.00	0.87	1.55	**0.52**	0.60
D4	1.00	**0.79**	1.13	**0.79**	0.85	1.00	0.77	0.93	0.74	**0.67**
D5	1.00	**0.53**	2.85	0.56	0.62	1.00	**0.21**	0.53	**0.21**	0.22
Time Series Datasets										
D6	1.00	**0.13**	0.50	**0.14**	**0.14**	1.00	0.96	1.68	0.70	**0.57**
D7	1.00	0.96	1.61	0.93	**0.37**	1.00	0.98	0.69	0.23	**0.15**
D8	1.00	**0.96**	12.40	1.65	1.67	1.00	0.73	0.30	**0.15**	0.18
D9	1.00	**0.01**	0.07	**0.01**	**0.01**	1.00	0.79	0.65	**0.18**	**0.17**
D10	1.00	0.83	2.90	0.84	**0.66**	1.00	0.67	1.60	**0.45**	0.54

Note: Each Field is the ratio $\frac{MSE_{\text{algorithm}}}{MSE_{\text{Reg}(V)}}$. Lower ratio implies lower MSE. Best ratios are in bold. The results for the regression datasets are averaged over 10 different permutations. The standard deviation is significantly lower in OMKR versions.

Uniform, Hedge, OGD represent OMKR with uniform weights, weight updated by Hedge, and weight updated by OGD, respectively.

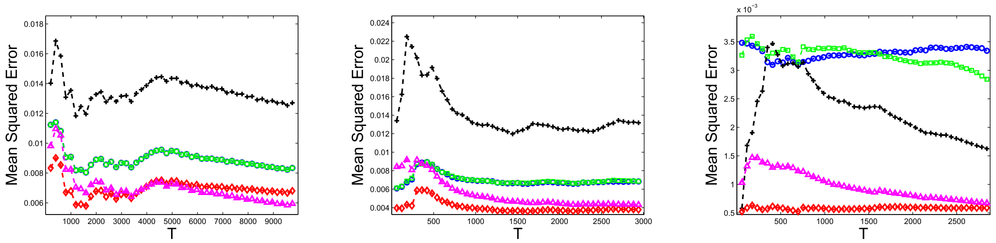
With almost no exception both our proposed methods OMKR (Hedge and OGD) outperform Reg(V) very significantly, at times achieving as low as 1% of error of Reg(V). We should note that in a real world setting, it is hard to choose a better kernel for unseen data than by a validation method. Reg(H) is the best kernel in hindsight, and is not known prior to running the experiments. Despite this, OMKR algorithms significantly outperform Reg(H) in most cases. In cases, where it OMKR does not beat Reg(H), their performance is very closely matched. Thus, without any *a priori* knowledge, OMKR is able to outperform even the best kernel in hindsight. This is because OMKR is able to identify a linear combination of kernels, which provide complementary information to each other in order to give a weighted prediction which beats any single best kernel. Uniform OMKR is affected by the usage of certain poor kernels and its performance is very inconsistent across datasets. It never beats OMKR(OGD), and beats OMKR(Hedge) in only one case (D1-Norma). This however is probably an exception, in which the optimal linear combination is close to a uniform distribution, because of which uniform weights are probably just a lucky guess. The difference in performance by Reg(V) and Reg(H), and the poor performance by Uniform(OMKR) highlight the difficulty of choosing the best kernel function for a given task. In terms of efficiency, deterministic OMKR takes roughly *m* times the amount of time take by single kernel regression.

Hedge and OGD are suitable in different scenarios. Due to a multiplicative update, Hedge converges very quickly, by identifying the single kernel that best represents the data, which is often the case. However, since Hedge only offers a linear combination of the best kernel(s), we expect the optimal linear combination determined by OGD to outperform Hedge. This does not happen if the data is not large enough for OGD to converge to optimal linear combination, or the data is non-stationary such that the appropriate kernel function changes too frequently for OGD to be able to learn the optimal combination. We plot the cumulative mean squared error against time for some representative datasets in Figures 1 and 2. It can be seen, that in most cases, OMKR(Hedge) attains a very low MSE from the beginning and does not improve much further, whereas, OMKR(OGD) starts with a relatively higher MSE, but it is continuously improving its



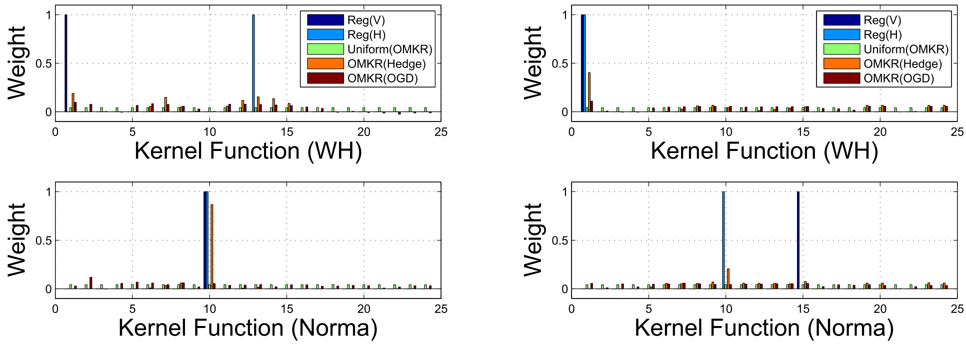
(a) D1 (b) D3 (c) D8

Fig. 1. Cumulative mean squared error with time (when Widrow-Hoff is used for regression): All results are displayed for data after the validation stage during which the parameters were determined.



(a) D1 (b) D3 (c) D8

Fig. 2. Cumulative mean squared error with time (when norma is used for regression): All results are displayed for data after the validation stage during which the parameters were determined.



(a) D1 (b) D3

Fig. 3. Weight distribution attained by all algorithms.

performance. Referring back to Table 3, it can be seen that in general that OMKR(OGD) has relative advantage in larger datasets, and OMKR(Hedge) in smaller ones. Additionally, we also look at the weight distribution attained by the algorithms, which is shown in Figure 3. The weight distribution by OMKR(Hedge) concentrates largely on the best kernel in hindsight, and otherwise has weights over certain reasonably good performing kernels. Unlike OMKR(Hedge), OMKR(OGD) does not have a concentrated distribution of weights over few kernels.

6.1.5 Evaluation of Efficiency Enhancers. The MSE and the time taken by Deterministic, Stochastic, and Budget OMKR are detailed in Tables 4 and 5. Clearly the time taken by both stochastic

Table 4. OMKR (Hedge) vs. Stochastic and Budget Strategies

	Widrow Hoff						Norma					
ID	Determinisitc		Stochastic		Budget		Det OMKR		Stochastic		Budget	
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	TIME
Regression datasets												
D1	**0.0075**	348	0.0079	39	0.0096	74	0.0121	220	**0.0113**	45	0.0122	68
D2	**0.0209**	707	0.0488	32	0.0436	109	**0.0451**	537	0.0544	45	0.0467	102
D3	**0.0028**	193	0.0035	22	0.0058	54	**0.0049**	159	0.0050	37	**0.0049**	53
D4	**0.0245**	8496	**0.0243**	376	0.0354	385	0.0477	6397	**0.0416**	354	**0.0413**	388
D5	**0.0054**	1950	0.0096	64	0.0122	183	**0.0108**	1338	0.0151	112	0.0116	171
Time series datasets												
D6	**0.0022**	4062	0.0034	146	0.0066	427	0.0058	2611	**0.0034**	182	0.0059	413
D7	**0.0025**	5728	0.0030	831	0.0088	312	**0.0008**	5101	0.0017	1118	**0.0008**	322
D8	0.0003	346	**0.0002**	15	0.0007	117	0.0005	274	**0.0002**	136	0.0005	111
D9	**0.0009**	352	0.0010	56	0.0011	118	**0.0006**	280	0.0010	119	**0.0006**	114
D10	**0.0074**	16	0.0086	3	0.0089	15	**0.0047**	12	0.0086	6	**0.0047**	12

Note: Here again, the results of regression datasets are averaged over 10 random permutations. All times are in seconds. The numbers in bold denote the best performance.

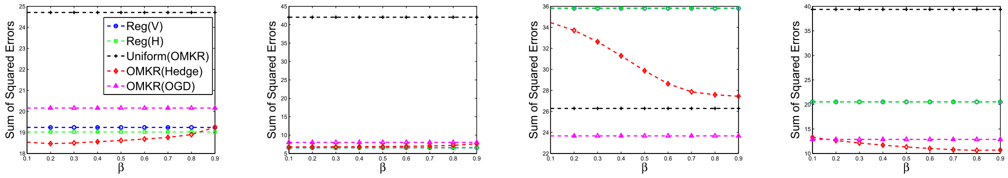
Table 5. OMKR (OGD) vs. Stochastic and Budget Strategies

	Widrow Hoff						Norma					
ID	Determinisitc		Stochastic		Budget		Det OMKR		Stochastic		Budget	
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	TIME
Regression datasets												
D1	**0.0082**	348	0.0086	43	0.0097	76	0.0105	220	**0.0095**	48	0.0105	69
D2	**0.0230**	707	0.0488	35	0.0432	111	**0.0442**	537	0.0521	49	0.0466	105
D3	**0.0034**	193	0.0045	24	0.0043	55	0.0055	159	**0.0044**	40	0.0056	54
D4	**0.0261**	8496	**0.0260**	414	0.0311	393	**0.0006**	6397	0.0322	382	0.0363	396
D5	**0.0060**	1950	0.0109	70	0.0108	187	**0.0115**	1338	**0.0115**	121	0.0121	174
Time series datasets												
D6	**0.0021**	4062	0.0039	160	0.0055	427	**0.0048**	2611	**0.0048**	200	**0.0048**	413
D7	**0.0010**	5728	0.0011	914	0.0023	318	**0.0005**	5101	0.0008	1230	0.0006	328
D8	0.0003	346	**0.0002**	17	0.0004	117	0.0006	274	**0.0002**	149	0.0006	111
D9	**0.0004**	352	0.0005	62	0.0005	118	0.0006	280	**0.0004**	130	0.0006	114
D10	**0.0058**	16	0.0113	3	0.0066	15	**0.0056**	12	0.0063	6	**0.0056**	12

Note: Here again, the results of regression datasets are averaged over 10 random permutations. All times are in seconds. The numbers in bold denote the best performance.

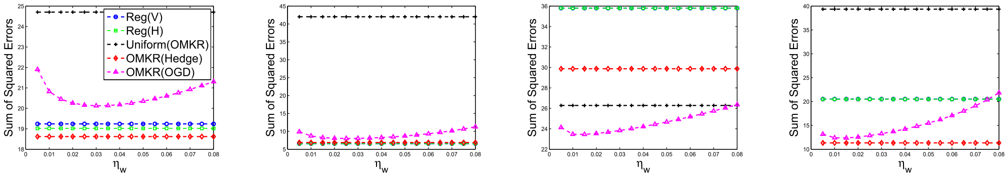
and budget techniques is significantly lower than Deterministic OMKR. Despite this, in most cases, the efficiency enhancers give comparable MSEs with respect to Deterministic OMKR. In many cases, particularly time series, the variants are able to outperform the deterministic version. This shows their ability to retain important information from the data, and adapt to changes in the pattern. Stochastic is faster than budget in smaller datasets, but in larger datasets, the number of SVs in stochastic start dominating even if only for a few kernels, and hence Budget is faster.

6.1.6 Sensitivity to Weight Update Parameters β and η_w. The results are shown in Figures 4 and 5. OMKR(Hedge) is not very sensitive to the value of the discount rate parameter β. There is a reasonably large range of values of β in which OMKR(Hedge)'s relative performance to other



(a) D1 (WH) (b) D3 (WH) (c) D1 (Norma) (d) D3 (Norma)

Fig. 4. Sensitivity of OMKR(Hedge) to discount rate parameterβ: We vary β keeping the performance of all other algorithms fixed.



(a) D1 (WH) (b) D3 (WH) (c) D1 (Norma) (d) D3 (Norma)

Fig. 5. Sensitivity of OMKR(OGD) to learning rate η_w: We vary η_w keeping the performance of all other algorithms fixed.

Table 6. List of Datasets for Evaluation of Large-Scale OMKR

ID	Name	#Instances	#Attributes
L1	Facebook comments	40,949	53
L2	Blog	52,397	280
L3	Year MSD	515,345	90
L4	Twitter	583,250	77

algorithms remains the same. OMKR(OGD)'s sensitivity to the learning rate η_w shows a tradeoff between large and small learning rates. This behavior is typical of all gradient descent algorithms.

6.2 Evaluation of Large-Scale OMKR Using Functional Approximation

In this section, we evaluate the efficiency and efficacy of the proposed Large Scale OMKR Methods. Apart from scalability concerns, we also look at the comparison between combination of kernels at the prediction level vs. the feature representation level (something that was non-trivial to achieve in previous approaches).

6.2.1 *Datasets.* For these experiments we consider larger datasets, for which it would be impractical to use deterministic OMKR strategies. This is because the runtime complexity of Determinisitc stragies is in $O(mT^2)$ for T instances with m kernel functions, and this time cost can be very large. We consider four datasets, whose details can be seen in Table 6. All of them were taken from the UCI repository.[5] Like before, the datasets were preprocessed with the features and the target scaled to lie $[0, 1]$. L1 is about predicting the volume of comments on Facebook. L2 is a similar task, where the total number of comments on a blog in the next 24 hours is to be predicted. L3 is about predicting the year to which the sound belongs based on audio features, and L4 is about predicting the buzz on Twitter.

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

Table 7. Large-Scale Online Multiple Kernel Regression

Algorithm	L1	Time (s)	L2	Time (s)
OMKR (Budget)	7.300e−04±0.0e+00	533	5.258e−04±0.0e+00	1051
FOMKR (Hedge)	**5.748e−04±3.0e−06**	6.5	**4.793e−04±9.8e−06**	8
FOMKR (OGD)	**5.742e−04±6.2e−06**	6.5	5.037e−04±2.2e−06	8
NOMKR (Hedge)	6.389e−04±0.0e+00	24	5.310e−04±5.4e−19	401
NOMKR (OGD)	6.059e−04±4.6e−19	24	**4.864e−04±7.7e−19**	401

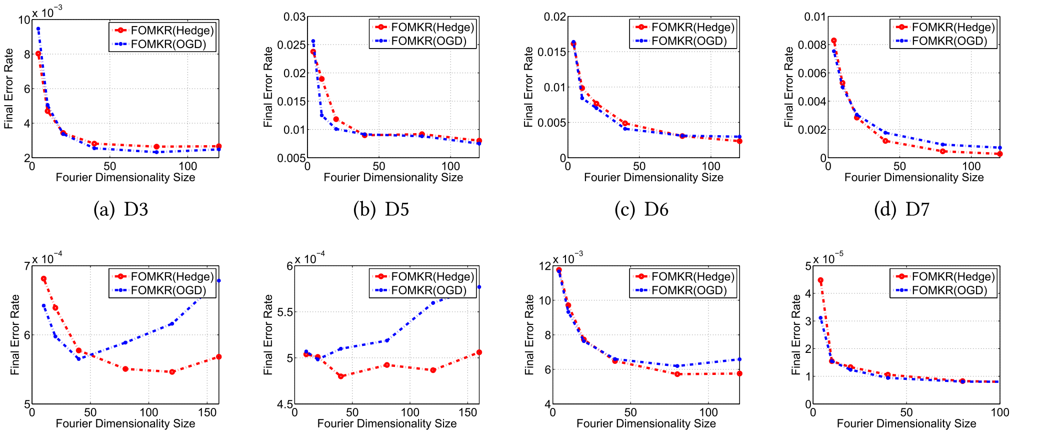
Algorithm	L3	Time (s)	L4	Time (s)
OMKR (Budget)	3.502e−02±0.0e+00	14097	5.574e−05±0.0e+00	6671
FOMKR (Hedge)	**6.455e−03±6.0e−06**	253	**9.412e−06±1.7e−07**	71
FOMKR (OGD)	6.600e−03±5.8e−06	253	**9.413e−06±4.8e−07**	71
NOMKR (Hedge)	1.149e−02±1.2e−17	894	2.093e−05±5.3e−20	1033
NOMKR (OGD)	1.016e−02±1.7e−18	894	1.365e−05±7.6e−19	1033

Note: The numbers represent the final cumulative error obtained by the algorithms. All numbers represent the final cumulative error obtained by the algorithms results are averaged over multiple permutations. The best performances are in bold.

6.2.2 Experimental Settings. For this set of experiments, we consider only 13 RBF kernels $\kappa(x, y) = e^{(\frac{-||x-y||^2}{2\sigma^2})}$ of kernel width parameter σ in $[2^{-6}, 2^{-5}, \ldots, 2^{6}]$. As the datasets are sufficiently large, it is infeasible to run OMKR(Deterministic) for these datasets. We evaluate the performance of naive OMKR with Budget Strategies. We also evaluate the performance of FOMKR and NOMKR with Hedge combination of kernels at the prediction level, and also using OGD combination. In this scenario, we consider the evaluation based on only the mean squared error. For the learning rate, we performed experiments with $\eta = 0.01$ and $\eta = 0.001$ and reported the best performance of each algorithm. For Budget OMKR, we set a budget of $\tau = 500$. For Fourier and Nyström Approximation based strategies, we set the parameters such that the total dimensionality of the new instance obtained from each kernel is 40 features. We further perform analysis of the sensitivity of the algorithm performance with the chosen dimensionality.

6.2.3 Results and Discussion. The results of the analysis of Large-Scale OMKR algorithms can be seen in Table 7. In general we can see that the OMKR variants are significantly better at approximating the kernel function than a naive budget approach. In all datasets, the Approximate OMKR approaches are able to significantly outperform the budget approach. Further, in general we are able to observe that Fourier Features are able to give the best performance. This is a likely result of the fact that we have used RBF kernels for our experiments. It is possible that choosing a higher level of approximation for Nyström features could possible give better results (and the same would apply to Fourier Features). However, Nyström features are relatively more computationally expensive. Having said that, we used RBF kernels only for the purpose of fair comparison between the algorithms. Nyström approach enjoys the ability to even use any arbitrary kernel function, and is not restricted to shift-invariant kernels. Thus, with a better choice of kernels in the predefined pool of kernels, Nyström approximation could potentially give better results.

Also, the approximation methods are much faster than the naive budget methods for OMKR. In all cases for the large datasets, we can see that the OMKR approximation techniques are much faster than the budget techniques. Fourier OMKR is the fastest, followed by Nyström OMKR. This is because Fourier OMKR just involves a simple projection for obtaining new features followed by another projection to obtain the classifier. In general, the proposed approximation techniques offer a promising direction to perform scalable OMKR.



(a) D3 (b) D5 (c) D6 (d) D7

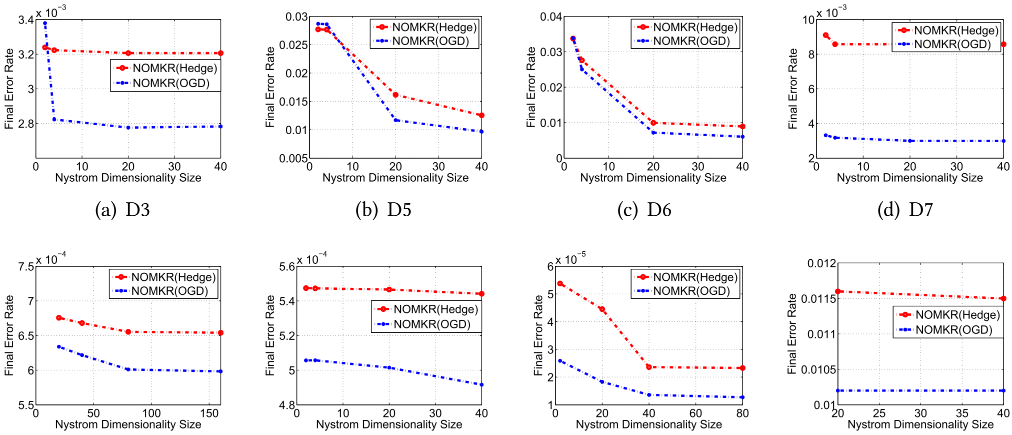
(e) L1 (f) L2 (g) L3 (h) L4

Fig. 6. Performance of FOMKR(Hedge) and FOMKR(OGD) with varying number of approximated features. The Fourier Dimensionality size refers to the number of features obtained per kernels. The total number of features is m times this number. For example, 40 features per kernel corresponds to 520 features for FOMKR (where $m = 13$).

6.2.4 Feature Fusion vs. Prediction Fusion. Here, we evaluate the performance of two-types of kernel combination approaches. Using the functional approximation approach, we can combine the kernel predictions, or we can combine the approximate features obtained from each kernel, and learn a single predictor. Additionally, the performance of both these types of algorithms would depend on the level of approximation or the number of features obtained from each approximation. We conduct experiments for varying levels of approximation using Fourier and Nyström methods on both the smaller datasets (D3, D5, D6, and D7) and the large scale datasets (L1, L2, L3, and L4). These analyses can be visualized in Figure 6 for Fourier feature based OMKR, and in Figure 7 for Nyström method based OMKR.

For the case of Fourier features, we observe that it is in general a stiff competition between the prediction level combination (FOMKR(Hedge)) and the feature level combination (FOMKR(OGD)). We do observe a trend that using more features usually helps, but when it increases too much, the algorithms probably suffer from convergence challenges, and there is performance degradation. In most cases we can see that for a fewer number of features, OGD combination gives a better performance, and as the number of features increases, Hedge combination starts giving improved performance. This is probably due to the fact that Hedge combination uses multiplicative updates over predictors which have fewer number of features, whereas OGD needs to operate on all features simultaneously, and thus starts facing challenges in quick convergence in the online setting. Having said that, the performances are quite similar, and largely depend on the dataset. Probably, for those scenarios where a specific 1–2 kernels could be identified as the best representation of the data, the performance of FOMKR(Hedge) would be better, and in the scenarios where all kernels are relatively weak representations, FOMKR(OGD) would give a better performance.

For the case of Nyström features, we observe the OGD combination invariably achieves better performance than Hedge combination. While this result is contrary to the one seen in the case of Fourier features, this can be explained by the fact that Nyström features in general obtained a worse performance than Fourier Features while approximating RBF kernels. This leads to the realization that each individual kernel obtained from Nyström features contributes with relatively weak predictive ability. Consequently, Hedging which tries to track the best predictor does a poor



(a) D3 (b) D5 (c) D6 (d) D7

(e) L1 (f) L2 (g) L3 (h) L4

Fig. 7. Performance of NOMKR(Hedge) and NOMKR(OGD) with varying number of approximated features. The Nystrom Dimensionality size refers to the number of features obtained per kernels. The total number of features is m times this number. For example, 40 features per kernel corresponds to 520 features for NOMKR (where $m = 13$).

job in comparison to OGD which tries to optimally ensemble a set a weak predictors. This observation is consistent with ensemble approaches such as boosting. However, this does not necessarily imply that Nyström methods are inferior to Fourier Features, as Nsytröm method can generalize to any type of kernel function, and unlike Fourier Features is not restricted to shift-invariant kernels.

6.3 Evaluation of OMKR for Application to Time-Series Prediction

In this section, we evaluate the performance of OMKR when applied to time-series prediction. Specifically, we focus on the problem of identifying the optimal window size of historical data while performing online learning for time-series prediction. For experiments on Online Learning of non-linear time-series prediction, see Section 6.1.

6.3.1 Datasets. We follow the experiments in [30], and consider four synthetic time series settings, and one real world time-series data. After obtaining the sequence, the values are normalized to lie between [0,1]. Consider an ARIMA model given as:

$$\nabla^d y_t = c + \Sigma_{i=1}^p \zeta_i \nabla^d y_{t-i} + \Sigma_{i=1}^q \xi_i \epsilon_{t-j} + \epsilon_t. \tag{46}$$

The first sequence is a stationary time-series, S1 is generated from an $ARIMA(p, d, q)$ model, with $d = 1$, $\zeta = [0.6, -0.5, 0.4, -0.4, 0.3]$ and $\xi = [0.3, -0.2]$. The noise terms are uniformly distributed as $\mathcal{N}(0, 0.3^2)$. The second sequence is a non-stationary time-series model generated by two sets of parameters, with the first half generated by $d = 1$, $\zeta = [0.6, -0.5, 0.4, -0.4, 0.3]$ and $\xi = [0.3, -0.2]$ and the second half generated by $d = 1$, $\zeta = [-0.4, -0.5, 0.4, 0.4, 0.1]$ and $\xi = [0.3, -0.2]$. For S2 the noise terms are distributed as $Uni = [0.5, 0.5]$. Sequence S3 is a non-stationary time-series with $\zeta = [0.6, -0.5, 0.4, -0.4, 0.3]$ and $\xi = [0.3, -0.2]$, but with $d = 1, 2, 3$ for the first, second, and third parts of the sequence respectively. S4 is generated by ARIMA model with $\xi = [0.3, -0.2]$ and $\zeta(t) = [-0.4, 0.5, 0.4, 0.4, 0.1] \times (\frac{t}{10^4}) + [0.6, -0.5, 0.4, -0.4, 0.3] \times (1 - \frac{t}{10^4})$. Finally, we also use a real world time-series dataset S5, which is the daily index value of the Dow Jones Industrial Average from 1885–1962. S1, S2, S3, and S4 comprise 10,000 instances each, while S5 has 35,000 instances.

Table 8. Application of OMKR to Multiple Window Sizes as Each Kernel for Online Learning for ARMA Time-Series Prediction

Algorithm	S1	S2	S3	S4	S5
AR(10) by OGD	0.000650	0.000726	0.000371	0.000554	0.000161
AR(20) by OGD	0.000429	0.000524	0.000196	0.000294	0.000096
AR(30) by OGD	0.000360	0.000453	0.000138	0.000214	0.000072
AR(40) by OGD	0.000324	0.000406	0.000110	0.000177	0.000057
AR(50) by OGD	0.000300	0.000371	0.000094	0.000156	0.000048
AR(60) by OGD	0.000282	0.000344	0.000084	0.000143	0.000041
AR(70) by OGD	0.000269	0.000322	0.000078	0.000133	0.000036
AR(80) by OGD	**0.000259**	0.000306	**0.000074**	0.000125	0.000033
AR(400) by OGD	0.005593	0.010046	0.004760	0.000667	0.000184
AR(800) by OGD	0.059990	0.063424	0.027564	0.001335	0.015066
OMKR(Hedge)	**0.000261**	**0.000298**	0.000086	**0.000111**	**0.000029**

The numbers in bold denote the best performance.

Table 9. Application of OMKR to Multiple Window Sizes as Each Kernel for Online Learning for ARIMA (d = 1) Time-Series Prediction

Algorithm	S1	S2	S3	S4	S5
ARIMA(10) by OGD	0.018026	0.007869	0.000511	0.006375	0.000913
ARIMA(20) by OGD	0.017414	0.007317	0.000370	0.006301	0.000818
ARIMA(30) by OGD	0.017189	0.007135	0.000317	0.006277	0.000789
ARIMA(40) by OGD	0.017076	0.007060	0.000290	0.006250	0.000777
ARIMA(50) by OGD	0.016821	0.007050	0.000273	0.006225	0.000773
ARIMA(60) by OGD	0.016688	0.007059	0.000262	0.006198	0.000772
ARIMA(70) by OGD	0.016509	0.007076	0.000256	0.006189	0.000775
ARIMA(80) by OGD	0.016450	0.007119	**0.000252**	0.006148	0.000778
ARIMA(400) by OGD	0.018305	0.010037	0.002530	0.006092	0.001145
ARIMA(800) by OGD	0.089543	0.027797	0.009196	0.00790	0.002851
OMKR(Hedge)	**0.016353**	**0.006764**	0.000261	**0.005968**	**0.000768**

The numbers in bold denote the best performance.

6.3.2 Baselines and Experimental Setting. We evaluate the time-series for predicting both ARMA (where the next instance in the time-series is to be predicted) and ARIMA (where the next differential in the time-series is to be predicted) target values obtained from the five sequences. We compare against varying window sizes from $[10, 20, 30, 40, 50, 60, 70, 80, 400, 800]$. These window sizes correspond to multiple kernels, and are used in the OMKR framework, which is our proposed method. The algorithms are evaluated on the basis of final mean squared error obtained after the entire online learning process is over. The hedge discount rate parameter is set as $\beta = 0.5$ like in the experiments before, and the learning rate for each window size is set as 0.01.

6.3.3 Results and Discussion. The results of application of OMKR to time-series prediction can be seen in Table 8 for ARMA and Table 9 for ARIMA. In most cases, the OMKR variant is able to achieve the best result as compared to any other individual selection of window size. In some cases, such as in S1 and S3, even though the OMKR performance is not the best by a significant margin, it is very close to the performance of the best window size. These results demonstrate the ability of OMKR to automatically identify the appropriate window size, and simultaneously leverage on complementary information from other window sizes to enhance the prediction performance while doing online learning for time-series prediction. It should further be noted that the appropriate window size for the task is not known before hand.

7 CONCLUSION

This work proposes a family of OMKR algorithms for kernel based regression using a pool of predefined kernels. They overcome the challenges of existing work, which are largely designed for a batch setting and assume that the appropriate kernel function is known. OMKR sequentially learns the kernel based regressor in an online and scalable fashion, and dynamically explores a pool of multiple diverse kernels to avoid problems of poor kernel choice by manual or heuristic selection. However, due to the unbounded number of SVs while learning the model online, OMKR faces severe computational limitations. To address these issues we proposed kernel approximation based strategies, and developed Fourier OMKR and Nyström OMKR algorithms. These algorithms had the added advantage that a combination of kernels at the representation level could also be learnt. Next, we demonstrated application of OMKR to online learning for time-series prediction by showing how the OMKR framework allowed to choose appropriate window-sizes while predicting for ARMA and ARIMA models. We also discussed application to online learning for non-linear time-series prediction. We conducted extensive empirical evaluation and demonstrated the ability of OMKR algorithms to automatically adapt to the best kernel combination from the data during the online learning procedure.

REFERENCES

[1] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. 2013. Online learning for time series prediction. In *COLT*, Vol. 30. 172–184.

[2] Oren Anava, Elad Hazan, and Assaf Zeevi. 2015. Online time series prediction with missing data. In *International Conference on Machine Learning*. 2191–2199.

[3] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32, 1 (2002), 48–77.

[4] Olivier Bousquet and Daniel J. L. Herrmann. 2003. On the complexity of learning the kernel matrix. In *Advances in NIPS*. 415–422.

[5] Dominik Brugger, Wolfgang Rosenstiel, and Martin Bogdan. 2011. Online SVR training by solving the primal optimization problem. *Journal of Signal Processing Systems* 65, 3 (2011), 391–402.

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

[7] Nicolo Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games*. Cambridge University Press.

[8] Badong Chen, Songlin Zhao, Pingping Zhu, and José C. Príncipe. 2012. Quantized kernel least mean square algorithm. *IEEE Transactions on Neural Networks and Learning Systems* 23, 1 (2012), 22–32.

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

[10] Koby Crammer, Jaz Kandola, and Yoram Singer. 2004. Online classification on a budget. In *Advances in NIPS*. MIT Press, Cambridge, MA.

[11] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2008. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing* 37, 5 (2008), 1342–1372.

[12] Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *25th International Conference on Machine Learning*. ACM, 264–271.

[13] Yoav Freund and Robert E. Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, Vol. 904. Springer, Berlin, 23–37.

[14] Joao Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys* 46, 4 (2014), 44.

[15] Mehmet Gönen and Ethem Alpaydın. 2011. Multiple kernel learning algorithms. *Journal of Machine Learning Research* 12 (2011), 2211–2268.

[16] Yujun He, Youchan Zhu, and Dongxing Duan. 2006. Research on hybrid ARIMA and support vector machine model in short term load forecasting. In *6th International Conference on Intelligent Systems Design and Applications (ISDA'06)*. IEEE, vol. 1, 804–809.

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

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

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

[20] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. 2018. Online learning: A comprehensive survey. *arXiv: 1802.02871*.

[21] Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. 2014. Libol: A library for online learning algorithms. *Journal of Machine Learning Research* 15, 1 (2014), 495–499.

[22] Wei-Chiang Hong and Ping-Feng Pai. 2006. Predicting engine reliability by support vector machines. *International Journal of Advanced Manufacturing Technology* 28, 1–2 (2006), 154–161.

[23] Rong Jin, Steven C. H. Hoi, and Tianbao Yang. 2010. Online multiple kernel learning: Algorithms and mistake bounds. In *Algorithmic Learning Theory*, Vol. 6331. Springer, Berlin, 390–404.

[24] Rudolph Emil Kalman. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 82, 1 (1960), 35–45.

[25] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *ICML*, Vol. 3. 321–328.

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

[27] James T. Kwok and Ivor W. Tsang. 2003. Learning with idealized kernels. In *ICML*. 400–407.

[28] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. 2004. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research* 5 (Dec. 2004), 27–72.

[29] Nick Littlestone and Manfred K. Warmuth. 1989. The weighted majority algorithm. In *30th Annual Symposium on Foundations of Computer Science*. IEEE, 256–261.

[30] Chenghao Liu, Steven C. H. Hoi, Peilin Zhao, and Jianling Sun. 2016. Online ARIMA algorithms for time series prediction. In *13th AAAI Conference on Artificial Intelligence*.

[31] Weifeng Liu, Puskal P. Pokharel, and Jose C. Principe. 2008. The kernel least-mean-square algorithm. *IEEE Transactions on Signal Processing* 56, 2 (2008), 543–554.

[32] Jing Lu, Steven C. H. Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. 2016. Large scale online kernel learning. *Journal of Machine Learning Research* 17, 1 (2016), 1613–1655.

[33] Jing Lu, Doyen Sahoo, Peilin Zhao, and Steven C. H. Hoi. 2018. Sparse passive-aggressive learning for bounded online kernel methods. *ACM Transactions on Intelligent Systems and Technology* 9, 4 (2018), 45.

[34] Jing Lu, Peilin Zhao, and Steven C. H. Hoi. 2016. Online sparse passive aggressive learning with kernels. In *2016 SIAM International Conference on Data Mining*. SIAM, 675–683.

[35] Weizhen Lu, Wenjian Wang, Andrew Y. T. Leung, Siu-Ming Lo, Richard K. K. Yuen, Zongben Xu, and Huiyuan Fan. 2002. Air pollutant parameter forecasting using support vector machines. In *2002 International Joint Conference on Neural Networks (IJCNN'02)*. IEEE, vol. 1, 630–635.

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

[37] Edward Moroshko and Koby Crammer. 2013. A last-step regression algorithm for non-stationary online learning. *Artificial Intelligence and Statistics* 451–462.

[38] K.-R. Müller, Alexander J. Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik. 1997. Predicting time series with support vector machines. In *International Conference on Artificial Neural Networks*. Springer, 999–1004.

[39] Tu Dinh Nguyen, Trung Le, Hung Bui, and Dinh Phung. 2017. Large-scale online kernel learning with random feature reparameterization. In *26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. 2543–2549.

[40] Motoya Ohnishi and Masahiro Yukawa. 2017. Online learning in L 2 space with multiple Gaussian kernels. In *25th European Signal Processing Conference (EUSIPCO'17)*. IEEE, 1594–1598.

[41] Francesco Orabona, Joseph Keshet, and Barbara Caputo. 2008. The projectron: A bounded kernel-based perceptron. In *ICML*. ACM, 720–727.

[42] Sophocles J. Orfanidis. 1988. *Optimum Signal Processing: An Introduction*. Macmillan Publishing Company.

[43] Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*. 1177–1184.

[44] Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65, 6 (1958), 386.

[45] W. Rudin. 1990. Fourier Analysis on Groups. Wiley-Interscience Publication.

[46] Doyen Sahoo, Steven Hoi, and Peilin Zhao. 2016. Cost sensitive online multiple kernel classification. In *Asian Conference on Machine Learning*. 65–80.

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

[48] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. 2018. Online deep learning: Learning deep neural networks on the fly. In *27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 2660–2666. DOI: DOI: https://doi.org/10.24963/ijcai.2018/369

[49] Nicholas Sapankevych and Ravi Sankar. 2009. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine* 4, 2 (2009), 24–38.

[50] Bernhard Schölkopf and Alexander J. Smola. 2002. *Learning with Kernels*. MIT Press.

[51] Li Cheng S. V. N. Vishwanathan Dale Schuurmans, Shaojun Wang, and Terry Caelli. 2007. Implicit online learning with kernels. In *NIPS*, Vol. 19. MIT Press, 249.

[52] Shai Shalev-Shwartz. 2007. Online learning: Theory, algorithms, and applications. Ph.D. Thesis. The Hebrew University.

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

[54] Alex J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14, 3 (2004), 199–222.

[55] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. 2006. Large scale multiple kernel learning. *Journal of Machine Learning Research* 7 (2006), 1531–1565.

[56] Francis E. H. Tay and Lijuan Cao. 2001. Application of support vector machines in financial time series forecasting. *Omega* 29, 4 (2001), 309–317.

[57] U. Thissen, R. Van Brakel, A. P. De Weijer, W. J. Melssen, and L. M. C. Buydens. 2003. Using support vector machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems* 69, 1 (2003), 35–49.

[58] Steven Van Vaerenbergh and Ignacio Santamaría. 2013. A comparative study of kernel adaptive filtering algorithms. In *2013 IEEE Digital Signal Processing Workshop and IEEE Signal Processing Education*. Software available at https://github.com/steven2358/kafbox/. DOI: https://doi.org/10.1109/DSP-SPE.2013.6642587

[59] V. G. Vovk. 1995. A game of prediction with expert advice. In *COLT*. ACM, 51–60.

[60] Bernard W.idrow and Marcian E. Hoff. 1960. Adaptive switching circuits. MIT Press Cambridge, MA.

[61] Christopher K. I. Williams and Matthias Seeger. 2001. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*. 682–688.

[62] Lingfei Wu, Ian E. H. Yen, Jie Chen, and Rui Yan. 2016. Revisiting random binning features: Fast convergence and strong parallelizability. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1265–1274.

[63] Yue Wu, Steven C. H. Hoi, Chenghao Liu, Jing Lu, Doyen Sahoo, and Nenghai Yu. 2017. SOL: A library for scalable online learning algorithms. *Neurocomputing* 260 (2017), 9–12.

[64] Zenglin Xu, Rong Jin, Irwin King, and Michael R. Lyu. 2008. An extended level method for efficient multiple kernel learning. In *NIPS*. 1825–1832.

[65] Haiqin Yang, Zenglin Xu, Irwin King, and Michael R. Lyu. 2010. Online learning for group lasso. In *27th ICML*. 1191–1198.

[66] Peilin Zhao, Jialei Wang, Pengcheng Wu, Rong Jin, and Steven C. H. Hoi. 2012. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. In *Proceedings of the 29th International Conference on Machine Learning*. Omnipress, 1075–1082.

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

[68] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *20th International Conference on International Conference on Machine Learning*.