12-2017

Online learning with nonlinear models

Doyen SAHOO
Singapore Management University, doyensahoo.2014@phdis.smu.edu.sg

Online Learning with Nonlinear Models

by

Doyen SAHOO

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

Singapore Management University
2017

Online Learning with Nonlinear Models

Doyen SAHOO

Abstract

Recent years have witnessed the success of two broad categories of machine learning algorithms: (i) Online Learning; and (ii) Learning with nonlinear models. Typical machine learning algorithms assume that the entire data is available prior to the training task. This is often not the case in the real world, where data often arrives sequentially in a stream, or is too large to be stored in memory. To address these challenges, Online Learning techniques evolved as a promising solution to having highly scalable and efficient learning methodologies which could learn from data arriving sequentially. Next, as the real world data exhibited complex nonlinear patterns, it warranted the need for development of learning techniques that could search complex hypotheses space. Among the most notable successful methods for learning nonlinear models are kernel methods and deep neural networks. While these models enable searching complex hypothesis to learn models with a better performance, they are mostly designed for the batch setting which affects their scalability, and they also suffer from the difficulty in selecting the right hypothesis search space (e.g. which kernel to use, what architecture of neural network to use, etc.). In this dissertation we study the intersection of both these fields, and design novel algorithms that combine the merits of both online learning and nonlinear models by proposing methods that can learn nonlinear models in an online setting. Specifically, we investigate Online Learning Algorithms for Multiple Kernel Learning and Deep Neural Networks.

Multiple Kernel Models represent a class of high capacity models which are designed for learning highly nonlinear patterns, and also designed to handle multimodal data. Despite the promising ability, Multiple Kernel Learning is computationally very expensive, and it is a significantly challenging task to use such models

in the online setting. In this dissertation we propose novel Online Multiple Kernel Algorithms, and make the following contributions:

- We propose Online Multiple Kernel Regression Algorithms, which learn a kernel-based regressor in an online fashion, and dynamically explore a pool of diverse kernels to enhance the model performance

- We propose Temporal Kernel Descriptors, i.e., we design new kernels to effectively capture temporal properties of the data, and demonstrate the application of Online Multiple Kernel learning to applications which are sensitive to time.

- We propose Cost-Sensitive Online Multiple Kernel Classification, to address the challenges of learning online nonlinear models from imbalanced data streams, and also demonstrate the application of the proposed methods to online anomaly detection.

Learning with Deep Neural Networks (DNNs) has received increasing interest in recent years due to the overwhelming success demonstrated in several applications. However, using DNNs in the online setting remains an open problem, as most solutions are designed for the batch setting. In particular, choosing a right model architecture for online learning is a challenging task (in addition to convergence challenges such a vanishing gradient and diminishing feature reuse). To address these limitations, we develop algorithms for Online Deep Learning:

- We develop a novel Hedge Backpropagation algorithm which evolves the DNN from shallow to deep, thereby making DNNs online compatible. This way they are able to enjoy the fast convergence of Online Learning, and the power of representation of Deep Learning.

Table of Contents

List of Figures

List of Tables

List of Publications

Most of the content in this dissertation has been presented in the following publications.

Main Publications

Doyen Sahoo, Quang Pham, Steven C.H. Hoi, and Jing Lu. Online Deep Learning: Learning Deep Neural Networks on the Fly. Available at `https://arxiv.org/abs/1711.03705`.

Doyen Sahoo, Steven C.H. Hoi, and Bin Li. Large-scale Online Multiple Kernel Regression with Application to Time-Series Prediction. Under review for *ACM TKDD*.

Doyen Sahoo, Peilin Zhao, and Steven C.H. Hoi. Cost-Sensitive Online Multiple Kernel Classification. In *Asian Conference on Machine Learning (ACML)*,2016.

Doyen Sahoo, Abhishek Sharma, Steven C.H. Hoi, and Peilin Zhao. Temporal Kernel Descriptors for Learning with Time-sensitive Patterns. In *SIAM International Conference on Data Mining (SDM)*,2016.

Doyen Sahoo, Steven C.H. Hoi, and Bin Li. Online multiple kernel regression.In *20th ACM SIGKDD international conference on Knowledge discovery and data mining.*, 2014.

The first two publications are under review, and the remaining three have been published in peer-reviewed conferences. All publications are on the topic of Online Learning with Non-linear Models. While the first publication on Online Learning with Deep Neural Networks addresses the problem of learning DNNs on the fly, the other publications are concerned with online learning using multiple kernels.

Others

The following publications were a part of the author's work not directly related to the dissertation topic.

Doyen Sahoo, Chenghao Liu, and Steven C.H. Hoi. Malicious URL Detection using Machine Learning: A Survey. Under review for *ACM TWEB*. Also available at `https://arxiv.org/abs/1701.07179`

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

Li, Bin, **Doyen Sahoo**, and Steven C.H. Hoi. OLPS: A Toolbox for On-Line Portfolio Selection. *Journal of Machine Learning Research (JMLR)*, 2016.

Li, Bin, Steven CH Hoi, **Doyen Sahoo**, and Zhi-Yong Liu. Moving average reversion strategy for on-line portfolio selection. *Artificial Intelligence*, 2015.

Acknowledgments

Though my journey from the date of admission to Ph.D. studies till submission of this dissertation has been roller coaster ride, it has been full of fun throughout and a day never passed without a learning something new. The person singularly responsible for this remarkable journey is Dr. Steven C. H. HOI, who guided and supervised my work. More importantly, he worked with me whenever I got stuck up and showed me the direction whenever I moved wayward. Intellectual prowess combined with passion and dedication, Dr. Hoi led by example, always giving his 100% and expecting nothing less in return. Discussions and arguments with him during my research are by far my favorite moments, where his sharp insights allowed me to learn more in an hour than what I would otherwise in a month. I could not have asked for a better adviser and mentor and for all the ways Dr. Hoi has helped me in this journey, I shall forever remain grateful.

I am also extremely grateful to other members of my dissertation committee, Dr. Hady Lauw and Dr. Akshat Kumar for their insightful comments and encouragement, that incented me to widen my research from various perspectives and avoid many potential insufficiencies. I would also like to thank Dr. Wee Sun Lee for identifying several limitations, and suggesting directions to improve the quality of the work.

I would like to thank my senior labmates, Dr. Peilin Zhao and Dr. Bin Li, with whom I spent sleepless nights on several projects to meet deadlines. They initiated me to the world of research and guided me through the difficult times. They helped me to cope with failures and overcome challenges time and again. They are fantastic role models, without whose guidance, my journey would not have been so pleasant. I am also grateful to my colleagues in Dr. Hoi's group: Jing Lu, Quang Pham, Hung Le, Chenghao Liu, Wan Ji, Wu Yue, Shuji Hao, Xingyu Gao, Xiongwei Wu, Pengcheng Wu, Dayong Wang, Xia Hao, Yi Ding, and Jintao Li. I am extremely privileged to be a member of such a talented group.

I would also like to thank all my friends and colleagues in SIS Research Center: Abhishek Sharma, Maksim Tkachenko, James Arambam, Kenny Choo, Camellia Zakaria, Felicia Natali, and Dung Le, who were not only worthy peers for intellectual discussions, but also provided me with familial support during my Ph. D. studies.

I am also grateful to Singapore Management University and Nanyang Technological University for providing me with amazing opportunities and an environment conducive for research. This research and my Ph. D. studies would not have been possible but for the generous scholarship.

Finally, I would like to thank my parents, who never allowed me to give up.

Chapter 1

Introduction

In this chapter, we introduce the background, and motivate the need for online learning with nonlinear models. Then we present the formal problem setting and discuss the main challenges. This is followed by briefly presenting the main contributions of this dissertation.

1.1 Background

Recent years have witnessed tremendous applications of big data analytics. The most critical challenges of big data analytics are categorized by the 3 Vs: Volume, Velocity, and Variety (Veracity or uncertainty in the quality of data is another major challenge, but it is not in the scope of this dissertation). Volume refers to the huge amount of data that needs to be analysed drawing attention to the limitations of models and machines that may not have the computational resources to process such huge data. Velocity refers to the high speed at which new data is being generated, which highlights the necessity to have models that can quickly process the new data and learn the relevant patterns. Variety refers to the format (unstructured, multi-modal, heterogeneous) in which the data appears and the highly complex patterns (nonlinear patterns) it may exhibit. These challenges have created a demand for scalable machine learning algorithms that have the capacity to learn complex patterns.

Traditionally, several challenges posed by volume and velocity of the data are handled by Online Learning methods which represent a class of highly efficient and scalable machine learning algorithms that learn from streaming data. The challenges posed by the variety and complexity of data are tackled through the usage of nonlinear models, some of which are designed explicitly for multi-modal data (e.g. multiple kernel learning) and others which aim to learn feature representation (e.g. Deep Learning). In this dissertation, we investigate the intersection of Online Learning and Nonlinear Models. Our goal is to design algorithms that perform Online Learning with Nonlinear models, in order to effectively address the challenges of big data analytics. There have been many approaches to learn models in the online setting through Online Learning, and also to learn nonlinear models through the usage of kernel methods and deep neural networks. However, there have been limited contributions in literature on the intersection of both these approaches. While individually each of them solve several challenges, both of them have several limitations that restrict their usage in the real world setting. Here, we briefly highlight the main merits and limitations of these two bodies of work.

1.1.1 Online Learning

Machine Learning techniques can be broadly categorized into batch learning and online learning. Batch Learning refers to training a prediction model on the entire training data, whereas online learning refers to training from instances of the training data arriving in a sequential stream. Online Learning [105, 51] dates back to the classical Perceptron[99]. There have been many follow up works in this domain with many approaches gaining popularity in the last decade. These include Online Gradient Descent [134], Relaxed Online Maximum Margin Algorithm (ROMMA) [73], Approximate Maximal Margin Algorithm (ALMA) [40], Margin Infused Relaxed Algorithm (MIRA) [24], Passive Aggressive (PA) algorithms [21], Confidence Weighted Algorithms [33], etc.

Online Learning techniques have several advantages over batch learning methods. We discuss the main advantages below:

Merits of Online Learning

- **Efficiency and Scalability**: Due to the sequential nature of processing the data, and continuously learning, Online Learning methods are very efficient and can scale to extremely large datasets. Another major impact of the training procedure is that for any new data that is observed, the model can be updated incrementally, and retraining on the entire new dataset can be avoided.
- **Adaptation to Temporal Changes**: Online Learning processes the data streams one instance at a time. As a result, if the data exhibits certain temporal patterns, the latest training updates to the model will allow adaptation to such temporal patterns. This makes online learning methods a natural solution to deal with concept drifting data.
- **Theoretical Guarantees**: Despite the efficiency, it is not surprising than batch models would show a superior performance that online algorithms. However, most online learning algorithms are designed such that they enjoy a sublinear regret with respect to the best batch model. This means that the performance of online learning algorithms is bounded in such a manner, that they are not much worse than the corresponding batch models.

Limitations of Online Learning

Despite their speed and memory efficiency, Online Learning approaches suffer from some critical drawbacks. We briefly discuss the main shortcomings below:

- **Limited Capacity**: Most online learning techniques are primarily designed for learning linear models, which limits their application for real world setting where the patterns to be learnt are significantly more challenging and nonlinear. While there have been attempts at using online learning with kernels to learn nonlinear models [61], these approaches are still limited in their functional expressiveness, which hurts their ability to learn highly complex patterns.

- **Fixed Model Capacity**: For most models used in the online setting, the model capacity (or the hypothesis search space) is fixed prior to the learning task. For example, for online learning with kernels, the kernel to be used must be specified before the training begins. Similarly, for using a deep neural networks for the online setting, the architecture of the network must be fixed prior to learning. This can become problematic in a streaming setting where it is almost impossible to guess which model capacity to use. This problem is more severe in cases of concept drifting scenarios [39] or data that exhibits temporal patterns, where different feature representations (through different kernels or different neural networks) could be more appropriate. Consequently, the fixed model capacity limits the potential of existing online approaches from capturing richer information about the data.

- **Nontrivial Adaptation to High Capacity Models**: Another limitation of using higher capacity models in the online setting is that it is significantly challenging to design online learning algorithms that can effectively use these high capacity models. In fact high capacity models such as multiple kernel learning and deep neural networks face tremendous challenges while being optimized in the batch setting itself. Using these models in the online setting faces several problems from the perspective of algorithmic design, computational complexity, and other optimization challenges.

1.1.2 Nonlinear Models

Nonlinear models refer to class of high capacity models that are able to express rich functions, and are thus very useful in learning highly complex nonlinear patterns exhibited in real world data. The nonlinearity is popularly achieved through the usage of kernel methods [108, 103], or through the usage of Deep Neural Networks [68]. We discuss the main advantages of nonlinear models over linear models below.

Merits of Nonlinear Models

- **Higher Expressive Capacity**: Real world data is filled with many complex nonlinear functions of the input (e.g. Computer Vision Tasks, XOR Separation, etc.). Linear models simply do not have the ability to even represent such complex functions. Consequently, nonlinear models that could learn much more complex functions gained popularity, and demonstrated superior performance in a variety of tasks. Most notably kernel methods [103] were immensely successful in the last two decades, and in the last decade Deep Neural Networks have demonstrated state of the art performances in several applications [68]. Nonlinear models, with a higher expressive capacity, enable searching for a significantly more complex hypothesis space, which enables superior performance on real world data.

- **Representation Learning**: Nonlinear models such as Multiple Kernel Learning and Deep Neural Networks are able to learn the important features of the data on their own, and thus require limited human intervention to perform feature engineering. This trait makes nonlinear models more appealing for completely automated end to end learning.

- **Ability to handle multi-modal data**: Another common occurrence in real world data is that the features are heterogeneous or multi-modal in nature. Treating all the features alike can degrade the predictive power of the models. Additionally, the target output may have linear dependence on one modality and nonlinear dependence on another. Using nonlinear models appropriately allows for handling of such scenarios.

Limitations of Nonlinear Models

However, using nonlinear models have several shortcomings with regard to computational complexity, especially when streaming data needs to be processed. We highlight the main drawbacks below:

- **Scalability**: While nonlinear models are able to learn expressive functions, they come coupled with several computational challenges, as they are mostly designed for the batch setting. Nonlinear models usually require the estimation of many more parameters than linear models, which means the learning process requires significantly more computation, and also would often suffer from slower convergence. In addition, training for most nonlinear models assumes that the entire training data is made available prior to the training task. Often, the data may be too large to be stored in memory, or may even arrive in a streaming manner. These two challenges limit the scalability of typical nonlinear models.

- **Retraining Cost**: Another severe issue with most existing batch nonlinear models is that for any new training data that arrives, to retrain the model, the entire optimization procedure must be repeated. This computational challenge prevents nonlinear models from being regularly updated.

- **Inability to adapt to Temporal Patterns**: Considering that in several real world applications data arrives in a streaming manner, and has temporal properties, it is imperative to have models that can automatically adapt to such a temporal nature. Capturing temporal properties, especially for newly arriving data is very difficult to handle for nonlinear models.

In this dissertation, our aim is to investigate the intersection of the Online Learning and Nonlinear models, and design new learning algorithms that are able to combine the merits of both these fields of work, and simultaneously address the drawbacks of each. Next, we briefly discuss the merits and limitations of Online

Learning and Nonlinear models, and motivate the need for the contributions of this dissertation.

1.1.3 Online Learning with Nonlinear Models

In order to effectively address the main challenges of big data analytics, we need methods that can combine online learning and nonlinear models. Existing approaches approaches for online learning with nonlinear models are mostly through the usage of kernel functions [61]. Enhanced models such as multiple kernel learning have also been developed for the online setting [58, 49]. These methods, however, do not extensively explore several aspects of Online Learning. In particular, they are not designed for regression tasks, temporal patterns, or learning from imbalanced data streams. Moreover, they do not comprehensively explore kernel combination techniques that could improve the performance of Online Multiple Kernel Learning. Another relatively less explored category of online learning with nonlinear models is Online Learning with Deep Neural Networks. Deep Neural Networks face several challenges which prevent their usage in the online setting. While there have been heuristic attempts at using them in the online setting, these attempts rely on mini-batch optimization making them unsuitable for streaming data. In the next section we formally present the problem setting for Online Learning with nonlinear models.

1.2 Problem Setting

Now we describe the main problem setting for Online Learning with Nonlinear Models. Without loss of generality, consider a generic learning task, in which we are given a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in |\mathcal{Y}| = \{-1, +1\}$ for binary classification, $y_t \in \{0, 1\}^C$ for multi-class classification with C classes, and $y_t \in \mathbb{R}$ for regression tasks, is the target label assigned to $\mathbf{x}_t$. The goal of online learning with nonlinear models is to learn a function $\mathbf{F} : \mathbb{R}^d \rightarrow \mathbb{R}^{|}\mathcal{Y}|$, which is a nonlinear function of the input $\mathbf{x}$. The prediction is denoted by $\hat{y}_t$. The performance of the learnt function is usually evaluated based on the cumulative penalty suffered by the prediction model. This penalty can be the error rate in classification prediction, or a squared loss for regression. To learn a prediction function which that minimizes this penalty over T instances, a loss function (hinge-loss, squared loss, cross-entropy, etc.) is often chosen for minimization. We denote this loss function as $\mathcal{L}(\mathbf{F}(\mathbf{x}), y)$.

Traditional approaches (Linear SVMs, Logistic Regression, Linear Regression, etc.) to learn such a prediction function are linear. This means that the prediction function takes the following form: $\mathbf{F}(\mathbf{x}) = \mathbf{w}^\top\mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^d$. However, linear models are limited in their capacity to learn expressive functions. As a result, in this dissertation, our aim is to learn nonlinear functions. Nonlinear functions that we consider in this dissertation are kernel functions, multiple kernel functions and deep neural networks. While linear models have only d parameters to be estimated for the function $\mathbf{F}(\mathbf{x})$, nonlinear models usually have more parameters. For example, in kernel methods, for a given kernel, the prediction function takes the form of a linear combination of all the instances when projected into a high dimensional space by the kernel - and the weight of each instance has to be estimated. Similarly, for deep neural networks, the function $\mathbf{F}(\mathbf{x})$ is a set of stacked linear functions each followed by a nonlinear activation function (e.g. sigmoid) - and the weight parameters of each stack have to be estimated. We explain the details of the methods in a later section.

For online learning of the parameters of function $\mathbf{F}(\mathbf{x})$, the learning algorithm follows a typical online setting, where instances $\mathbf{x}_t, t = 1, \ldots, T$ arrive sequentially, and a prediction $\hat{y}_t$ is made on each instance. This is followed by the environment revealing the true target y_t. Using this feedback, the parameters of the nonlinear function $\mathbf{F}(\mathbf{x})$ are updated. This framework is outlined in Algorithm 1. Based on specific challenges that are posed by the data and remain unaddressed by the existing approaches for Online Learning with Nonlinear Models, our goal is to design new update strategies for the models.

Algorithm 1 Online Learning with Nonlinear Models

```
**Initialization**: Initialize $\mathbf{F}(\mathbf{x})$ with capacity to learn nonlinear models
for $t = 1, 2, \ldots, T$ do
    Receive an instance: $\mathbf{x}_t$
    Predict $\hat{y}_t = \mathbf{F}_t(\mathbf{x}_t)$
    Receive target value: $y_t$
    Update $\mathbf{F}_{t+1}(\mathbf{x}) \leftarrow UpdateStrategy(\mathbf{F}_t(\mathbf{x}))$
end for
```

1.3 Contribution Summary

In this dissertation we aim to design novel methods for online learning with nonlinear models, addressing specific limitations of existing approaches for this task, with special focus on temporal patterns exhibited by the data. Figure 1.1 shows the taxonomy of the subfields of machine learning, where our primary contributions lie. Specifically, we focus on Online Multiple Kernel Learning and Online Deep Learning. Existing Online MKL strategies are limited, and do not address several concerns especially for temporal properties that may exist. We also develop novel algorithms for Online Deep Learning which is an open problem, with no clear solutions in literature. Below we highlight these limitations and discuss the main contributions in this dissertation.

- **Online Multiple Kernel Regression**: We propose Online Multiple Kernel Regression Algorithms, which learn a kernel-based regressor in an online fashion, and dynamically explore a pool of diverse kernels to enhance the model performance. This builds on the existing OMKL work which is primarily designed for classification tasks [58, 49] or for structured prediction[81]. We extend the OMKL principles for regression tasks, and improve the kernel combination methods over the previous work. We also propose and develope kernel approximation strategies for large scale online multiple kernel regression. Further, we demonstrate its application to Time Series Modeling.

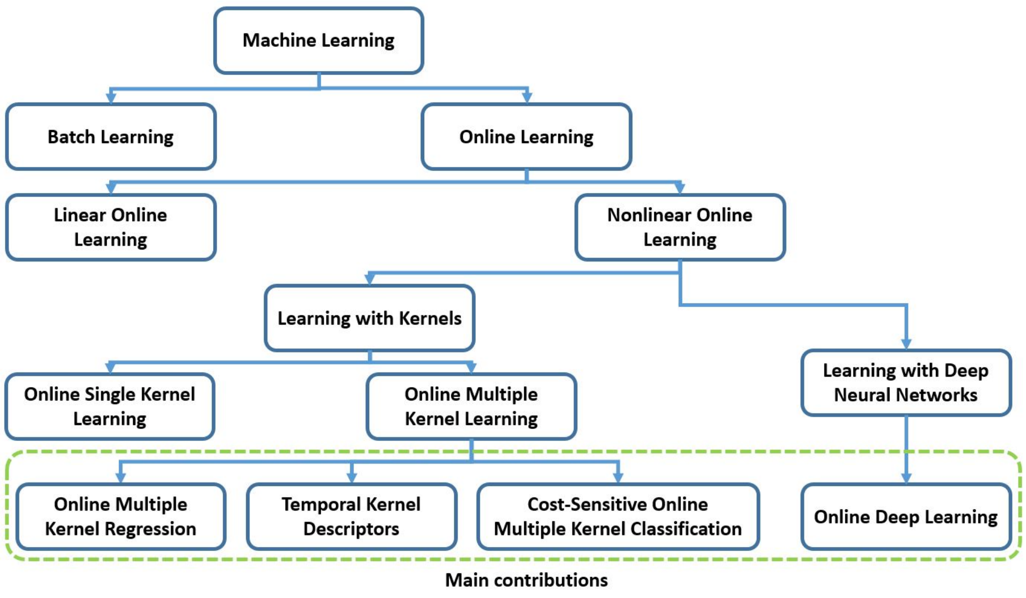

Machine Learning
Batch Learning
Online Learning
Linear Online Learning
Nonlinear Online Learning
Learning with Kernels
Learning with Deep Neural Networks
Online Single Kernel Learning
Online Multiple Kernel Learning
Online Multiple Kernel Regression
Temporal Kernel Descriptors
Cost-Sensitive Online Multiple Kernel Classification
Online Deep Learning
Main contributions

Figure 1.1: Main Contributions of the Dissertation

Related Publications

- **Doyen Sahoo**, Steven C.H. Hoi, and Bin Li. Large-scale Online Multiple Kernel Regression with Application to Time-Series Prediction. Under review for *ACM TKDD*.
- **Doyen Sahoo**, Steven C.H. Hoi, and Bin Li. Online multiple kernel regression.In *20th ACM SIGKDD international conference on Knowledge discovery and data mining.*, 2014.

• **Temporal Kernel Descriptors for Learning with Time-Sensitive Patterns**: Successful performance of kernel based methods depends on the choice of the kernel. We consider scenarios where patterns are sensitive to time (or time-stamps), and accordingly we propose Temporal Kernel Descriptors. These kernel descriptors are able to automatically learn the association of time-stamps at different resolutions (e.g. hourly, daily, etc.) with different data modalities. This enables more effective capturing of the information, and results in superior performance as compared to directly using traditional kernels.

Related Publications

- **Doyen Sahoo**, Abhishek Sharma, Steven C.H. Hoi, and Peilin Zhao. Temporal Kernel Descriptors for Learning with Time-sensitive Patterns. In *SIAM International Conference on Data Mining (SDM)*,2016.

- **Cost-Sensitive Online Multiple Kernel Classification**: Often data streams are can be highly imbalanced, which may make evaluation of algorithms on common metrics such as accuracy unreliable. Consequently, for imbalanced data streams, we consider evaluation over cost-sensitive metrics. We then propose Cost-Sensitive Online Multiple Kernel Classification and also demonstrate the application of the proposed methods to online anomaly detection. To achieve this we develop new cost-sensitive kernel learning approaches and cost-sensitive kernel combination approaches.

  Related Publications

  - **Doyen Sahoo**, Peilin Zhao, and Steven C.H. Hoi. Cost-Sensitive Online Multiple Kernel Classification. In *Asian Conference on Machine Learning (ACML)*,2016.

- **Online Deep Learning**: Another dimension that we aim explore for Online Learning with Nonlinear Models is Online Deep Learning. Deep Learning has experienced tremendous success in the recent years in terms of performance (and have shown superior performance compared to kernel based techniques). However, it is mostly designed for the batch setting, and would significantly benefit from having an online variant. We leverage on the shallow to deep principle, according to which shallow networks converge faster than deeper networks, and propose a novel *Hedge Backpropagation* algorithm to learn Deep Neural Networks online.

  Related Publications

- **Doyen Sahoo**, Quang Pham, Steven C.H. Hoi, and Jing Lu. Online Deep Learning: Learning Deep Neural Networks on the Fly. Under review for *AAAI 2018*

1.4 Dissertation Structure

We first present the relevant literature for Online Learning with Nonlinear Models in Chapter 2. Here we conduct brief surveys on the two main bodies of work (that are relevant for this dissertation): Online Learning and Learning with Nonlinear Models. We then survey the work at the intersection of these studies.

This is followed by our contributions: Online Multiple Kernel Regression in Chapter 3, Temporal Kernel Descriptors for Learning with Time-Sensitive Patterns in Chapter 4, and Cost-Sensitive Online Multiple Kernel Classification in Chapter 5. We then present Online Deep Learning in Chapter 6. We conclude with the dissertation and offer directions for future research in Chapter 7.

Chapter 2

Literature Review

2.1 Introduction

This chapter reviews the related work in literature. We mainly focus on the work related to the two broad categories: Online Learning and Learning with Nonlinear Models. We then discuss the work that lies at the intersection of these two fields, and highlight the limitations of these efforts.

2.2 Online Learning

Online Learning refers to class of highly efficient and scalable algorithms that learn from data streams by processing the data sequentially instance by instance [14, 105, 51]. Due to the nature of on-the-fly learning Online Learning has found tremendous success in several applications [52], particularly to counter the big data challenges of volume, velocity, and variety.

In this section, we will give a formal overview of Online Learning, followed by discussing some applicatins where Online Learning has been found to be successful. This if followed by presenting a brief overview of three categories of supervised online learning: First-Order Online Learning, Second Order Online Learning, and Online Learning with Expert Advice.

2.2.1 Problem Setting and General Framework

Consider a generic supervised machine learning task (e.g. classification, regression). Given a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in |\mathcal{Y}| = \{-1, +1\}$ for binary classification, $y_t \in \{0, 1\}^C$ for multi-class classification with C classes, and $y_t \in \mathbb{R}$ for regression tasks, is the target label assigned to $\mathbf{x}_t$. The goal of online learning is to learn a function $\mathbf{F} : \mathbb{R}^d \rightarrow \mathbb{R}^{|\mathcal{Y}|}$. The prediction is denoted by $\hat{y_t}$. The performance of the learnt function is usually evaluated based on the cumulative penalty suffered by the prediction model. This penalty can be the error rate in classification prediction, or a squared loss for regression. To learn a prediction function which that minimizes this penalty over T instances, a loss function (hinge-loss, squared loss, cross-entropy, etc.) is often chosen for minimization. We denote this loss function as $\mathcal{L}(\mathbf{F}(\mathbf{x}), y)$. The instances $\mathbf{x}_t, t = 1, \ldots, T$ arrive sequentially, and a prediction $\hat{y_t}$ is made on each instance. This is followed by the environment revealing the true target y_t. Using this feedback, the parameters of the nonlinear function $\mathbf{F}(\mathbf{x})$ are updated. This framework is outlined in Algorithm 2.

Algorithm 2 Online Learning

```
Initialization: Initialize $\mathbf{F}(\mathbf{x})$
for $t = 1, 2, \ldots, T$ do
    Receive an instance: $\mathbf{x}_t$
    Predict $\hat{y_t} = \mathbf{F}_t(\mathbf{x}_t)$
    Receive target value: $y_t$
    Update $\mathbf{F}_{t+1}(\mathbf{x}) \leftarrow UpdateStrategy(\mathbf{F}_t(\mathbf{x}))$
end for
```

2.2.2 Applications of Online Learning

The specific problem setting of Online Learning is found in several applications including Cyber Security, Finance, Recommendation Systems, etc. In particular online learning algorithms aim to learn prediction models from large data streams.

For example, in Finance, a popular application is Online Portfolio Selection

[71, 72]. Here, the function $\mathbf{F}$ to be learnt is the portfolio vector, i.e., to identify how to distribute wealth in a set of assets with a specific objective (e.g. maximizing wealth). From the perspective of the online setting, in every time period (or iteration), the environment reveals a returns vector (corresponding to instance $\mathbf{x}$). Based on the current portfolio vector, the algorithm realizes some profit or loss. Using this loss information, the algorithm updating the portfolio vector through some strategy, thereby modifying the prediction function $\mathbf{F}$. Similar settings are also observed in time-series modeling or high frequency trading, where decisions are to be made immediately based the streaming environmental factors. Another example is in anomaly detection [128], where the activities are being monitored continuously. In every time period, the the environment reveals some actvity recorded in the form of a vector $\mathbf{x}$. The prediction model decides whether this activity is an anomaly or not using the prediction function $\mathbf{F}(\mathbf{x})$. Consequently, the model may recieve some feedback (e.g. from a human user, or new statistics about the data distribution), according to which the model would update its prediction function. While Algorithm 2 shows a specific example of supervised learning, in reality online learning may also be unsupervised. Another very common application of Online Learning is in Recommendation Systems [27], where the data in the form of user ratings arrive sequentially and rapidly, and accordingly the recommendation system needs to be modified. Moreover, user preferences could evolve over time, and thus the models should be able to online adapt to such temporal patterns.

While the above example demonstrates application of online learning in a natural online setting, Online Learning is also popularly used for training models on very large data, which can not be stored in memory, and models need to be updated regularly. For example, a popular application is Malicious URL Detection [101]. Typically, labelled data is obtained in the form of blacklists, and prediction models are trained. Twp specific challenges arise: (i) The amount of data can be too large to be stored in memory to training a batch learning model; (ii) the model needs to be updated when fresh data is available to keep the model updated. Using the labelled data, one can train a prediction model in an online manner and not need the resources to store the entire data in memory. Further, as fresh data is available, the model can be updated using online update rules, instead of training the entire model from scratch.

Based on the application, the learning process could be for various types of settings (e.g. supervised [71], unsupervised[6], etc.) and could operate on different varieties of data (time series [2], graphs [35, 82], ratings[27], etc.). While some problems focus on specific prediction tasks [80], others aim to understand and summarize statistical properties of the data [57, 44].

Next we review some of the important related topics in supervised online learning with respect to the main conributions in this dissertation. Online Learning (using linear models) is traditionally categorized into First Order and Second Order Online Learning, based on the type of information used by the Update Strategy. Here we also discuss a third category Online Learning for Prediction with Expert Advice.

2.2.3 First Order Online Learning

First-order algorithms learn by updating the weight vector $\mathbf{w}$ for classification sequentially by utilizing only the first-order information with training data. We briefly describe a few exemplar first-order online learning algorithms below.

Perceptron [99] is the earliest online learning algorithm. In each iteration, whenever a mistake is made by the prediction model, Perceptron makes an update as follows:

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

Online Gradient Descent (OGD) [134] updates the weight vector $\mathbf{w}$ by applying the (Stochastic) Gradient Descent principle only to a single training instance arriving sequentially. Specifically, OGD makes an online update iteratively as:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla \ell(\mathbf{w}_t, \mathbf{x}_t; y_t)$$

where η is a step size parameter, and $\ell(\mathbf{w}_t, \mathbf{x}_t; y_t)$ is some predefined loss function, e.g., Hinge-Loss, Negative Log-Likelihood, Squared Loss, etc.

Passive-Aggressive learning (PA) [21] is an online learning method that trades off two concerns: (i) passiveness: to avoid the new model deviating too much from the existing one, and (ii) aggressiveness: to update the model by correcting the prediction mistake as much as possible. The optimization of PA learning can be cast as follows:

$$\mathbf{w}_{t+1} \leftarrow \underset{\mathbf{w}}{\text{argmin}} \frac{1}{2}||\mathbf{w}_t - \mathbf{w}||^2 \quad \text{subject to } y_t(\mathbf{w} \cdot \mathbf{x}_t) \geq 1$$

The closed-form solution to the above can be derived as the following update rule:

$$\begin{aligned} &\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \tau_t y_t \mathbf{x}_t \\ \text{where} \quad &\tau_t = \max\big(\frac{1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)}{||\mathbf{x}_t||^2}, 0\big) \end{aligned}$$

The above model assumes a hard margin exists, that is, data can be linearly separable, which may not be always true, especially when data is noisy. To overcome this limitation, soft-margin PA variants, such as PA-I and PA-II, are often commonly used, which also have closed-form solutions [21].

There are many other first-order online algorithms in literature including Relaxed Online Maximum Margin Algorithm (ROMMA) [73], Approximate Maximal Margin Algorithm (ALMA) [40], Margin Infused Relaxed Algorithm (MIRA) [24].

2.2.4 Second Order Online Learning

Unlike the first order online learning, second order online learning aims to boost the learning efficacy by exploiting second-order information, e.g., the second order statistics of underlying distributions. For example, they usually assume the weight vector $\mathbf{w}$ follows a Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. This is particularly useful for high dimensional sparse features (e.g. bag-of-words features). The second order information along with the first helps in accelerating the convergence. Next we briefly describe some popular second-order algorithms.

Confidence-Weighted learning (CW) [33] is similar to the PA learning algorithms in terms of passiveness and aggressiveness tradeoff, except that CW exploits the second-order information. In particular, CW learning maintains a different confidence measure for each individual feature, such that weights of lower confidence will be updated more aggressively than those of higher confidence. Specifically, by modeling the weight vector as a Gaussian distribution, CW trades off between (i) passiveness: to avoid the new distribution of the model from deviating too much from the existing one; and (ii) aggressiveness: to update the model by not only correcting the prediction mistake if any, but also improving the classification confidence. More specifically, the CW learning can be cast into the following optimization:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \leftarrow \operatorname*{argmin}_{\boldsymbol{\mu},\Sigma} D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \Sigma)||\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t))$$
$$\text{subject to } y_t(\boldsymbol{\mu}, \mathbf{x}_t) \geq \phi^{-1}(\eta)\sqrt{\mathbf{x}_t^\top \Sigma \mathbf{x}_t}$$

Like the PA algorithms, the closed-form solutions for the CW optimization can be derived.

Adaptive Regularization of Weights (AROW) is an enhanced variant of CW algorithms which is designed for linearly non-separable data. Here the CW optimization objective is modified into:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \leftarrow \operatorname*{argmin}_{\boldsymbol{\mu},\Sigma} D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \Sigma)||\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)) + \lambda_1 \ell(y_1, \mu \cdot \mathbf{x}_t) + \lambda_2 \mathbf{x}_t^\top \Sigma \mathbf{x}_t$$

Through this formulation, AROW acquires the two desirable properties of CW learning (not to change to model too radically, and have a small loss on the last instance processed), and in addition the third term of the formulation helps increase the confidence in the parameters as more examples are seen.

CW algorithms have seen additional enhancements in the form of Soft-Confidence Weighted Learning [118]. Other second order online learning algorithms include IELLIP [127] which assumes that the classifier $\mathbf{w}$ lies on an ellipsoid, NAROW [90], and NHERD [23].

2.2.5 Online Learning with Expert Advice

Online prediction with expert advice deals with making a final prediction based on the prediction made by a set of experts. Consider N experts. At each time step $t = 1, 2, \ldots, T$, the algorithm decides on a distribution $\mathbf{p}_t$ over the experts, where $p_{t,i} \geq 0$ is the weight allocated to expert i, and $\sum_{i=1}^{N} p_{t,i} = 1$. Each expert i then suffers some loss $\ell_{t,i}$ which determined by the environment (like the general online learning setting). The loss suffered by the algorithm is then $\sum_{i=1}^{N} p_{t,i}\ell_{t,i} = \mathbf{p}_t^\top \ell_t$, i.e., the average loss of the experts with respect to the distribution chosen by the algorithm. Without loss of generality, it is assumed that $\ell_{t,i} \in [0, 1]$.

In this section, we present an algorithm, called Hedge [37], for the online prediction with expert advice. This algorithm is a direct generalization of Littlestone and Warmuth's weighted majority algorithm [75]. The algorithm maintains a non-negative weight vector whose value at time t is denoted $\mathbf{w}_t = (w_{t,1}, \ldots, w_{t,N})$. If it is believed that one expert performs the best, it is better to assign it the most weight. If no prior is known, it is better to set all the initial weights equally, i.e., $w_{1,i} = 1/N$ for all i. The algorithm uses the normalized distribution to make prediction, i.e.,

$$\mathbf{p}_t = \frac{\mathbf{w}_t}{\sum_{i=1}^{N} w_{t,i}}$$

After the loss ℓ_t is disclosed, the weight vector $\mathbf{w}_t$ is updated using a multiplicative rule

$$w_{t+1,i} = w_{t,i}\beta^{\ell_{t,i}}, \quad \beta \in [0, 1]$$

which implies that the weight of expert i will exponentially decrease with the loss $\ell_{t,i}$. This allows Hedge to track the performance of the best expert.

Besides Hedge, there are some other algorithms for online prediction with expert advice under more challenging settings, including exponentially weighted average forecaster (EWAF) and Greedy Forecaster (GF) [14]. The difference of EWAF from Hedge is that, for Hedge the loss is the inner product between the distribution and the loss suffered by each expert, while for EWAF, the loss is between the prediction and the true label, which can be much more complex.

2.3 Learning with Nonlinear Models

In this section we briefly discuss the popular machine learning methods that have been designed for learning nonlinear models. Specifically, we focus on learning with kernel methods and learning with deep neural networks.

2.3.1 Learning with Kernels

The most popular usage of kernels has been through Support Vector Machines (SVMs). Here, we start by introducing SVMs followed by how kernels are used to increase the learning capacity of the models. To learn a classification model $\mathbf{w}$, we can solve a quadratic programming formulation as given below:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{T}\xi_i$$

$$\text{subject to} \quad y_t(<\mathbf{w}, \phi(\mathbf{x}_i)> + b) \geq 1 - \xi_i$$

$$\text{and} \quad \xi_i \geq 0 \quad \forall \quad t$$

$\mathbf{w}$ is the vector of coefficients to be learnt, C is regularization trade-off parameter, ξ represents the slack variables, b is the bias term, and $\phi(\mathbf{x})$ represents the mapping of the instance $\mathbf{x}$ into a higher dimensional space known as the Reproducible

Kernel Hilbert Space. To solve this optimization, we can use the Lagrangian dual function, which gives us:

$$\begin{aligned}&\text{maximize}_\alpha \sum_{t=1}^{T}\alpha_t - \frac{1}{2}\sum_{i=1}^{T}\sum_{j=1}^{T}\alpha_i\alpha_j y_i y_j < \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) > \\ &\text{subject to } \alpha \in \mathbb{R}^+, \quad \sum_{i=1}^{T}\alpha_i y_i = 0, \quad C \geq \alpha_i \geq 0 \quad \forall \quad i\end{aligned}$$

The term $< \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) >$ is the dot product of a high dimensional feature mapping of the instances $\mathbf{x}_i$ and $\mathbf{x}_j$. This dot product is interpreted as the kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) =< \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) >$. Upon solving the optimization, we get $\mathbf{w} = \sum_{i=1}^{T}\alpha_i y_i \phi(\mathbf{x}_i)$, and thus the final discriminant function can be written as:

$$\mathbf{F}(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{i=1}^{T}\alpha_i y_i < \phi(\mathbf{x}_i), \phi(\mathbf{x}) > \quad = \sum_{i=1}^{T}\alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$$

Due to the nature of the optimization, $\alpha_i = 0$ will hold for most of the instances (usually), and the learnt model would be a linear combination of only a few instances with $\alpha_i \neq 0$. These instances are the support vectors of the model.

Due to the high dimensional mapping, the kernels allow SVMs to learn linear patterns in this new space, which corresponds to nonlinear pattern in the original space. And due to the dual formulation, explicit high dimensional mapping need not be performed, and only the dot product in this space needs to be computed, which can be done by working in the original space. This is also known as the kernel trick. The popular kernel types include the Linear Kernels, Polynomial Kernels and Gaussian Kernels. However, the usage of kernels is non-trivial, as it requires a lot of effort to decide which kernel to use (e.g. through extensive validation). Further, different types of kernels may provide complementary information, which could enhance the performance of the overall prediction model. Additionally, many of the data sources are often heterogeneous or multi-modal in nature. Using a single kernel function across all modalities combined may lead to a poor feature representation. Ideally, each modality should have its own kernel. To address these issues, Multiple Kernel Learning evolved as a promising research direction.

Multiple Kernel Learning aims to use a given set of kernel functions, and learn their optimal combination. There are many types of approaches to learn the combination of the kernel functions [43]. The fundamental idea is, that give a predefined pool of M kernel functions $\kappa_m, i = m, \ldots, M$, we need to learn or estimate a super kernel function κ^* which is a combination of all the M kernel functions.

Here we briefly summarize the key ideas of the main categories of multiple kernel learning as categorized by [43] based on the training method used.

Fixed Rules: combinations without parameters, e.g. summation and multiplication of kernels [93, 7].

Heuristic Approaches: Give the kernels some weight based on their individual performance [87, 29].

Optimization Approaches: This is one of the most extensively explored type of learning the kernel combination. The idea is to incorporate the kernel combination learning into the optimization objective. Two popular approaches in this category include: *Target Alignment* and *Structural Risk Minimization.* Target Alignment aims to learn a kernel combination such that the combined kernel function is as close as possible to the ideal kernel function where $\kappa(\mathbf{x}_i, \mathbf{x}_j) = 1$ if $y_i = y_j$, and 0 otherwise [65, 20]. Structural Risk Minimization approaches solve an SVM like objective function with additional parameters to learn a combination of the kernels [65, 110, 95, 96, 124]. The aim is to simultaneously learn the weight vector and the kernel combination. Based on the requirement, the kernel combination maybe linear, convex or conic. It may also be nonlinear.

There are other approaches for learning the multiple kernel combination including **Bayesian Approaches** [42, 19] and **Boosting Approaches** [10, 107, 122].

Despite the increased capacity of models offered by multiple kernels, their training and optimization is extremely expensive, both in terms of memory and computation cost. This affects their usage in large scale datasets, or in a streaming data setting.

2.3.2 Learning with Deep Neural Networks

In recent years, the success of Deep Neural Networks has surged. Here, we give a basic overview of how Deep Neural Networks are able to help in learning nonlinear patterns. The final prediction function of a Deep Neural Networks (DNN), is a set of stacked linear transformations, each followed by a nonlinear activation. Given an input $\mathbf{x} \in \mathbb{R}^d$, the prediction function of DNN with L hidden layers ($\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)}$) is recursively given by:

$$\mathbf{F}(\mathbf{x}) = \text{softmax}(W^{(L+1)}\mathbf{h}^{(L)}) \quad \text{where}$$
$$\mathbf{h}^{(l)} = \sigma(W^{(l)}\mathbf{h}^{(l-1)}) \quad \forall l = 1, \ldots, L$$
$$\mathbf{h}^{(0)} = \mathbf{x}$$

where σ is an activation function, e.g., sigmoid, tanh, ReLU, etc. This equation represents a feedforward step of a neural network. The hidden layers $\mathbf{h}^{(l)}$ are the feature representations learnt during the training procedure. To train a model with such a configuration, we use the cross-entropy loss function denoted by $\mathcal{L}(\mathbf{F}(\mathbf{x}), y)$. We aim to estimate the optimal model parameters W_i for $i = 1, \ldots (L+1)$ by applying Online Gradient Descent (OGD) on this loss function. Following the online learning setting, the update of the model in each iteration by OGD is given by:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \nabla_{W_t^{(l)}} \mathcal{L}(\mathbf{F}(\mathbf{x}_t), y_t) \qquad \forall l = 1, \ldots, L+1$$

where η is the learning rate parameter. Using backpropagation, the chain rule of differentiation is applied to compute the gradient of the loss with respect to $W^{(l)}$ for $l \leq L$. This is the process of Backpropagation.

Here we have given a fundamental overview of the working of Deep Neural

Networks (which are popularly called Multi-Layer Perceptrons). There have been several extensions and enhancements to the simple model, to show performance improvement in several applications, such as Convolutional Neural Networks [63] for computer vision tasks.

2.4 Online Learning with Nonlinear Models

2.4.1 Online Learning with Kernels

Online Learning with kernels was developed to help in online learning with nonlinear models [61]. Consider a typical online learning task. Given a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in |\mathcal{Y}| = \{-1, +1\}$ for binary classification, $y_t \in \{0, 1\}^C$ for multi-class classification with C classes, and $y_t \in \mathbb{R}$ for regression tasks, is the target label assigned to $\mathbf{x}_t$. On each iteration, the loss suffered by the prediction model $\mathbf{f}(\mathbf{x})$ is denoted by $\ell(\mathbf{f}(\mathbf{x}), y)$. The update rule takes the following form while applying gradient descent.

$$\begin{aligned} f_{t+1}(x) &= f_t(x) - \eta \nabla_f \ell(f_t, (\mathbf{x}_t, y_t)) \\ &= f_t(x) + \eta y_t \kappa(\mathbf{x}_t, \mathbf{x}) \end{aligned}$$

Here η is the step size parameter. Note that if a non-zero loss is suffered, the instance gets added to the model as a support vector with $\alpha_t = \eta_t y_t$, and the prediction vector $\mathbf{w}$ takes the form of a linear combination of the support vectors weighted by α_t.

As discussed in the previous section, the online learning with kernels has limitations associated with the usage of a single kernel, and a multiple kernel learning approach would be preferred because: (i) it is difficult to determine the choice of kernel; (ii) different kernels could provide complementary information (iii) they are more suited for heterogeneous data. Consequently Online Multiple Kernel Learning approaches were developed, with the aim of combining the merits of scalable learning through the usage of online learning, and the ability to learn high complexity models through the usage of multiple kernels. Online MKL for structured prediction was developed [81], but required multiple passes through the data, making them unsuitable for the online setting. Other Online Multiple Kernel Learning approaches were developed including OM2 [79], which used Follow the regularized leader and subgradient methods to update the parameters of the objective function and Online Multiple Kernel Learning [58, 49] which sequentially learned independent kernel classifiers, and also used the Hedging [37] algorithm to learn the combination of the independent kernels.

In this dissertation we focus on the limitations of Online Multiple Kernel Learning (inability to handle regression tasks, lack of suitability for time-series and temporal patterns, lack of unsuitability for imbalanced data streams, etc.), and propose solutions to address these challenges.

2.4.2 Online Deep Learning

There is very limited, and almost no work towards addressing Online Deep Learning in literature. In general, optimization of Deep Neural Networks is a very challenging task. There are many challenges that need to be addressed, which include (but not limited to) vanishing gradient, diminishing feature reuse [112], presence of saddle points (and local minima) [18, 28], immense number of parameters to be tuned, internal covariate shift during training [56], difficulties in choosing a good regularizer, choosing hyperparameters, etc. Despite many promising advances recently [89, 56, 46, 112], etc., which are designed to address specific problems for optimizing deep neural networks, most of these existing approaches assume that the DNN models are trained in a batch learning setting.

The most common mode of optimizing DNNs is by gradient descent through backpropagation. A simple solution to using DNNs in the online setting is to apply backpropagation in every online learning iteration. Unfortunately, using such a model for an online learning (i.e. Online Backpropagation) task faces several issues with convergence. Most notably: (i) For such models, a fixed depth of the neural network has to be decided a priori, and this cannot be changed during the training process. This is problematic, as determining the depth is a difficult task. Moreover, in an online setting, different depths may be suitable for a different number of instances to be processed, e.g. because of convergence reasons, shallow networks maybe preferred for small number of instances, and deeper networks for large number of instances; (ii) vanishing gradient is well noted problem that slows down learning. This is even more critical in an online setting, where the model needs to make predictions and learn simultaneously; (iii) diminishing feature reuse, according to which many useful features are lost in the feedforward stage of the prediction. This again is very critical for online learning, where it is imperative to quickly find the important features, so as to not suffer from poor performance for the initial training instances.

There have been attempts at making deep learning models compatible with online learning [132, 70]. However, they operate via a sliding window approach where a (mini)batch training stage is always involved, making them unsuitable for a streaming data setting.

2.5 Summary

In this chapter we presented a brief survey of the related studies for this dissertation. In particular, we first reviewed the literature in Online Learning. While Online Learning has many branches, we focussed on First-order Online Learning, Second-order Online Learning and Online Learning with Expert Advice. Then we moved on to reviewing the literature in learning nonlinear models. We specifically discussed learning with kernels, multiple kernel learning and learning with deep neural networks. This was followed by reviewing the work at the intersection of these categories. We reviewed online learning with kernels, online multiple kernel learning and online deep learning.

Chapter 3

Online Multiple Kernel Regression

In this chapter we present our proposed Online Multiple Kernel Regression Algorithms, which learn a kernel-based regressor in an online fashion, and dynamically explore a pool of diverse kernels to avoid suffering from a single fixed poor kernel so as to remedy the drawback of manual/heuristic kernel selection. We propose large-scale OMKR algorithms which rely on kernel approximation techniques. We also evaluate the kernel combinations at prediction level and the representation level. Finally, we also demonstrate its application to online learning for time-series models.

3.1 Introduction

Kernel methods have been extensively studied for regression tasks and found successes in many real-world applications [109, 106]. 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 [106], 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 there is any new training data, 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 [120] and NORMA learning [61], 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 [36] that can track the best kernel regressor, and *Online Gradient Descent*(OGD)[134] 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 [78], methods that perform online learning with kernels suffer from an unbounded number of support vectors. 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 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 online multiple kernel regression, which are based on kernel approximation techniques [121, 94]. 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 nonlinear time-series prediction.

The rest of the chapter is organized as follows: In Section 3.2, we review the related work, specifically focussing on contributions in literature in the domain of online learning and multiple kernel learning. In Section 3.3, we present the main framework for Online Multiple Kernel Regression. This is followed by the presentation OMKR algorithms for large scale learning in Section 3.4. We discuss the application time-series prediction in Section 3.5. In Section 3.6 we present our detailed experimental results, and finally conclude in Section 3.7.

3.2 Related Work

We review some of related work in online learning and kernel learning in the context of OMKR.

3.2.1 Online Learning

Online learning refers to a class of scalable algorithms that learn sequentially from streamlining data [105, 14, 51], and they have been extensively explored indifferent contexts and applications [21, 125, 88]. 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. Many online algorithms are designed to learn linear models [21]. A closely related area is online prediction with expert advice [36, 75, 116], where predictions from multiple experts are weighted and update in every online iteration. One of the most well-known algorithms is the Hedge Algorithm [36], which was a direct generalization of Weighted Majority Algorithm [75].

Many online algorithms have been proposed for extending kernel methods in an online setting, in which several techniques have have been proposed for online kernel regression, such as Naive Online R_{reg} Minimization Algorithm (NORMA) [61], Online Passive Aggressive Regression [21], Sparse Implicit Online Learning with Kernels (ILK and SILK) [104] and Primal Online Algorithm (PRIONA) [12]. While these methods provide with promising directions, they suffer from problems of kernel selection, and they are not able to utilize complementary information from multiple kernels. Another major problem with existing online kernel methods is the curse of kernelization, where the number of support vectors is unbounded. In the online setting, every instance that suffers a non zero loss becomes a support vector, which results in updating the prediction function such that every new prediction requires a kernel function computation with all the support vectors in memory. This is even more problematic in the case of using multiple kernels, where the kernel function computation has to be done for all support vectors, and for all the kernels. To address the scalability concerns of kernel-based online learning, many studies focus on the budget issue [22, 13]. These speed up the algorithms by bounding the number of support vectors. Some well-known example algorithms include Forgetron [30], Projectron [91], and the Bounded Online Gradient Descent (BOGD) [131]. Recently, kernel approximation strategies have been proposed to address the scalability concerns [78].

3.2.2 Kernel Learning

Kernel methods have gained popularity due to their ability to learn nonlinear patterns in the data [103]. Several Kernel methods have been applied to regression tasks [109]. 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 [59], idealized kernel learning [64], graph-based spectral kernel learning [11, 50], and non-parametric kernel learning [48, 133]. These methods often follow a batch (and transductive) learning setting and thus are difficult to be applied in an online learning scenario.

Another prevalent kernel learning technique is Multiple Kernel Learning (MKL) [65], which aims to find the optimal combination of multiple kernels. Unlike most existing MKL techniques that are batch learning [65, 110, 43], our work focuses on online regression tasks, and is related to existing online MKL studies that focus on classification tasks [58, 49] and that addresses structured prediction [81]. Existing studies in Online Learning with Multiple Kernels have several limitations, including computational problems arising from unbounded number of support vectors, unsuitability to concept drift and time series applications. Further, the kernel combination strategies are limited.

3.3 Online Multiple Kernel Regression

3.3.1 Problem Setting

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

Consider a set of training examples $\{(\mathbf{x}_t, y_t), t = 1, \dots, T\}$ where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ and a collection of m kernel functions $\mathcal{K} = \{\kappa_i : \ \chi \times \chi \rightarrow \mathbb{R}, i = 1, \dots, m\}$. Multiple Kernel Learning aims to learn a kernel-based prediction model by identifying the best linear combination of the m kernels, that is, a weighted combination $\theta = (\theta_1, \dots, \theta_m)$. The learning task can be cast into the following optimization [65]:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_K(\theta)} \frac{1}{2} |f|^2_{\mathcal{H}_{K(\theta)}} + C \sum_{t=1}^{T} \ell(f(\mathbf{x}_t), y_t) \tag{3.1}$$

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 [110, 124, 43]. 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: (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.

3.3.2 OMKR Framework

To overcome the limitations of MKL for a regression task, we propose a new scheme of Online Multiple Kernel Regression (OMKR) by applying the emerging online multiple kernel learning principle [49] 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: (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.

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 [120] and NORMA [61], which follows the same principle of Online Gradient Descent (OGD) [134] 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:

$$\mathcal{L} = \Sigma_{t=1}^{T} \ell(f_t(x_t), y_t) \triangleq \Sigma_{t=1}^{T} \mathcal{L}_t(f_t) \tag{3.2}$$

where $f_t(x_t)$ is the prediction made by a kernel regressor on the t-th 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 [134], we have the following online update rule given a data instance $(\mathbf{x}_t, y_t)$:

$$f_{t+1} \leftarrow f_t - \eta_t \nabla \mathcal{L}_t(f_t) \tag{3.3}$$

where $\eta_t > 0$ is a learning rate parameter that can be either a small constant $\eta_t = \eta$ used in Widrow-Hoff [120] 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

$$f_{t+1}(\cdot) \leftarrow f_t(\cdot) - \eta_t(f_t(\mathbf{x}_t) - y_t)\kappa(\mathbf{x}_t, \cdot). \tag{3.4}$$

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 support vectors (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 [61], which replaces $\mathcal{L}_t(f_t)$ by the following regularized loss:

$$\mathcal{L}_t(f_t) = \frac{\lambda}{2}||f||^2_{\mathcal{H}_\kappa} + \ell(f_t(\mathbf{x}_t), y_t) \tag{3.5}$$

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

$$f_{t+1} \leftarrow (1 - \eta_t\lambda)f_t - \eta_t\nabla\ell(f_t(\mathbf{x}_t), y_t) \tag{3.6}$$

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:

$$\ell_t(f_t, \mathbf{x}_t) = \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t \tag{3.7}$$

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:

$$f_{t+1} \leftarrow \begin{cases} (1 - \eta_t\lambda)f_t + \eta_t * sgn(d)\kappa(\mathbf{x}_t, \cdot) & \text{if } |d| > \epsilon_t \\ (1 - \eta_t\lambda)f_t & \text{otherwise} \end{cases} \tag{3.8}$$

$$\epsilon_{t+1} \leftarrow \begin{cases} \epsilon_t + (1 - \nu)\eta_t & \text{if } |d| > \epsilon_t \\ \epsilon_t - \eta_t\nu & \text{otherwise} \end{cases} \tag{3.9}$$

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 [103]:

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

where the α_i coefficients are computed based on the updating rules in (3.4) or (3.8). When $\alpha_i \neq 0$, the i-th instance is often called as a Support Vector (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 support vectors. By contrast, when using the ϵ-insensitive loss, whenever the difference between the prediction on the i-th 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.

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:

$$F_t(\mathbf{x}) = \sum_{i=1}^{m} w_t^i f_t^i(\mathbf{x}) \tag{3.10}$$

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 [36] and the OGD algorithm [134]. 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 [116, 14]. 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:

$$w_{t+1}^i = w_t^i \beta^{\ell_t^i}, i = 1, \ldots, m \tag{3.11}$$

where $\beta \in (0, 1)$ is a discounting (learning rate) parameter, and ℓ_t^i denotes the loss suffered by the i-th 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 9. In the algorithm, we can update each kernel regressor f_{t+1}^i by adopting either the Widrow-Hoff learning in (3.4) or NORMA in (3.8).

Algorithm 3 Deterministic OMKR (Hedge)

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \to i = 1, \ldots, m$
  - 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 $f_{t+1}^i$ = Eq. (3.4) OR (3.8)
      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 $W_t = \sum_{i=1}^{m} w_t^i,\ i = 1, \ldots, m$
  **end for**
```

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 online gradient descent (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

$$\mathbf{w}_{t+1} \leftarrow \arg\min_{\mathbf{w}} \ell(\mathbf{w}^\top \mathbf{f}_t(\mathbf{x}_t), y_t) \triangleq (\mathbf{w}^\top \mathbf{f}_t(\mathbf{x}_t) - y_t)^2 \tag{3.12}$$

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

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_w(\hat{y}_t - y_t)\mathbf{f}_t(\mathbf{x}_t) \tag{3.13}$$

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 4. Like in OMKR(Hedge), we can also update each kernel regressor by either Widrow-Hoff in (3.4) or NORMA in (3.8).

Remark. In online MKL work related to classification [58, 49], 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

Algorithm 4 Deterministic OMKR (OGD)

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \to i = 1,\dots,m$
  - Learning rate parameter: $\eta_w$
  - Step size parameter for each kernel: $\eta$
  - Regression parameters: $\lambda$ and $\nu$ for OMKR(NORMA)
Initialization: $\mathrm{f}_1 = \mathbf{0}$, $\mathbf{w}_1 = \mathbf{0}$
  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 $f_{t+1}^i =$ Eq. (3.4) OR (3.8)
    end for
    Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_w(\hat{y}_t - y_t) f_t(x_t)$
  end for
```

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 3.6.

3.3.3 Theoretical Analysis

Without loss of generality we assume that $\forall\ i,\ \forall\ t,\ \kappa^i(x_t \cdot x_t) \leq 1$, and $\ell_t(f_t^i(\mathbf{x}_t), y_t) \leq 1$.

Theorem 1. *After receiving a sequence of T instances, the cumulative loss suffered* *by OMKR (Hedge) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \frac{\ln(\frac{1}{\beta})}{1-\beta} \min_{1\leq i\leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\ln(m)}{1-\beta} \tag{3.14}$$

where

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f\in\mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^{T}(f(x_t)-y_t)^2}{1-\eta} + \frac{||f||^2}{\eta}\right] \tag{3.15}$$

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:

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

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.

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

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

$$\begin{aligned}\phi_{t+1}^i - \phi_t^i &\leq \eta^2 \ell_t^{i2} - 2\eta\ell_t^{i2} + \eta(1-\eta)\ell_t^{i2} + \frac{\eta}{1-\eta}\ell_t^{*2} \\ &= -\eta\ell_t^{i2} + \frac{\eta}{1-\eta}\ell_t^{*2}\end{aligned} \tag{3.16}$$

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

$$\mathcal{L}_{WH}^i \leq \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^T (f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta} \right] \tag{3.17}$$

where $\mathcal{L}_{WH}^i$ is the cumulative loss suffered by the regression function learnt by the the Widrow-Hoff Algorithm in the RKHS by the i^{th} kernel. As number of instances T grows large, the average loss per instance

Plugging this result into the Hedge Algorithm gives us the bound. The choice of β maybe overestimated because of the assumption that the loss suffered by the algorithm is T. $\square$

Theorem 2. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (OGD) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \min_{1 \leq i \leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\eta G T}{2} + \frac{C}{2\eta} \tag{3.18}$$

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*

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f \in \mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^T (f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta} \right] \tag{3.19}$$

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 Online Gradient Descent and the Widrow-Hoff Regression. Online Gradient Descent provides a sublinear 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 $\ell(\mathbf{w}) < \ell(\mathbf{w}^*) \quad \forall \mathbf{w}$.

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 t^{th} iteration for the i^{th} kernel is denoted as:

$$\ell_t(f_t^i) = \frac{\lambda}{2}||f||^2_{\mathcal{H}_\kappa} + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t \tag{3.20}$$

There are two sets of parameters to be updated: f and ϵ. The loss function is convex in both these parameters. Since the update rule takes the form of Online Gradient Descent [134], both f and ϵ are learnt via the same update rule. Thus we incorporate ϵ into the prediction function f.

Lemma 1. *After receiving a sequence of T instances, the cumulative loss suffered by NORMA regression is bounded as*

$$\mathcal{L}_{NORMA} \leq \min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^{T} \Big(\frac{\lambda}{2}||f||^2_{\mathcal{H}_\kappa} + \max(0, |y_t - f_t(\mathbf{x}_t)| - \epsilon_t) + \nu\epsilon_t\Big) + \sqrt{C_2 G_2 T} \tag{3.21}$$

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

$$\mathcal{L}_{OMKR} \leq \frac{\ln(\frac{1}{\beta})}{1-\beta}\min_{1\leq i\leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\ln(m)}{1-\beta} \tag{3.22}$$

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

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

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

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

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

Theorem 4. *After receiving a sequence of T instances, the cumulative loss suffered by OMKR (OGD) using the Widrow-Hoff Algorithm is bounded as*

$$\mathcal{L}_{OMKR} \leq \min_{1\leq i\leq m} F(\kappa_i, \ell, \mathcal{D}) + \frac{\eta G T}{2} + \frac{F}{2\eta} \tag{3.23}$$

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

$$F(\kappa_i, \ell, \mathcal{D}) = \min_{f\in\mathcal{H}_\kappa} \left[\frac{\Sigma_{t=1}^{T}(f(x_t) - y_t)^2}{1-\eta} + \frac{||f||^2}{\eta}\right] \tag{3.24}$$

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

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

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

3.3.4 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 support vectors in the dataset (and when squared loss is used, all instances invariably become support vectors). 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 support vectors; and (ii) functional approximation schemes to approximate the kernel functions without explicitly computing the kernel function with all the support vectors.

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 support vector 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 [39], in which case many of the old support vectors 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.

Stochastic Update for OMKR

An update to a kernel regressor involves adding a new support vector. 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.

$$q_t^i = \frac{|w_t^i|}{\max_{1 \leq j \leq m} |w_t^j|} \tag{3.25}$$

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:

$$p_t^i = (1 - \delta) q_t^i + \frac{\delta}{m} \tag{3.26}$$

Here δ is a small value. A similar idea was used in [4], 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 5.

Budget OMKR

As the number of support vectors 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

Algorithm 5 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: $\boldsymbol{f_1} = \mathbf{0}$, $\mathbf{w_1} = \frac{\mathbf{1}}{\boldsymbol{m}}\mathbf{1}$
  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 $f_{t+1}^i =$ Eq. (3.3) OR (3.6)
      end if
    end for
    Update $\mathbf{w}_{t+1}$ based on Hedge or OGD
  end for
```

budget τ is specified, and the number of support vectors is not allowed to exceed this number. These are broadly classified into three categories: Removal, Projection and Merging. Removal refers to replacing old support vectors with new ones when the budget is exceeded, based on certain criteria. Projection

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 support vectors that explain the data. This is also particularly helpful in the case of NORMA, where in each iteration, the old support vectors get reduced by a factor of $(1 - \eta\lambda)$ due to the regularization term. As t grows, the α values of the old support vectors get reduced to almost zero. Such support vectors can be ignored without any significant impact to the prediction. Therefore, we propose a parameter τ which restricts the total number of support vectors that are allowed to be stored by each regressor. The older support vectors are deleted.

3.4 Large-Scale Online Multiple Kernel Regression via Functional Approximation

The proposed OMKR scheme follows the usage of traditional Online Learning with kernels [61], 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 [36], and each kernel predictor is also updated. This scheme suffers from two drawbacks:

- They suffer from the curse of kernelization, which means that the number of support vectors is unbounded. This adds a significant computational burden for the algorithm. In particular poor performing kernels have a tendency to acquire a lot of support vectors, 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 support vectors, which can be noisy or insufficient to make accurate predictions.
- The design of the original batch Multiple Kernel Learning 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 [121, 94].

3.4.1 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:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j) \tag{3.27}$$

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

$$f_{t+1}(\mathbf{x}) = \Sigma_{i=1}^t \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \Sigma_{i=1}^t \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}_\kappa^\top \mathbf{z}(\mathbf{x}) \tag{3.28}$$

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

Applying online gradient descent [134], 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:

$$\mathbf{z}_{\kappa_i}(\mathbf{x}) \text{ for } i = 1, \ldots, m \tag{3.29}$$

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:

$$f^i(\mathbf{x}) = \mathbf{w}_{\kappa_i}^\top \mathbf{x} \tag{3.30}$$

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, etc. For this class of kernels, random Fourier Features can be obtained to approximate the kernel function [94, 78]. Applying inverse Fourier transform to a shift-invariant kernel function, we get:

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u} \tag{3.31}$$

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, the kernel function can be expressed as an expectation of the random variable $\mathbf{u}$ [94] 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 $\mathbf{u}_1, \dots, \mathbf{u}_D$ and obtain the new feature representation as:

$$\mathbf{z}(\mathbf{x}) \;=\; [\sin(\mathbf{u}_1^\top \mathbf{x}), \cos(\mathbf{u}_1^\top \mathbf{x}), \dots, \sin(\mathbf{u}_D^\top \mathbf{x}), \cos(\mathbf{u}_D^\top \mathbf{x})]^\top \qquad (3.32)$$

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 which 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 $\mathbf{K} = \mathcal{V}\mathbf{D}\mathcal{V}^\top$, where the columns in $\mathcal{V}$ are orthogonal, and $\mathbf{D}$ is a diagonal matrix $\mathbf{D} = \text{diag}(\sigma_1, \dots, \sigma_r))$. For $k < r, \mathbf{K}_k = \Sigma_{i=1}^{k} \sigma_i V_i V_i^\top = \mathcal{V}_k \mathbf{D}_k \mathcal{V}_k^\top$ 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:

$$\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^{+}\mathbf{C}^\top \approx \mathbf{K} \qquad (3.33)$$

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 support vectors, we consider a budget of B, where we are given a maximum of B support vectors, where $B \ll T$. From equation (3.33), we can see that the kernel value of two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ can be approximated as:

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

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

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

3.4.2 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 [36]. Following the approach described in Section 3.3, the OMKR process can be split into 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 support vectors are obtained. This is followed by using these B support vectors 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 Online Gradient Descent [134].

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 [134], we get the following update rule:

$$\mathbf{w}_{\kappa_t+1} \leftarrow \mathbf{w}_{\kappa_t} - \eta_t(\hat{y}_t - y_t) f_t(\mathbf{x}_t) \tag{3.34}$$

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 [61] 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 [36], and following the update rule as described in Section 3.3.2, we get:

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

This entire scheme is outlined in Algorithm 7.

Algorithm 6 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: $\boldsymbol{f_1} = \mathbf{0}$, $\mathbf{w_1} = \frac{\mathbf{1}}{\boldsymbol{m}}\mathbf{1}$
  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
```

3.4.3 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 multiple kernel learning procedure into two steps: learning the kernel regressors independently, followed by combining the multiple predictions using Hedging. However, following the principle of the original batch MKL optimization:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_K(\theta)} \frac{1}{2}|f|^2_{\mathcal{H}_{K(\theta)}} + C \sum_{t=1}^{T} \ell(f(\mathbf{x}_t), y_t) \tag{3.35}$$

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

$$\mathbf{Z}(\mathbf{x}) = [\mathbf{z}_1(\mathbf{x}), \ldots, \mathbf{z}_m(\mathbf{x})]^\top \tag{3.36}$$

$\mathbf{Z}(\mathbf{x})$ 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 Online Gradient Descent 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:

$$F(\mathbf{x}) = \mathbf{w}^\top \mathbf{Z}(\mathbf{x}) \tag{3.37}$$

Using the squared loss function $\ell(F_t(\mathbf{x}_t), y_t) = (F_t(\mathbf{x}_t) - y_t)^2$, by applying Online Gradient Descent[134], we get the update rule as:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t(\hat{y}_t - y_t)F_t(\mathbf{x}_t) \tag{3.38}$$

Algorithm 7 OMKR(OGD) via Functional Approximation

```
INPUT:
  - Kernels: $\kappa(\cdot,\cdot) : \chi \times \chi \rightarrow i = 1, \ldots, m$
  - Update Parameter: $\eta$ for OGD
Initialization: $\boldsymbol{f_1} = \mathbf{0}$, $\mathbf{w_1} = \frac{\mathbf{1}}{\boldsymbol{m}}\mathbf{1}$
  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
```

3.5 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 [115, 102]. For our problem setting, the aim is to perform online learning for time series prediction [2]. We first introduce the popular time series prediction models: Autoregressive (AR) models and Autoregressive Moving Average models (ARMA). Then we present how to apply the OMKR framework to online learn for time-series prediction. This is followed by discussing the capturing of nonlinearity for time-series prediction through kernelized models.

3.5.1 Time Series Models

Autoregressive(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 autoregressive process of order p, i.e., y_t is described by a noisy linear combination of $[y_{t-1} y_{t-2} \dots y_{t-p}]$:

$$y_t = c + \Sigma_{i=1}^{p} \zeta_i y_{t-i} + \epsilon_t \tag{3.39}$$

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., $Y_{t-1}^{p} = \{y_{t-1}, y_{t-2}, \dots, y_{t-p}\}$, and thus the equation simplifies to:

$$y_t = c + \zeta^p Y_{t-1}^{p} + \epsilon_t \tag{3.40}$$

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

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

- 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:

$$\mathcal{K} = \Big\{\kappa(Y_t^{p_1}, \cdot), \dots, \kappa(Y_t^{p_k}, \cdot)\Big\} \tag{3.41}$$

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 sublinear regret with respect to the best performing order p. Apart from the obvious benefit of not having to manually select the 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 *Autoregressive Moving Average*(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:

$$y_t = c + \Sigma_{i=1}^{p} \zeta_i y_{t-i} + \Sigma_{i=1}^{q} \xi_i \epsilon_{t-j} + \epsilon_t \tag{3.42}$$

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, [2] showed that an $ARMA(p,q)$ model could be learned online by learning an $AR(m+p)$ model, where m was set as $m = q.\log_{1-\epsilon}\Big((TLM_{max})^{-1}\Big)$. m controls the level of approximation. Under certain assumptions (discussed in [2]), Online Learning of an $AR(m+p)$ model could achieve sublinear regret compared to the best $ARMA(p,q)$ model. Thus, applying OMKR, to Autoregressive models can also obtain sublinear regret with respect to the best $ARMA(p,q)$ model.

This approach can further be extended to *Autoregressive Integrated Moving Average*(ARIMA) time-series modeling. While ARMA is designed for stationary settings, ARIMA is used for modeling nonstationary 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:

$$\nabla^d y_t = c + \Sigma_{i=1}^{p} \zeta_i \nabla^d y_{t-i} + \Sigma_{i=1}^{q} \xi_i \epsilon_{t-j} + \epsilon_t \tag{3.43}$$

Following [76], 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.

3.5.2 Online Nonlinear 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 nonlinear time series modeling. Consider the AR model described in Section 3.5, where:

$$y_t = c + \zeta^p Y_{t-1}^p + \epsilon_t \tag{3.44}$$

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

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

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 $p \in [p_1, p_2, \ldots, p_k]$, and m kinds of diverse kernels for each p. This gives us the following pool of mk kernel functions:

$$\mathcal{K} = \Big\{\kappa^i(Y_t^{p_1}, \cdot), \ldots, \kappa^i(Y_t^{p_k}, \cdot) \text{ for } i = 1, \ldots, m.\Big\} \tag{3.45}$$

The above can now be directly plugged into the OMKR framework for solving time series prediction tasks. In comparison to existing kernel methods for time 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.

3.6 Experiments

3.6.1 Evaluation of OMKR on stationary datasets and nonlinear time-series

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 ten 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 3.1.

Table 3.1: 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

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 uni-

[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`

variate 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.

Kernels

We evaluate the performance of OMKR by using a pool of 24 predefined kernels. These include 4 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., $\hat{y_t} = \max(0, \min(1, \hat{y_t}))$.

Baselines and Experimental Setting

We compare the algorithms based on Mean Squared Error (MSE), time taken, and the weight distribution. The algorithms compared are - (i) *Regression(V)*: Best Kernel by validation; (ii) *Regression(H)*: Best Kernel in hindsight; (iii) *Uniform OMKR*: Uniform weight distribution over kernels (to see if this can eliminate the impact of a poor kernel choice); (iv) *Deterministic OMKR (Hedge)*; and (v) *Deterministic OMKR (OGD)*. We then analyze the performance of efficiency enhancing variants of OMKR and study the tradeoff between accuracy and efficiency.

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 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 $\tau = 500$ support vectors. In stochastic OMKR, the smoothing parameter δ was set to 0.05 in all cases.

Results and Discussion

The detailed results of single kernel regression against OMKR can be seen in Table 3.2. Columns Reg(V) and Reg(H) represent single kernel regression by validation and in hindsight. Columns Uniform, Hedge, OGD represent OMKR with uniform weights, weight updated by Hedge, and weight updated by OGD respectively.

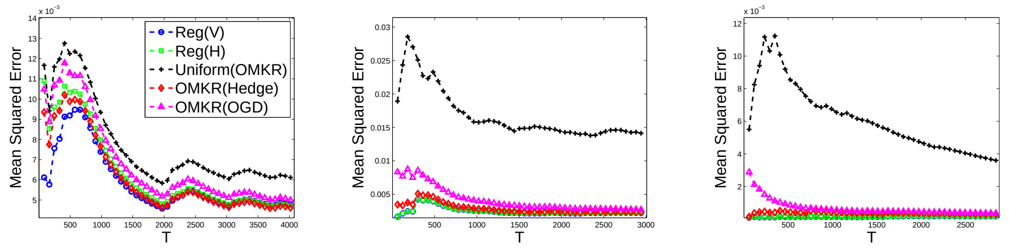
With almost no exception both our proposed methods OMKR (Hedge and OGD) outperform Reg(V) very significantly, at times achieving as low as 1% of error of Reg(V). We should note that in a real world setting, it is hard to choose a better kernel for unseen data than by a validation method. Reg(H) is the 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

Table 3.2: Single Kernel Regression vs Multiple Kernel Regression:
Each field is the ratio $\frac{MSE_{algorithm}}{MSE_{Reg(V)}}$. 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.

	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

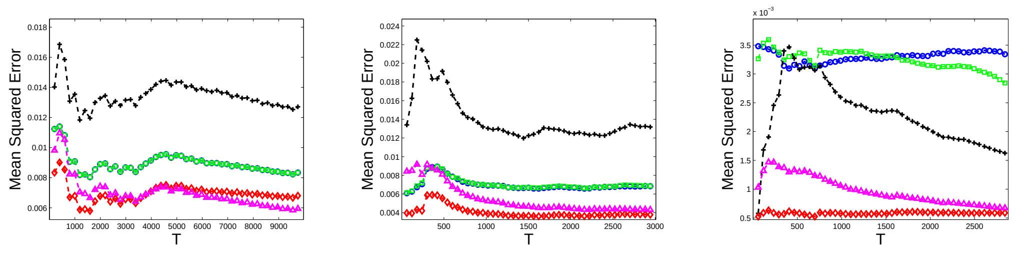
represents the data, which is often the case. However, since Hedge only offers a linear combination of the best kernel(s), we expect the optimal linear combination determined by OGD to outperform Hedge. This does not happen if the the data is not large enough for OGD to converge to optimal linear combination, 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 3.1 and 3.2. 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 performance. Referring back to Table 3.2, 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 3.3. 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.



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

Figure 3.1: Cumulative Mean Squared Error with time (when Widrow-Hoff is used for regression):

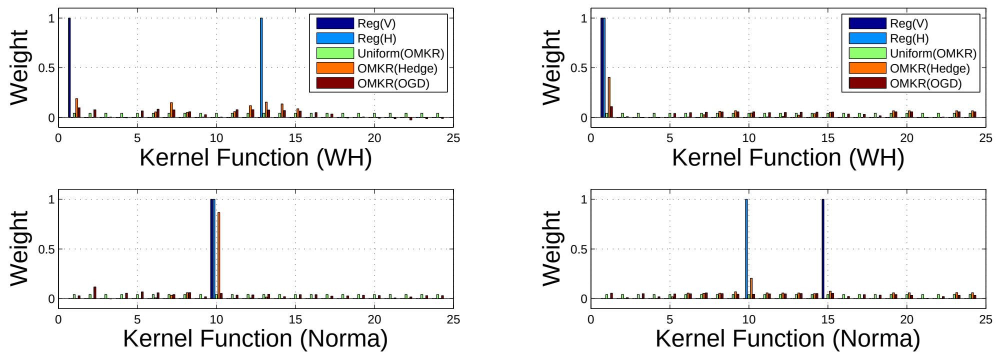
All results are displayed for data after the validation stage during which the parameters were determined.



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

Figure 3.2: 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

Figure 3.3: Weight distribution attained by all algorithms

Table 3.3: OMKR (Hedge) vs Stochastic and Budget Strategies:
Here again, the results of regression datasets are averaged over 10 random permutations. All times are in seconds.

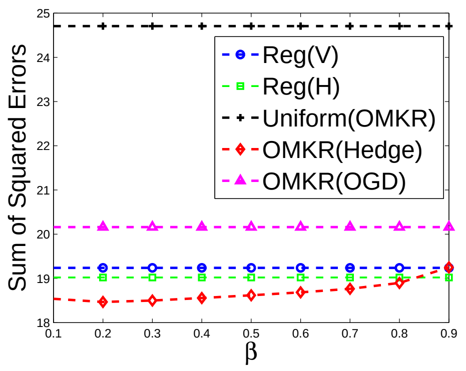
	Widrow Hoff						**Norma**					
ID	**Determinisitc**		**Stochastic**		**Budget**		**Det OMKR**		**Stochastic**		**Budget**	
	MSE	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**TIME**
Regression Datasets												
D1	**0.0075**	348	0.0079	39	0.0096	74	0.0121	220	**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

Table 3.4: OMKR (OGD) vs Stochastic and Budget Strategies:
Here again, the results of regression datasets are averaged over 10 random permutations. All times are in seconds.

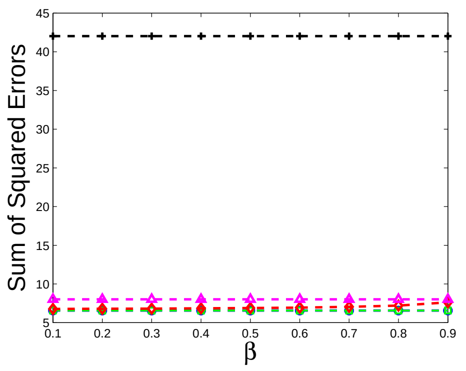
	Widrow Hoff						**Norma**					
ID	**Determinisitc**		**Stochastic**		**Budget**		**Det OMKR**		**Stochastic**		**Budget**	
	MSE	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**Time**	**MSE**	**TIME**
Regression Datasets												
D1	**0.0082**	348	0.0086	43	0.0097	76	0.0105	220	**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

Evaluation of efficiency enhancers

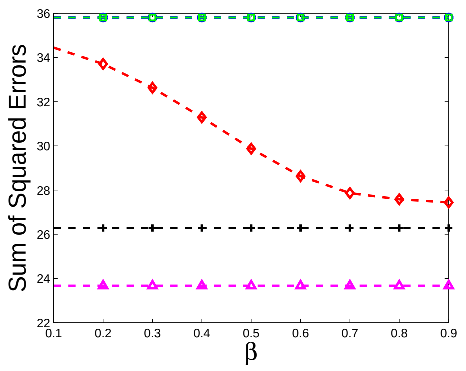
The MSE and the time taken by Deterministic, Stochastic and Budget OMKR are detailed in Tables 3.3 and 3.4. Clearly the time taken by both stochastic and budget techniques is significantly lower than Deterministic OMKR. Despite this, in most cases, the efficiency enhancers give comparable MSEs with respect 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.



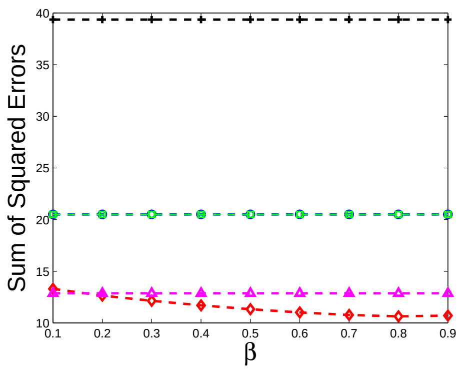
(a) D1 (WH)



(b) D3 (WH)



(c) D1 (Norma)



(d) D3 (Norma)

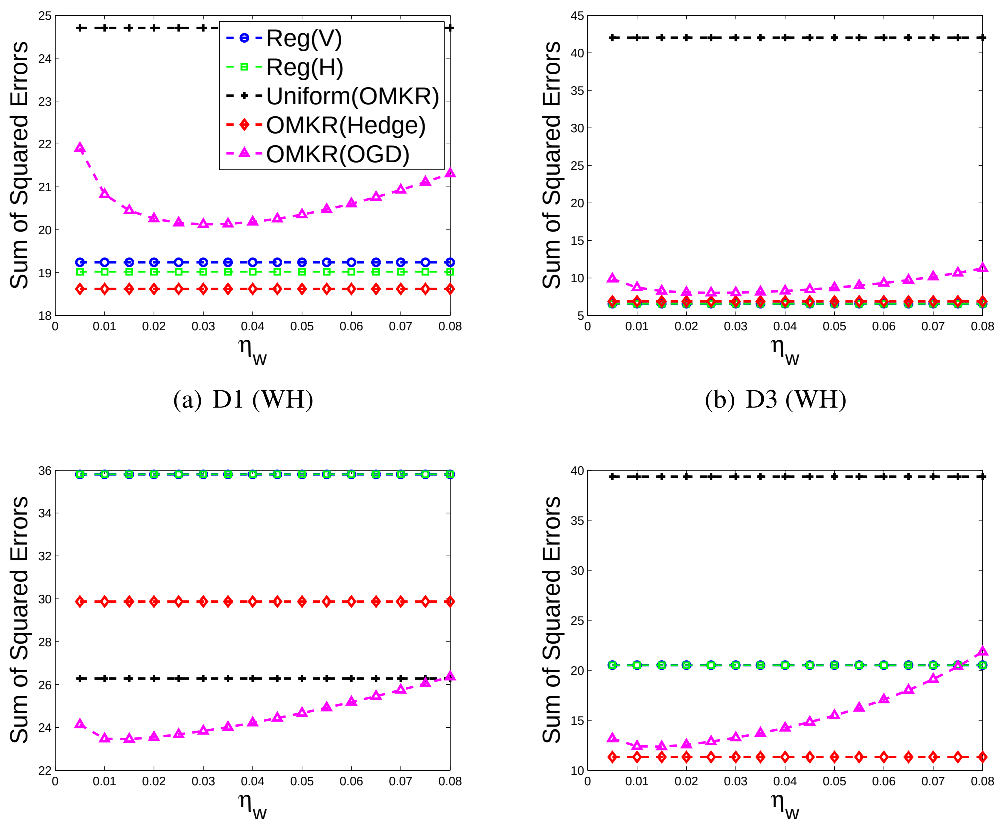
Figure 3.4: Sensitivity of OMKR(Hedge) to discount rate parameterβ: We vary β keeping the performance of all other algorithms fixed.

Sensitivity to weight update parameters β and η_w

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 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.

3.6.2 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



(a) D1 (WH)

(b) D3 (WH)

(c) D1 (Norma)

(d) D3 (Norma)

Figure 3.5: Sensitivity of OMKR(OGD) to learning rate η_w:
We vary η_w keeping the performance of all other algorithms fixed.

between combination of kernels at the prediction level vs the feature representation level (something that was non trivial to achieve in previous approaches).

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 4 datasets, whose details can be seen in Table 3.5. 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

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

L4 is about predicting the buzz on Twitter.

Table 3.5: 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

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.

Results and Discussion

The results of the analysis of Large Scale OMKR algorithms can be seen in Table 3.6. 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 online multiple kernel regression.

Table 3.6: Large Scale Online Multiple Kernel Regression:
The numbers represent the final cumulative error obtained by the algorithms. All results are averaged over multiple permutations. The best performances are in bold.

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

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 3.6 for Fourier feature based OMKR, and in Figure 3.7 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 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.

3.6.3 Evaluation of OMKR for Application to Time-Series Prediction

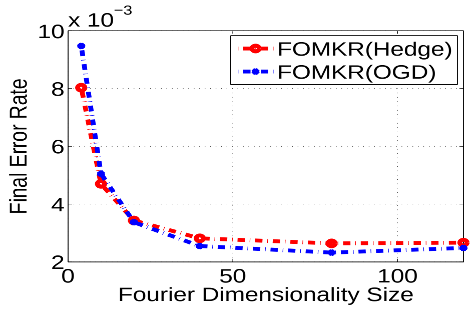
In this section we evaluate the performance of OMKR when applied to time-series prediction. Specifically we focus on the problem of identifying the optimal window size of historical data while performing online learning for time-series prediction. For experiments on Online Learning of non-linear time-series prediction, see Section 3.6.1.

Datasets

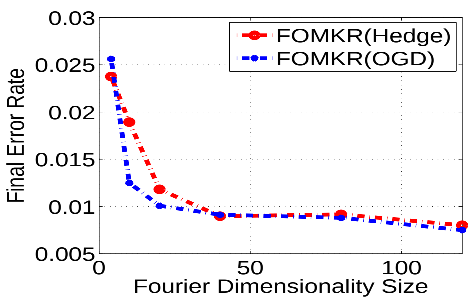
We follow the experiments in [76], and consider 4 synthetic time series settings, and one real world time-series data. After obtaining the sequence, the values are normalized to lie between [0,1]. Consider an ARIMA model given as:

$$\nabla^d y_t = c + \Sigma_{i=1}^{p} \zeta_i \nabla^d y_{t-i} + \Sigma_{i=1}^{q} \xi_i \epsilon_{t-j} + \epsilon_t \tag{3.46}$$

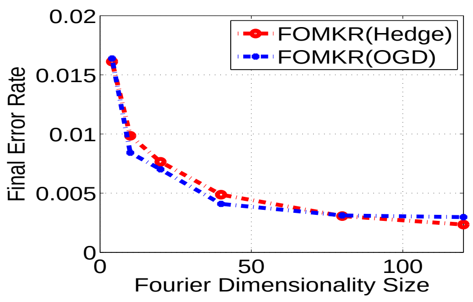
The first sequence is a stationary time-series, S1 is generated from an $ARIMA(p, d, q)$ model, with $d = 1$, $\zeta = [0.6, -0.5, 0.4, -0.4, 0.3]$ and $\xi = [0.3, -0.2]$. The noise terms are uniformly distributed as $\mathcal{N}(0, 0.3^2)$. The second sequence is a non-stationary time-series model 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 $d = 1$, $\zeta = [-0.4, -0.5, 0.4, 0.4, 0.1]$



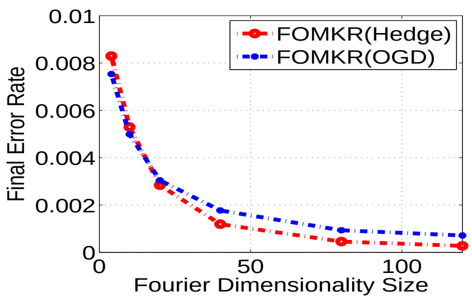
(a) D3



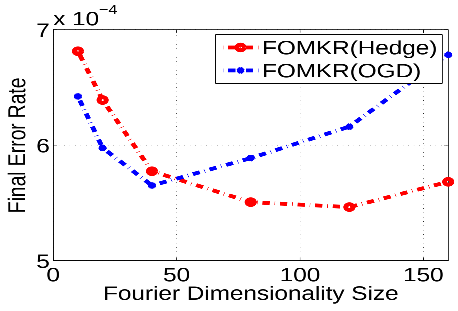
(b) D5



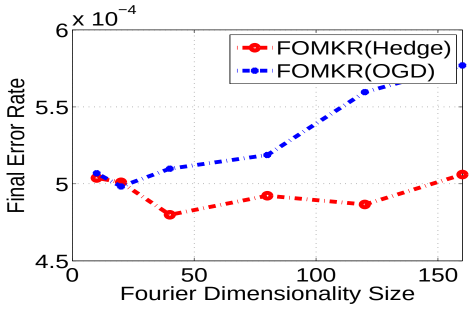
(c) D6



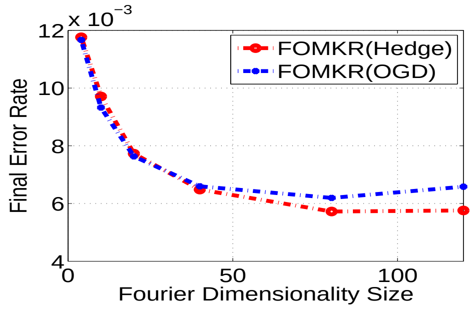
(d) D7



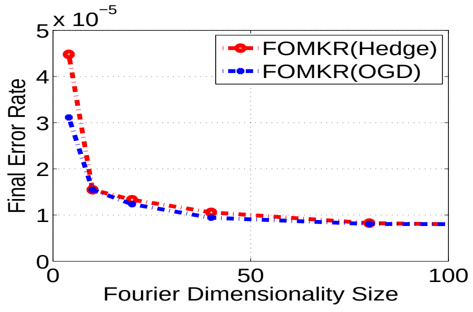
(e) L1



(f) L2

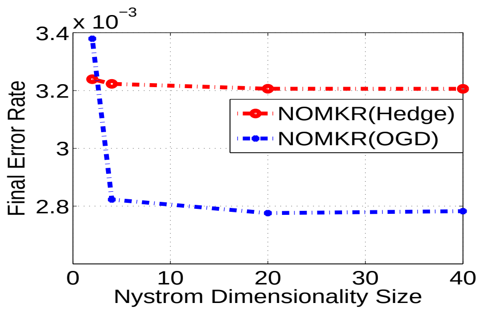


(g) L3

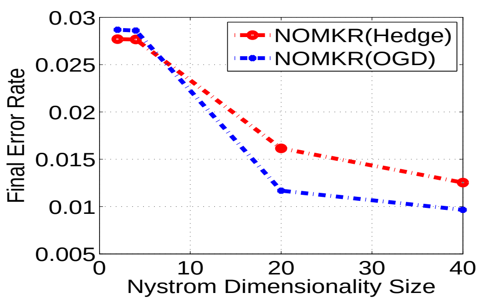


(h) L4

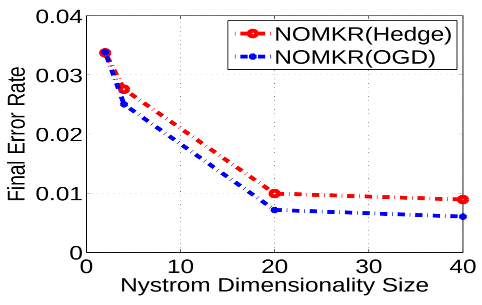
Figure 3.6: Performance of FOMKR(Hedge) and FOMKR(OGD) with varying number of approximated features. The Fourier Dimensionality size refers to the number of features obtained per kernels. The total number of features is m times this number. For example, 40 features per kernel corresponds to 520 features for FOMKR (where $m = 13$).



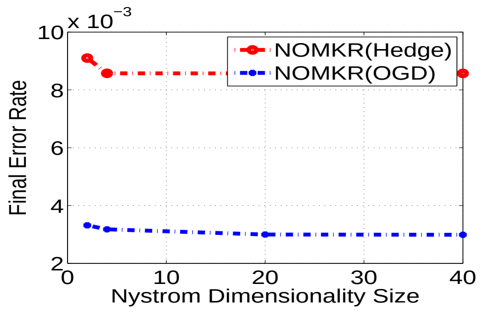
(a) D3



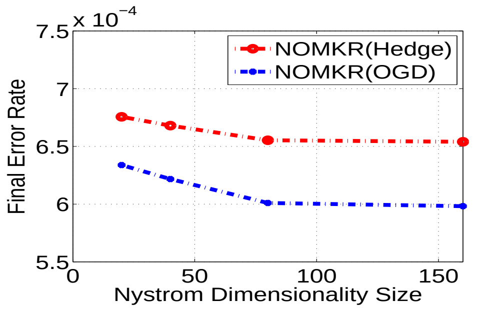
(b) D5



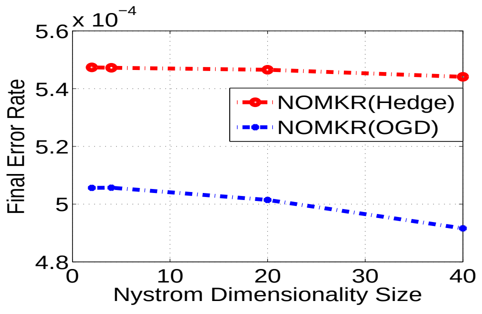
(c) D6



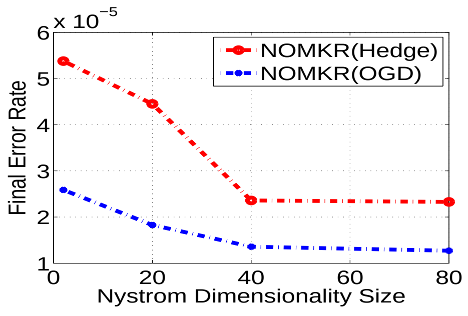
(d) D7



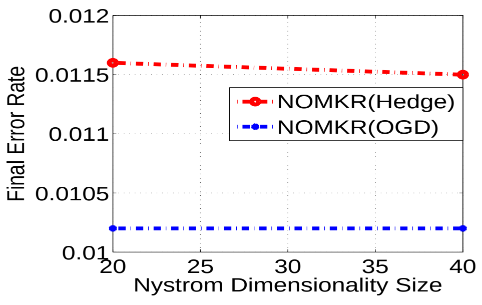
(e) L1



(f) L2



(g) L3



(h) L4

Figure 3.7: 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 $m = 13$).

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.

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 5 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.

Results and Discussion

The results of application of OMKR to time-series prediction can be seen in Table 3.7 for ARMA and Table 3.8 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.

Table 3.7: 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**

Table 3.8: 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	**0.016353**	**0.006764**	0.000261	**0.005968**	**0.000768**

3.7 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 support vectors 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 nonlinear 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.

Chapter 4

Temporal Kernel Descriptors for Time-Sensitive Patterns

Detecting temporal patterns is one of the most prevalent challenges while mining data. Often, timestamps or information about when certain instances or events occurred can provide us with critical information to recognize temporal patterns. Unfortunately, most existing techniques are not able to fully extract useful temporal information based on the time (especially at different resolutions of time). They miss out on 3 crucial factors: (i) they do not distinguish between timestamp features (which have cyclical or periodic properties) and ordinary features; (ii) they are not able to detect patterns exhibited at different resolutions of time (e.g. different patterns at the annual level, and at the monthly level); and (iii) they are not able to relate different features (e.g. multimodal features) of instances with different temporal properties (e.g. while predicting stock prices, stock fundamentals may have annual patterns, and at the same time factors like peer stock prices and global markets may exhibit daily patterns). To solve these issues, in this chapter we offer a novel multiple-kernel learning view and develop Temporal Kernel Descriptors which utilize Kernel functions to comprehensively detect temporal patterns by deriving relationship of instances with the time features.

4.1 Introduction

Mining temporal patterns is one of the most prevalent problems in data mining, and has immense applications and practical utility. Detecting how events are associated or linked with timestamp features can greatly improve our predictive ability and decision making. For example, estimating the weather conditions can help in agriculture, event planning etc.; estimating trends in demands for resources can help in resource allocation (e.g. what time of the day would we have high demand for car rentals, what would be the demand for resources in a super market at different times); estimating stock prices can help us make better investment decisions. It is therefore critical to have techniques that can comprehensively detect temporal patterns from the data.

Several existing machine learning techniques consider timestamp features as ordinary features, and directly plug them into the learning method. Some alternate techniques explicitly address the temporal nature of data (without timestamp usage), by weighting (e.g. exponential weighting) the instances based on recency in the optimization formulation. These approaches suffer from three major drawbacks: (i) they ignore the cyclical or periodic nature of timestamp features (e.g. hour feature repeats itself after every 24 iterations); (ii) they can not recognize temporal patterns at different resolutions of time (e.g. distinction is required between patterns observed at different days of the week, and different hours of the day); and (iii) they do not associate different multi-modal features with different timestamp resolutions (e.g. for stock price prediction, the fundamentals may exhibit monthly patterns, whereas global markets and peer stocks may exhibit daily patterns). To solve these problems, we develop novel multi-resolution kernel functions called *Temporal Kernel Descriptors* which use kernels to describe the temporal patterns in the data by appropriately linking events to timestamps at various resolutions of time. We automatically learn the optimal kernel function that can appropriately measure the temporal similarity between instances by formulating the problem as a Multiple

Kernel Learning (MKL) optimization [5, 65, 110].

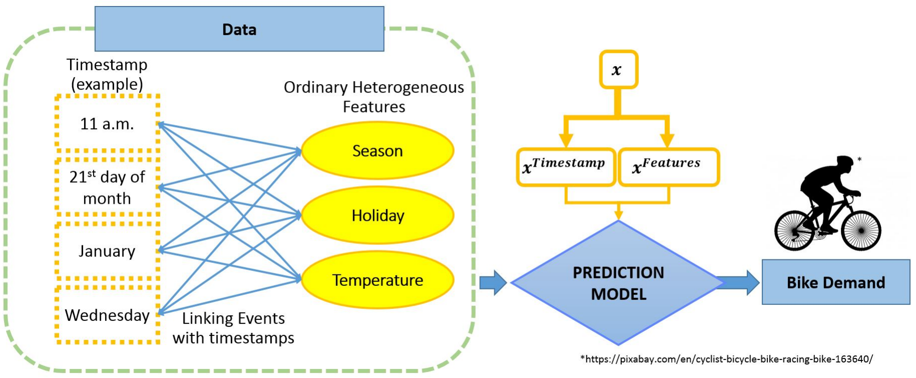


Figure 4.1: To estimate the bike demand, we aim to learn the association between each multi-modal (heterogeneous) feature set with each resolution of time. The prediction model then accepts an input and estimates the demand.

We motivate this problem further through a real world example of bike demand prediction at a given time [34] (See Figure 4.1). The demand may be affected by the hour of the day, or day of the week, or be a function of both in conjunction. Likewise, (multimodal) factors like weather, temperature, or if the day is a holiday (these can be considered heterogeneous data sources) will also be responsible in varying the bike demand. Additionally, these multi-modal factors will have a different effect based on the timestamp (at different resolutions). We aim to design *Temporal Kernel Descriptors* that can not only adapt according to periodic properties of timestamp features at different resolutions, but also be able to find appropriate association of different modalities (or heterogeneous features) with each of the timestamp resolutions. First, we introduce *temporal kernels* which address the problem of time features having a periodic nature. We then extend this to *multi-resolution temporal kernels* to measure time similarity at different resolutions. Finally, we design *temporal kernel descriptors* that associate various modalities to different resolutions of time. We also devise a method to use these descriptors in absence of timestamps. We formulate the optimization as a Multiple Kernel Learning problem, in order to automatically learn the optimal temporal similarity between two instances. We also empirically validate the superiority of using these kernel descriptors in detecting temporal similarity as compared to traditional state of the art linear and kernel methods.

4.2 Related Work

There are two main categories of work related to chapter: Temporal Pattern Mining and Multiple Kernel Learning. Temporal data mining has been studied extensively in literature [3, 67, 86]. We restrict our discussion to kernel methods or timestamp usage. Kernel methods, particularly support vector regression have found success in several time-series applications including finance [113], electricity load forecasting [60], credit rating [55], machine reliability forecasting [126], control systems and signal processing [41] and online learning [97]. Most of these techniques primarily focus directly using the traditional kernels, and applying them directly to the features. Periodic kernel functions have also been used to address issues temporal periodicity in time series [83, 84]. Additionally, some attempts have been made to explicitly address the temporal issues by modifying the objective function like exponential weighting [114], giving more importance to recent data. A similar approach is seen in several online learning methods, in which they adapt to the changing pattern [26]. Another related work is the usage of autoregressive kernels for time series prediction [25]. These methods address a slightly different problem and do not exploit timestamp information to improve predictability. Some efforts have been made to use tree-kernels for linking events to timestamps [85, 53], but here the focus is primarily on natural language processing. None of these methods consider the temporal patterns of multi-modal data at multiple resolutions of time.

Kernel methods are more effective than linear methods when learning a nonlinear target hypothesis [103]. They are also used as notions of similarity. In practice several types of kernel functions exist, with a variety of parameters. It is usually not known before-hand which kernel would be suitable for the task. Further, often data sources are heterogeneous, implying different kernels may be suited to different features (e.g. if an instance has features that may be numerical or strings). In addition, the usage of a single kernel usually restricts the learning capacity by not leveraging on information that could be gotten from more kernels. For a more comprehensive review on Multiple Kernel Learning, see Chapter 2. Unfortunately, none of these MKL techniques consider the usage of kernels to better describe the temporal nature of the multi-modal data at multi-resolution timestamps.

4.3 Temporal Kernel Descriptors

4.3.1 Overview and Problem Setting

Consider a set of N instances $\mathbf{x}_i$, for $i = 1, \ldots, N$, with each instance labeled with a target value y_i, where $y_i \in \{1, -1\}$ for classification or $y_i \in \mathcal{R}$ for regression tasks. Each instance comprises two sets of features: $\mathbf{x}_i^T$, which is a timestamp, or the time at which the event took place, e.g., $\mathbf{x}_i^T = \{12^{th} Jan, 2015|10:28:31\}$ (or any other customized time feature designed by the user); and $\mathbf{x}_i^F$ are those standard features in a typical learning setting that describe the instance.

Based on these two sets of features we aim to learn a kernel based prediction model $\mathbf{f}(\mathbf{x}_i)$ which can optimally predict the target variable y_i. A prediction on instance $\mathbf{x}_i$ with target y_i suffers a loss which is denoted by $\ell(\mathbf{f}, (\mathbf{x}_i, y_i))$. Typically this loss function is convex, and can be application dependent. For example, for binary classification, it can be the hinge loss: $\ell(\mathbf{f}, (\mathbf{x}_i, y_i)) = \max(0, 1 - y_i\mathbf{f}(\mathbf{x}_i))$; and for regression it can be the squared loss: $\ell(\mathbf{f}, (\mathbf{x}_i, y_i)) = \frac{1}{2}(\mathbf{f}(x_i) - y_i)^2$.

Our goal is to learn a function that minimizes this loss over all the instances in the dataset. This can be cast into the following (SVM-like) optimization problem:

$$\min_{\mathbf{f}} \frac{1}{2}||\mathbf{f}||^2 + C\sum_{i=1}^{N} \ell(\mathbf{f}, (\mathbf{x}_i, y_i))$$

where the first term is the regularizer penalizing the complexity of the model, and the C is the tradeoff parameter. In the following, we will describe the kernels used to learn this kernel-based prediction function.

4.3.2 Temporal Kernels

Kernels have found success in many algorithms and applications [108] owing to their ability to detect nonlinear patterns. Kernels implicitly map the input space into a high dimensional space (also called the Reproducing Kernel Hilbert Space (RKHS)), and allow the algorithms to detect linear patterns in this new space. This enables the algorithms to infer nonlinear patterns in the original space. What makes kernels efficient and popular is the *kernel trick*, which allows us to skip the explicit high dimensional mapping, and learn the prediction model in the RKHS while operating in the original space. The kernel trick is used in Kernel Functions, which compute the dot product between instances in a high dimensional space. Popularly used kernel functions include: Polynomial Kernels $\kappa(\mathbf{x}_1, \mathbf{x}_2) = (c + \mathbf{x}_1^\top \mathbf{x}_2)^d$ (c and d are parameters), and Gaussian Kernels $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{||\mathbf{x}_1 - \mathbf{x}_2||^2}{2\sigma^2})$ (σ is the bandwidth parameter). Since kernel functions essentially compute the dot product between two instances, they are often viewed as similarity measures, i.e., the kernel function on two instances returns the similarity between the two instances, where different types of kernels (with different parameters) denote different notions of similarity.

Using this similarity view of kernels, how should we compute similarity between two timestamps? Unfortunately traditional kernels e.g. linear kernel, or (Euclidean) distance based kernels (e.g. Hat Kernel, Gaussian Kernel, etc.) give poor quality similarity scores for time values owing to the cyclical nature of the many time measures (e.g. hour value repeats itself after 24 iterations, seconds value repeats itself after every 60 iterations). Intuitively we can comprehend that 2300 hours and 0100 hours occur at similar times of the day, but traditional kernels consider these two times as very dissimilar due to the large Euclidean distance. To address this problem, we design a *temporal kernel* which can account for the cyclical or periodic nature of timestamps. Let $\mathbf{x}_1^t$ and $\mathbf{x}_2^t$ be two time stamp instances with a cycle length C_t (for hours $C_t = 24$). The temporal kernel similarity measure based on the Gaussian kernel for the the two instances is given by:

$$\kappa_t(\mathbf{x}_1^t, \mathbf{x}_2^t) = \exp - \frac{(\min(|\mathbf{x}_1^t - \mathbf{x}_2^t|, C_t - |\mathbf{x}_1^t - \mathbf{x}_2^t|))^2}{2\sigma^2}$$

Here σ is the bandwidth parameter as used in the Gaussian Kernel. This new temporal kernel measures the Euclidean distance between two time values from the shorter sides for the cycle, and thus gives a better similarity measure. We note that this is similar, albeit a more simplified version of the Mackay Periodic kernel function. An example of this is graphically illustrated in Figure 4.2.

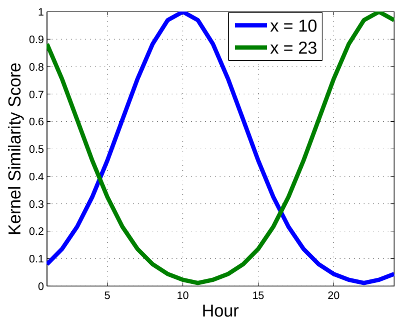


Figure 4.2: Temporal Kernel: The kernel similarity score for $\mathbf{x} = 10$ and $\mathbf{x} = 23$ for $\sigma = 4$. Here, $\mathbf{x}$ represents hours.

4.3.3 Multi-resolution Temporal Kernels

Having addressed the periodic nature of temporal patterns through temporal kernels, we are still faced with the challenge of addressing time similarity measures when multi-resolution timestamps are provided. For example, what would be the kernel similarity between 09:05 a.m. and 11:35 a.m.? Should the similarity be high considering that the time stamps are only a few hours apart, or should the similarity score be low as the minutes feature of the time stamps are far apart? A naive approach would be to convert everything to a standardized unit of time (convert everything to hours), and try to use the temporal kernels. However, temporal patterns at different resolutions may get ignored. The data may exhibit patterns at different resolutions of time - for example, there is a global pattern in every hour and a local pattern in every minute (within each hour). We want to be able to capture the global pattern (changes in every hour), as well as the local pattern (changes at a minutes level).

To do this, we consider each of the specific time features as heterogeneous (or multi-modal) and define the multi-resolution temporal kernel similarity between two stamps as the positively weighted linear combination of kernel similarity of each timestamp feature. Let $\mathbf{x}_1^T$ and $\mathbf{x}_2^T$ be two timestamp instances comprising $T > 1$ features (i.e. $\mathbf{x}_1^T = [x_1^1, \dots, x_1^T]$). The *multi-resolution temporal kernel* similarity based on multiple kernels between the instances is given by:

$$
\begin{aligned}
&\kappa_{Multi-resolutionTemporalKernel}(\mathbf{x}_1^T, \mathbf{x}_2^T) \\
&\qquad = \alpha_1 \kappa_1(\mathbf{x}_1^T, \mathbf{x}_2^T) + \cdots + \alpha_T \kappa_T(\mathbf{x}_1^T, \mathbf{x}_2^T) \\
&\qquad = \sum_{t=1}^{T} \alpha_t \kappa_t(\mathbf{x}_1^T, \mathbf{x}_2^T) \\
&\qquad \text{subject to } \alpha_t \geq 0
\end{aligned}
$$

$\alpha_t \geq 0$ ensures that the resultant kernel function is positive semi-definite keeping the learnt kernel a valid Mercer Kernel. Each kernel function used in the above equation has its own bandwidth parameter and cycle length (C_t), depending on the application and the timestamp feature being used used from the instance $\mathbf{x}^T$.

4.3.4 Temporal Kernel Descriptors

Using appropriate temporal kernels, we now associate the timestamps to the heterogeneous feature sets. First we define feature level similarity between two instances. Let $\mathbf{x}_1^F$ and $\mathbf{x}_2^F$ be two instances (without timestamp information) comprising $F > 1$ feature sets (i.e. $\mathbf{x}_1^F = [x_1^1, \dots, x_1^F]$, where every x_1^f represents a multi-modal data source, and it can have one or more features). The similarity between these two instances is given as a positively weighted linear combination of kernel similarity scores of each feature set. It is denoted as $\kappa_{FeatureKernel}$ and is given by:

$$\begin{aligned}
&\kappa_{FeatureKernel}(\mathbf{x}_1^F, \mathbf{x}_2^F)\\
&\qquad = \beta_1 \kappa_1(\mathbf{x}_1^F, \mathbf{x}_2^F) + \cdots + \beta_F \kappa_F(\mathbf{x}_1^F, \mathbf{x}_2^F)\\
&\qquad = \sum_{f=1}^{F} \beta_f \kappa_f(\mathbf{x}_1^F, \mathbf{x}_2^F)\\
&\qquad \text{subject to } \beta_f \geq 0
\end{aligned}$$

$\beta_t \geq 0$ ensures that the resultant kernel function is positive semi-definite keeping the learnt kernel a valid Mercer Kernel. Each kernel function used can be of any suitable type based on the application and the heterogeneous source. κ_f computes the kernel similarity of instances based on the f^{th} set of heterogeneous features.

Finally, we define *Temporal Kernel Descriptors* (TKD) which depict the instance based similarity based on the time and feature similarity by linking similarity scores of both the time features and multi-modal data features to obtain a multi-resolution kernel similarity function. This similarity is the product of multi-resolution temporal kernel similarity and the instance based feature similarity. It is denoted by $\kappa^{TKD}(\mathbf{x}_1, \mathbf{x}_2)$ for instances $\mathbf{x}_1 = [\mathbf{x}_1^T, \mathbf{x}_1^F]$ and $\mathbf{x}_2 = [\mathbf{x}_2^T, \mathbf{x}_2^F]$:

$$\begin{aligned}
&\kappa_{TKD}(\mathbf{x}_1, \mathbf{x}_2) \qquad (4.1)\\
&\quad = \kappa_{HTK}(\mathbf{x}_1^T, \mathbf{x}_2^T) \cdot \kappa_{FK}(\mathbf{x}_1^F, \mathbf{x}_2^F)\\
&\quad = \left(\sum_{t=1}^{T} \alpha_t \kappa_t(\mathbf{x}_1^T, \mathbf{x}_2^T)\right)\left(\sum_{f=1}^{F} \beta_f \kappa_f(\mathbf{x}_1^F, \mathbf{x}_2^F)\right)\\
&\quad = \sum_{t=1}^{T}\sum_{f=1}^{F} \alpha_t \beta_f \left(\kappa_t(\mathbf{x}_1^T, \mathbf{x}_2^T) \cdot \kappa_f(\mathbf{x}_1^F, \mathbf{x}_2^F)\right)\\
&\quad = \sum_{t=1}^{T}\sum_{f=1}^{F} w_{tf} \left(\kappa_t(\mathbf{x}_1^T, \mathbf{x}_2^T) \cdot \kappa_f(\mathbf{x}_1^F, \mathbf{x}_2^F)\right)
\end{aligned}$$

As can be seen, the TKD similarity is a linear combination of a set of different kernel functions. Since $\alpha_t \geq 0$ and $\beta_f \geq 0$, $w_{tf} = \alpha_t \cdot \beta_f \geq 0$. The final kernel function is a conic combination of a product of kernel functions. Both kernel product and conic combination preserve a positive semi-definite kernel, which means the final kernel function is also positive semi-definite, and is a Mercer Kernel.

Each product of two kernel functions establishes a link or an association between a timestamp feature and each of the multi-modal data sources. This helps us associate events for a specific set of multi-modal features with the time at which they occurred and allows us to capture the temporal similarity in the data at various resolutions of time for each modality. Our next step is to learn the optimal coefficients w_{tf} for each of the multiple kernel functions, in order to derive the optimal kernel function.

4.3.5 Optimization and Algorithms

We wish to minimize human intervention in determining the best kernel function, and we want an optimization objective that can automatically find this best kernel function. Our objective is to learn the optimal coefficients w_{tf} such that the optimal kernel function can minimize the loss over the entire data. Given T timestamp features, and F instance features, we get a total of $m = T \times F$ kernel functions, whose weighted combination needs to be learnt. Following a structural risk minimization principle, we can cast this problem into the following multiple kernel learning (MKL) optimization:

$$\min_{\mathbf{w}\in\Delta} \min_{\mathbf{f}\in\mathcal{H}_\kappa} \frac{1}{2}||\mathbf{f}||^2_{\mathcal{H}_\kappa(\mathbf{w})} + C\sum_{i=1}^{N} \ell(\mathbf{f}, (\mathbf{x}_i, y_i)) \tag{4.2}$$

where $\Delta = \{\mathbf{w} \in \mathcal{R}^m_+ | \sum_{j=1}^{m} w_j = 1\}$ and $\kappa(w)(\mathbf{x}_1, \mathbf{x}_2) = \sum_{t=1}^{T}\sum_{f=1}^{F} w_{tf}\Bigg(\kappa_t(\mathbf{x}_1^T, \mathbf{x}_2^T)\cdot\kappa_f(\mathbf{x}_1^F, \mathbf{x}_2^F)\Bigg)$. Further $\ell(\mathbf{f}, (\mathbf{x}_i, y_i))$ is a convex loss function that can suitably chosen based on the task at hand. There are many techniques to solve this optimization involving structural risk minimization with multiple kernels [110, 95, 124, 43].

Unfortunately, MKL optimization shown above is computationally very expensive. In particular when data has a lot of instances, or the number of kernel functions used is large, the training cost (and re-training cost) is very high. In fact, in the scenario of multi-modal data with multi-resolutions of timestamps, the number of kernels quickly expands as a product of the number of modalities and number of timestamp resolutions. To alleviate this problem, Online MKL [58, 49, 100] has evolved as a promising research direction, in which both the kernel predictions and their optimal combination are learnt in an online or sequential fashion. In the following we briefly discuss this approach in order to learn a kernel-based prediction function for time-sensitive patterns using Temporal Kernel Descriptors.

The idea of OMKL is to first define a pool of base kernels. For our task, the predefined pool of base kernels are the $m = T \times F$ kernels. The entire learning task is done in the online setting. This means that the instances arrive sequentially. In every iteration, an instance $\mathbf{x}_i$ is revealed to the model. The model makes a prediction $\hat{y}_i = \mathbf{F}_i(\mathbf{x}_i)$ in each iteration. Subsequently, the environment reveals the true value y_i. As a result, the model suffers loss $\ell(\mathbf{F}_i, (\mathbf{x}_i, y_i))$. Finally, the model updates itself based on this loss, with the objective to achieve the lowest possible loss across all the instances. The main task is to decide how to do the update in such a manner that we are not only learning the optimal prediction function, but also the optimal combination of multiple kernels simultaneously. OMKL approaches this problem with a two-step procedure in every iteration: (i) First each kernel predictor is updated based on the loss it has individual suffered; and (ii) Second, the combination weights of the multiple kernel predictors is updated based on loss suffered by not having the optimal combination.

Learning the Kernel Predictor: An online kernel model with the prediction function $\mathbf{f}(\mathbf{x})$ is updated by gradient descent [134, 61] with learning rate parameter η when the model suffers loss. Such a prediction model is learnt for *each* of the $m = T \times F$ kernels in the predefined pool. The loss for classification can be the hinge loss ($\ell(\mathbf{f}, (\mathbf{x}, y)) = \max(0, 1 - y(\mathbf{f} \cdot \mathbf{x}))$, and for regression, we consider the squared loss ($\ell(\mathbf{f}, (\mathbf{x}, y)) = (\mathbf{f} \cdot \mathbf{x} - y)^2$). The update is made as follows:

$$\begin{aligned}
\mathbf{f}_{i+1}(\mathbf{x}) &= \mathbf{f}_i(\mathbf{x}) - \eta \nabla_{\mathbf{f}} \ell(\mathbf{f}_i, (\mathbf{x}_i, y_i)) \\
&= \mathbf{f}_i(\mathbf{x}) + \eta y_i \kappa^{TKD}(\mathbf{x}_i, \mathbf{x})(\text{for classification}) \\
& \quad \text{OR} \\
&= \mathbf{f}_i(\mathbf{x}) - \eta y_i(\mathbf{f}_i(\mathbf{x}_i) - y_i)\kappa^{TKD}(\mathbf{x}_i, \mathbf{x})(\text{for regression})
\end{aligned}$$

At the end of each online learning round, we can express the prediction function as a kernel expansion [103]:

$$\mathbf{f}_{i+1}(\mathbf{x}) = \Sigma_{j=1}^{i} \lambda_j \kappa^{TKD}(\mathbf{x}_j, \mathbf{x}) \tag{4.3}$$

where the λ_i coefficients are computed based on the update rule. If a non-zero loss was suffered on the i^{th} instance, then $\lambda_i \neq 0$ and the instance becomes a support vector; and if no loss is suffered, the i^{th} instance is not a support vector for which $\lambda_i = 0$. As can be seen, the final prediction of a single kernel is linear combination of kernel similarity of the instance to be predicted with all existing support vectors.

Learning the Multiple Kernel Combination: Having designed the methodology to optimize each kernel predictor, all of them should be suitably combined to give the final prediction. The final prediction is a weighted combination of the $m = T \times F$ kernel predictors, given by:

$$\hat{y_i} = \mathbf{F}_i(\mathbf{x}_i) = \frac{\sum_{k=1}^{m} w_i^k(\mathbf{f}_i^k(\mathbf{x}_i))}{\sum_{k=1}^{m} w_i^k} \tag{4.4}$$

To update combination of weights $\mathbf{w} = (w^1, \ldots, w^m)^\top$, where w^i is set to $1/m$ at the beginning of the learning task, we use the *Hedge* algorithm [37, 49]. At the end of each learning iteration, the weights are updated by:

$$w_{i+1}^k = w^k \beta^{\ell(\mathbf{f}_i^k;(\mathbf{x}_i,y_i))} \tag{4.5}$$

where $\beta \in (0,1)$ is the discount rate Hedge parameter, and $\ell(\mathbf{f}_i^k; (\mathbf{x}_i, y_i)) \in (0,1)$ represents the loss suffered by the kernel predictor. At the time of prediction, the weights are normalized to a distribution, ensuring that the weights of the prediction sum up to 1. Similarly, other linear online learning methods could be adopted to learn the optimal combination.

4.3.6 Discussion

In this part, we will briefly discuss some properties of OMKL with Temporal Kernel Descriptors.

Theoretical Analysis: We derive a loss bound for OMKL with Temporal Kernel Descriptors. We assume $\kappa(\mathbf{x}, \mathbf{x}) \leq 1$ for all κ and $\mathbf{x}$. We define the optimal objective value for the kernel $\kappa_k(\cdot, \cdot)$ denoted by $O(\kappa_k, \ell, \mathcal{D})$ with respect to the dataset $\mathcal{D}$ as the regret of an online learning algorithm used for a single kernel predictive model. Online gradient descent gives us the regret of the algorithm with respect to the best linear predictor in the Reproducible Kernel Hilbert space induced by kernel κ_k. Since Online Gradient Descent is a no regret algorithm, the regret tends to zero, as the number of instances N goes to infinity.

On processing a sequence of N instances, the cumulative loss suffered by the OMKL using temporal kernel descriptors, where each kernel predictor is updated by online gradient descent is bounded as

$$L_{TKD} \leq \frac{\ln(\frac{1}{\beta})}{1-\beta} \min_{1 \leq k \leq m} O(\kappa_k, \ell, \mathcal{D}) + \frac{\ln m}{1-\beta}$$

L_{TKD} denotes the cumulative loss suffered by the algorithm. By setting $\beta =$

$\frac{\sqrt{N}}{\sqrt{N}+\sqrt{\ln m}}$, we get:

$$L_{TKD} \quad \leq \quad \Big(1 \;+\; \sqrt{\frac{\ln m}{N}} \min_{1\leq k\leq m} O(\kappa_k, \ell, \mathcal{D}) \;+\; \ln m \;+\; \sqrt{N \ln m}\Big)$$

Proof. The proof combines the Zinkevich theorem [134] and the bound for Hedge Algorithm [37]. The Zinkevich algorithm gives us a no regret algorithm for a particular Reproducible Kernel Hilbert Space induced by a particular kernel. Using a set of m kernels, and updating their weight distribution via the Hedge algorithm, we can directly plug the regret into the loss bound of Hedge algorithm as shown in Equation (4.6). Then, optimally choosing a value for discount rate parameter β we get the result in the lemma. This tells us that OMKL using temporal kernel descriptors in the worst case scenario will converge in performance to the most informative (or the best) kernel descriptor (though in practice, using complimentary information from different kernels is likely to enhance the performance). $\square$

Time Complexity: In the worst case scenario, when all instances become support vectors for a single kernel, the computational complexity of each iteration for each kernel is in $O(N)$. Repeating this for N iterations the time complexity is in $O(N * N) = O(N^2)$. Further, these operations have to be performed for all $m = T \times F$ kernels, which means the time complexity of running OMKL is in $O(mN^2)$. An unbounded number of support vectors can make the algorithm less practical, despite a polynomial running time. Several techniques in literature have been proposed to address this issue based on budget approximations [13, 30, 91]. These techniques require a prespecified budget B, which is the maximum number of support vectors a kernel prediction model can store. This budget reduces time complexity of a single kernel predictor on the entire data from $O(N^2)$ to $O(NB)$, and consequently reduces time complexity of OMKL to $O(mNB)$. This is linear in the number of instances N, and hence scales well with large number of instances (as compared to batch MKL methods).

Usage in absence of timestamps: The key to detecting temporal patterns through temporal kernel descriptors lies in having suitable temporal kernels. The temporal kernel descriptors that we have presented are primarily designed for settings where timestamps are available. However, many applications, despite having a temporal nature, may not have timestamps. A new challenge surfaces - how to use temporal kernels in such scenarios? To address this issue we propose user-defined timestamps, which are parameterized by cycle length C_t. We assume that the data instances arrive sequentially, and in a chronological order (i.e. the instances that occurred first, arrive first). The user can specify temporal cycles as:

$$\mathbf{C}_T = (C_1, C_2, \ldots, C_T)^\top$$

Using these cycle length values, we can design appropriate temporal kernel descriptors. The parameters $\mathbf{C}_T$ can be determined through some simple validation technique to determine appropriate cycle lengths.

4.4 Experiments

4.4.1 Experimental Setting

Baselines

We perform our analysis for regression tasks, by choosing a squared loss function. In principle, this can be trivially extended to classification tasks as well. We compare the performance of traditional kernels against the proposed Temporal Kernel Descriptors.

Traditional Kernels: First set is the baselines, which comprise traditional usage of kernel methods. These include simple online **Linear** (linear kernel) regression based on gradient descent [134]. We also perform online regression using a variety of kernel functions. We define diverse set of 10 kernels which include three polynomial kernels $\kappa(x, y) = (x^T y)^p$ of degree parameter $p = 1, 2, 3, 4$, five RBF kernels ($\kappa(x, y) = e^{(\frac{-||x-y||^2}{2\sigma^2})}$) of kernel width parameter $\sigma = 2^{-2}, 2^{-1}, 2^0, 2^1, 2^2$, and a sigmoid kernel($\kappa(x, y) = \tanh(xy)$). The algorithm **Best Kernel(Val)** denotes the performance of single kernel online regression, where the choice of kernel has been determined by validating all kernels on first 10% of the instances. **Best Kernel** follows the same principle, but denotes the best kernel choice determined in hindsight (hence usually not realistically attainable). The final algorithm in this set is Online Multiple Kernel Regression (**OMKR**) [100], which tries to learn the optimal combination of multiple kernels (all 10 kernels in this case) to make the optimal prediction. **Temporal Kernel Descriptors**: The next set of 3 algorithms make use of temporal kernel descriptors. **(Temporal+Feature) Kernels** essentially uses multi-resolution temporal kernels and regular feature kernels as multi-modal kernel functions (and does not multiply or link the temporal properties to each multi-modal feature like in Equation (4.1)). This algorithm aims to see the direct advantage of using multi-resolution temporal kernels over existing methods. Like in our proposed method, these kernels are also combined using the Hedge algorithm. The next two algorithms are based on temporal kernel descriptors, i.e., to evaluate the advantage of associating different multi-modalities with different time features, and finding temporal patterns among those (by multiplying temporal kernels with feature kernels like in Equation (4.1)). The first is **TKD(Uniform)** where all the kernel descriptors are equally weighted, and the second algorithm is **TKD(Hedge)** (our proposed method), where the weights of the kernel descriptors are learnt using Hedge Algorithm (as outlined in the previous section). We also compared variants of TKD(Hedge) with the usage of linear indicator kernels instead of the proposed temporal kernels to directly show the advantage of the nonlinearity offered by the temproal kernel. The first variant is **TKD(Indicator)**, where each time feature is represented as a one-hot vector, and using this we compute a linear kernel, and incorporate into the TKD framework. Similarly, we consider another variant **TKD (Neighbour)**, where each time feature is represented as an indicator vector, with the time value, and its immediate neighbour being 1s, and other elements as 0s. Again, the linear kernel is used, and this kernel is incorporated into the TKD framework. For all gradient descent algorithms, we set the learning rate $\eta = 0.1$, and for Hedge, we set the discount rate parameter $\beta = 0.5$. We also set a budget for online kernel methods $B = 1000$ support vectors. These parameters are common to all algorithms, and hence changes in them would not significantly affect our results of comparison between temporal kernels and traditional kernel methods.

Data

We perform our experiments on 2 datasets, that are applications in which time-stamp data is available. First, is the **Bike Demand** dataset obtained from UCI repository. The goal is the predict the bike demand based on factors such as weather, humidity, whether a holiday, etc. We use the Day of the Week (Cycle = 7), and Hour of the day (Cycle = 24), as the multi-resolution temporal kernels. Second, is the **Twitter Traffic** dataset, which we have collected ourselves. The aim is to predict the number of tweets in a specific hour. The entire dataset was processed after analyzing over 68 million tweets from June, 2012 to May, 2013. All tweets are from micrologies who identified themselves as software developers. Again Day of the Week and Hour of the day were used for computing temporal kernels. We also evaluated our method on univariate time series data obtained from Santa Fe Time Series Competition. Here, timestamps are absent, and we arbitrarily designed our own temporal kernels for the task. The two datasets are **Astrophysical** Data, and a **Synthetic** dataset. For both of them, the past 20 values were considered as the input features to predict the next value. In our experiments, we scaled all features (including the target) to lie in $[0, 1]$. For all our datasets, we use 2 resolutions of timestamps, and 2 multi-modal data sources based on the type of features available. We also use an additional modality of features to obtain a trivial kernel function that always evaluate to 1. When this is multiplied by a multi-resolution temporal kernel, it gives us the ability to use pure temporal kernels in our final model as well. The parameter cycle length C_T, and the bandwidth parameter σ for each kernel was set via a grid search on validation data. The other relevant details of the datasets can be seen in Table 6.1 (here the instance features are combined number of features from multi-modal sources).

Table 4.1: Datasets used in experiments

Dataset	Instances	Features	Timestamp Features
Bike	10884	13	Hour, Day of Week
Twitter	7296	12	Hour, Day of Week
Astrophysical	598	20	User Defined
Synthetic	50000	20	User Defined

Table 4.2: Final Mean Squared Error of algorithms on datasets with time-stampsand on time-series datasets

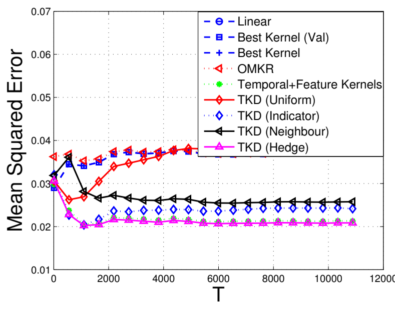
The first 4 are based on traditional kernels. The last 3 algorithms are based on Temporal Kernels. The best performance is in bold.

	Time-stamp Applications		**Time Series**	
Algorithms	**Bike Demand**	**Twitter**	**Astrophysical**	**Synthetic**
Linear	0.1925	0.2591	0.0088	0.0111
Best Kernel (Val)	0.0379	0.0242	0.0088	0.0111
Best Kernel	0.0379	0.0242	0.0088	0.0091
OMKR	0.0380	0.0244	0.0080	0.0090
(Temporal+Feature)Kernels	0.0214	0.0048	0.0077	**0.0081**
TKD(Uniform)	0.0420	0.0089	0.0101	0.0176
TKD(Indicator)	0.0242	0.0048	0.1265	0.0390.
TKD(Neighbour)	0.0258	0.0048	.0503	0.0534
TKD(Hedge)	**0.0209**	**0.0046**	**0.0071**	0.0085

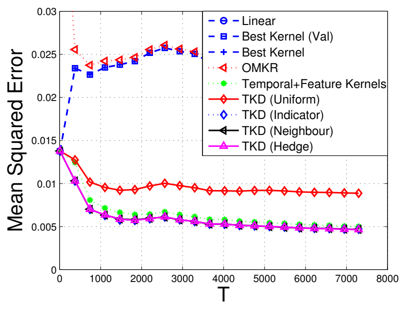
4.4.2 Results and Discussion

The final mean squared error achieved by the algorithms on the datasets are shown in Table 4.2, and the performance of the algorithms as the number of instance increases in an online learning setting can be visualized in Figure 4.3. It can be seen that the algorithms using temporal kernels achieve a significantly lower mean squared error than the algorithms that use traditional kernels. This impact is particularly much more significant in Applications with Time-stamps, where the MSE achieved by our proposed method is almost 50% of traditional kernels for Bike Demand, and for Twitter Traffic Prediction, the MSE is lower than 20% of the best of the traditional

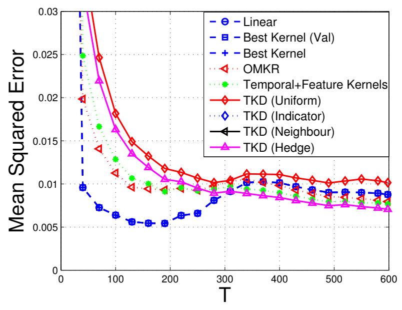
Figure 4.3: Mean Squared Error suffered by the algorithms as the number of instances (T) increases. For (a) and (b), the performance of Linear Kernel Regression is extremely poor, and is out of scale.



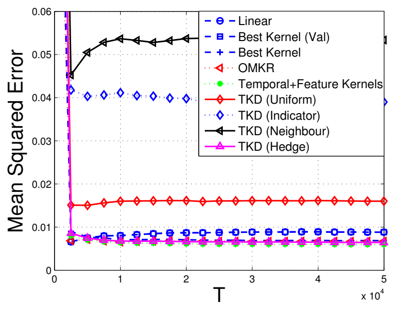
(a) Bike Demand



(b) Twitter Traffic



(c) Astrophysical (Santa Fe)



(d) Synthetic (Santa Fe)

kernel methods. This indicates that the patterns are indeed time-sensitive, and validates the usage of Temporal Kernel Descriptors. TKD(Uniform) struggles to given a good performance, indicating, that a trivial combination of multiple kernels may give unstable results. Hence, it is important to learn the optimal kernel combination like we have proposed for TKD(Hedge).

The other observation is that the usage of temporal kernels itself significantly enhances the pattern recognition power as can be seen in performance of (Temporal+Feature) Kernels. Furthermore, using TKD(Indicator) and TKD(Neighbour) also give a good performance compared to other baselines, but they are not as good as using the proposed nonlinear temporal kernel. This validates the power of using multi-resolution temporal kernels to measure timestamp similarity over the usage of timestamps as ordinary features with traditional kernels. Further, this performance is improved by using the temporal kernel descriptors in most cases. This validates the motivation to associate various modalities of data with multiple resolutions of time, thus giving superior performance. On the whole, it is very evident that the usage of our proposed temporal kernels and temporal kernel descriptors are significantly able to boost the performance of kernel algorithms to detect temporal patterns.

4.5 Conclusion

We propose temporal kernel descriptors, which overcome the existing limitations of kernel methods in finding temporal patterns by appropriately associating multi-resolution timestamps to multi-modal instance features and obtain the optimal temporal similarity between instances. Temporal kernels are devised which account for periodicity of time features. These are extended to multi-resolution temporal kernels, which measure time similarity at different resolutions. Finally, temporal kernel descriptors are developed, which find the optimal association of a multi-modal features with the different resolutions of time. The goal is to automatically learn the optimal combination of multiple kernel similarity scores. This is formulated as a Multiple Kernel Learning problem. We solve the optimization using an online learning approach. Empirically, our kernel descriptors significantly outperform traditional kernel methods due to their ability to obtain the right feature representation for the time-sensitive patterns, thus offering a novel multi-kernel view to detecting temporal patterns in the data.

Chapter 5

Cost-Sensitive Online Multiple Kernel Classification

In this chapter, we investigate supervised machine learning techniques for mining data streams with application to online anomaly detection. Unlike conventional machine learning tasks, machine learning from data streams for online anomaly detection has several challenges: (i) data arriving sequentially and increasing rapidly, (ii) highly class-imbalanced distributions; and (iii) complex anomaly patterns that could evolve dynamically. To tackle these challenges, we propose a novel Cost-Sensitive Online Multiple Kernel Classification (CSOMKC) scheme for comprehensively mining data streams and demonstrate its application to online anomaly detection. Specifically, CSOMKC learns a kernel-based cost-sensitive prediction model for imbalanced data streams in a sequential or online learning fashion, in which a pool of multiple diverse kernels is dynamically explored.

5.1 Introduction

With an increasing interest in mining large data streams, there is a need to design scalable and effective learning algorithms that can comprehensively address emerging big data analytics challenges. In this chapter we focus our attention on supervised learning for data streams with imbalanced labels with application to anomaly detection. Real world examples include intrusion detection [98], anomaly detection in video surveillance [123], fraud detection in markets [32], and many others [15]. Despite extensive studies this remains a challenging problem due to a number of issues including: (i) *high complexity (nonlinearity)* of the (anomaly) patterns; (ii) *high class-imbalanced distributions* where the number of anomaly examples could be significantly less than normal ones; (iii) *high variety* of patterns dynamically changing due to a variety of anomaly behaviors, and *high variety* of data to be processed (heterogeneous data sources, multi-modal data etc.); (iv) *high volume and velocity* of sequentially arriving data; and (v) *pattern evolution* or concept drifts.

Most existing strategies only partly address the challenges posed by imbalanced data streams, and as a result the current state of the art is not able to provide a comprehensive solution to mining imbalanced data streams or for anomaly detection. We design Cost-Sensitive Online Multiple Kernel Classification (CSOMKC) algorithms, which provide a novel method to address all the above challenges of cost-sensitive online classification (and online anomaly detection) from big data streams, including (i) highly complex patterns via kernel methods; (ii) high class imbalance via cost-sensitive learning; (iii) high variety and data heterogeneity via multiple kernel learning; (iv) high volume and velocity via online learning algorithms; as well as (v) dealing with the concept drifting via online multi-kernel learning. Designing such a technique is a significantly challenging, and would have several applications. To the best of our knowledge, this is the first learning method that can simultaneously address all these issues in a simple, efficient, scalable yet effective framework.

Traditional classification algorithms aim to maximize accuracy. However, if the data exhibits imbalanced label distribution, accuracy becomes a poor measure of performance. As a result, we consider alternate metrics *sum* (weighted combination of specificity and sensitivity) and *cost* (weighted cost of misclassification of positive and negative instances) for evaluation of algorithms on imbalanced data streams. We develop a cost-sensitive multiple kernel formulation for the problem to be solved based on maximizing *sum* or minimizing *cost* and then design an online solution. We split the learning procedure in each online learning iteration - by first updating the kernel based prediction function for each of the kernels in the predefined pool (each kernel can be a different function, or different modality of data source), followed by dynamically exploring the multiple kernels, and updating the kernel combination. Both are done in an online manner, thus dealing with scalability concerns and concept drift. In addition, both the updates account for the cost-sensitive nature of the data streams. Further, we derive theoretical guarantees, and obtain the lower bound and upper bound of *sum* and *cost* respectively, obtained by our algorithms. We conduct extensive empirical analysis and show how our proposed methods outperform other state of the art cost-sensitive algorithms.

5.2 Related Work

Our work is primarily related to online learning and cost-sensitive learning and their intersecting studies. Online Learning refers to a family of scalable learning methods that incrementally update the model from a stream of data [14, 51]. See Chapter 2 for a comprehensive review. Most of these methods do not consider imbalanced data distribution and hence are not suitable for imbalanced data streams or online anomaly detection. The most closely related work is the class of cost-sensitive online learning methods that directly try to optimize over a cost-sensitive metric. These include PAUM [74] which is a Perceptron based algorithm for uneven margins, cost-sensitive variant of Passive-Aggressive algorithms CPA$_{PB}$ [21]; and CSOC - cost-sensitive online classification [117]. There also have been some efforts in attempting to maximize the Area Under the Curve in an online manner [130, 31]. Yet, none of these methods explore how to address the complexity of data through other methods (e.g. kernels or multi-kernel solutions), thus limiting their applicability in real world settings. Kernel methods have shown tremendous success in detecting complex nonlinear patterns owing to their ability to induce a high-dimensional reproducible kernel Hilbert space, and learning linear patterns in this space [103]. See Chapter 2 for a comprehensive review. Online Learning with Kernels [61] and Online Multiple Kernel Learning [58, 81, 49]have also been proposed to address scalability, but they do not generalize well with imbalanced data streams.

5.3 Cost-Sensitive Online Multiple Kernel Classification

5.3.1 Problem Setting

Consider a binary classification task. Here, our goal is to learn a function $f : \mathbb{R}^d \to \mathbb{R}$ based on a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in |\mathcal{Y}| = \{-1, +1\}$ is the class label assigned to $\mathbf{x}_t$. We use $\hat{y} = \text{sign}(f(\mathbf{x}))$ to predict the class assignment for any $\mathbf{x}$, and the magnitude of $f(\mathbf{x})$ to measure the classification confidence. Performances of the learnt functions are usually evaluated based on accuracy A:

$$A = \frac{\sum_{t=1}^{T} \mathbb{I}_{(\hat{y}_t = y_t)}}{T}$$

Here $\mathbb{I}$ is the indicator function resulting in 1 if the condition is true, and 0 otherwise. Unfortunately, many real world datasets present imbalanced labels. For a dataset with 99% labels as -1, a model that classifies all instances as -1 has an accuracy of 99%, which prima facie seems good, but it is obviously not a good performance. Clearly, Accuracy is not a good performance indicator for (imbalanced) classification. Accordingly, for imbalanced labels, we evaluate algorithms on cost-sensitive measures: *sum* and *cost*.

Sum is the weighted sum of *sensitivity* and *specificity* of the algorithm. Let $T_p = \{t \,|y_t = +1\}$ and $T_n = \{t \,|y_t = -1\}$ denote the set of positive and negative instances respectively. $\mathcal{M} = \{t \,|y_t \neq \hat{y_t}\}$ is the set of indexes that correspond to a mistake. Similarly, we have $\mathcal{M}_p = \{t \,|y_t \neq \hat{y_t};\; y_t = +1\}$, and $\mathcal{M}_n = \{t \,|y_t \neq \hat{y_t};\; y_t = -1\}$ for mistakes on positive and negative instances. Lastly, $|S|$ denotes the number of instances in any set S. Sensitivity (S_e) and Specificity (S_p) are defined as: $S_e = \frac{|T_p|-|\mathcal{M}_p|}{|T_p|}$ and $S_p = \frac{|T_n|-|\mathcal{M}_n|}{|T_n|}$. The weighted *sum* parameterized by $\alpha \in [0, 1]$ is given as:

$$sum = \alpha(S_e) + (1 - \alpha)(S_p)$$

For $\alpha = 0.5$, sum is reduced to balanced accuracy.

Cost is the weighted sum of mistakes on positive and negative instances, and is parameterized by $c \in [0, 1]$:

$$cost = c(|\mathcal{M}_p|) + (1 - c)(|\mathcal{M}_n|)$$

Here, the aim is to tradeoff the cost of wrongly classifying a positive instance against the cost of wrongly classifying a negative instance using tradeoff parameter c.

Our objective is to either maximize sum or minimize $cost$. We transform both to the following objective:

$$\min_f \sum_{y_t=+1} \rho \mathbb{I}_{\hat{y_t} \neq y_t} + \sum_{y_t=-1} \mathbb{I}_{\hat{y_t} \neq y_t} \tag{5.1}$$

5.3.2 Cost-Sensitive Multiple Kernel Classification

Data streams may exhibit complex nonlinear patterns. Kernels have evolved as popular tools to detect nonlinearity by mapping a low dimensional feature space to a high dimensional space. We aim to learn a kernel-based prediction function to optimize the cost-sensitive measure in Eq. (5.1), in order to detect nonlinear patterns. We propose to use Multiple Kernel Learning (MKL) so that: (i) prior knowledge of appropriate kernel is not required; (ii) model's learning capacity increases when multiple kernels complement each other; and (iii) heterogeneous data sources can be combined into one prediction model (e.g. using different kernels for numeric and text data, or different kernels for different modalities of data). This way we are able to learn a powerful model which can detect complex nonlinear patterns and handle a variety of data.

To do this, we first define a loss function as the convex surrogate of the indicator function (which is not convex and has been used in Eq. (5.1)), and we get:

$$\ell^{\rho}(f, (\mathbf{x}, y)) = (\rho \mathbb{I}_{y=1} + \mathbb{I}_{y=-1}) * \max(0, 1 - y(f \cdot \mathbf{x})) \tag{5.2}$$

Using this loss function, Eq. (5.1) can be cast into the following regularized optimization (C is the regularization tradeoff parameter):

$$\min_{f} \frac{1}{2}\|f\|^2 + C\sum_{t=1}^{T} \ell^{\rho}(f, (\mathbf{x}, y)) \tag{5.3}$$

Our goal is to solve this using MKL. Consider a collection of m different predefined kernel functions $\mathcal{K} = \{\kappa_i : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, i = 1, \ldots, m\}$. MKL aims to learn a kernel-based prediction model by identifying the best convex combination of the m kernels, that is, a weighted combination $\theta = (\theta_1, \ldots, \theta_m)$. The proposed cost-sensitive multiple kernel machine can be cast into the following optimization:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_{K}(\theta)} \frac{1}{2}\|f\|^2_{\mathcal{H}_{K(\theta)}} + C\sum_{t=1}^{T} \ell^{\rho}(f, (\mathbf{x}_t, y_t)) \tag{5.4}$$

where $\Delta = \{\theta \in \mathbb{R}^m_+ | \theta^T \mathbf{1}_m = 1\}$, $K(\theta)(\cdot,\cdot) = \sum_{i=1}^{T} \theta_i \kappa_i(\cdot,\cdot)$; $\mathcal{H}_{K(\theta)}$ is the Reproducible Kernel Hilbert Space induced by the multiple kernel combination; and $\ell^{\rho}(f, (\mathbf{x}_i, y_i))$ is a convex loss function as defined in Eq. (5.2). The optimization can be solved by adapting existing techniques (see section on Related Work). However, it is computationally challenging and not suitable for large data streams, and data with temporal properties. Most of the techniques suffer from extremely high retraining cost and expensive memory requirements. To tackle this challenge, we propose online learning based CSOMKC algorithms.

5.3.3 CSOMKC Algorithms

We design Cost-Sensitive Online Multiple Kernel Classification (CSOMKC), which learns the model in an online learning (hence scalable and adaptive to temporal patterns) setting. Instances are sequentially processed, and in each iteration we aim to update the kernel prediction model and the kernel combination. Doing both simultaneously in an online manner is significantly challenging, and due to imbalanced data, traditional methods can not be directly applied. We update the model via a 2-step approach: updating each kernel predictor, and updating the kernel combination.

Cost-Sensitive online kernel classification

We first develop a single-kernel cost-sensitive online kernel classification method. In every iteration of the online learning procedure, a kernel classifier with the prediction function $f(\mathbf{x})$ is updated by gradient descent [61, 117] when the classifier suffers a nonzero loss. Using the cost-sensitive loss from Eq. (5.2) we can obtain the cost-sensitive gradient descent update by taking the derivative. The update rule for each individual kernel-based model is given by:

$$\begin{aligned} f_{t+1}(x) &= f_t(x) - \eta \nabla_f \ell_t^\rho(f_t, (\mathbf{x}_t, y_t)) \\ &= f_t(x) + \eta \rho_t y_t \kappa(\mathbf{x}_t, \mathbf{x}) \end{aligned}$$

where ρ_t manages the cost-sensitivity, by setting $\rho_t = \mathbb{I}_{(y_t=+1)} + \rho \mathbb{I}_{(y_t=-1)}$, and η is the learning rate parameter. At the end of each online learning round, we can express the prediction function as a kernel expansion [103] $f_{t+1}(\mathbf{x}) = \Sigma_{i=1}^{t} \lambda_i \kappa(\mathbf{x}_i, \mathbf{x})$ where the λ_i coefficients are computed based on the update rule. For non-zero loss on the i^{th} instance, $\lambda_i \neq 0$ (the instance becomes a support vector) otherwise $\lambda_i = 0$.

Online multi-kernel combination learning

All $i = 1, \ldots, m$ kernel predictions (denoted by f_t^i) are combined to make a final weighted prediction on each iteration:

$$\hat{y}_t = \text{sign}\Big(\sum_{i=1}^{m} w_t^i \big(f_t^i(\mathbf{x}_t)\big)\Big)$$

We propose two new cost-sensitive weight combination learning schemes: 1) Based on Exponentiated Gradient; and 2) Based on Online Gradient Descent.

EG Combination: We aim to learn the optimal cost-sensitive convex combination of weights $\mathbf{w} = (w^1, \ldots, w^m)^\top$, where w^i is set to $1/m$ at the beginning of the learning task. In our approach, we modify and adapt the *EG* algorithm [62] to update the cost-sensitive weights. We define

$$\mathbf{f}_t(\mathbf{x}_t) = (f_t^1(\mathbf{x}_t), \ldots, f_t^m(\mathbf{x}_t))^\top, \tag{5.5}$$

and formulate the rule of updating $\mathbf{w}$ as follows:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\Delta} D_{KL}(\mathbf{w}\|\mathbf{w}_t) + \eta_{eg}\ell^\rho(\mathbf{w}, (\mathbf{f}_t(\mathbf{x}_t), y_t)), \tag{5.6}$$

This optimization trades off two major concerns: (i) minimizing weight distribution between new weights and old weights (measured by KL-divergence); and (ii) new weights should suffer a small loss on the instance in the current iteration. The trade-off parameter is $\eta_{eg} > 0$. To obtain a closed-form solution for the above optimization, we approximate the loss function by using its first-order Taylor expansion at $\mathbf{w}_t$, and we get:

$$\begin{aligned}\mathbf{w}_{t+1} = &\arg\min_{\mathbf{w}\in\Delta} D_{KL}(\mathbf{w}\|\mathbf{w}_t) + \eta_{eg}\ell^\rho(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t)) \\ &+ \eta_{eg}\partial_{\mathbf{w}}\ell^\rho(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t)) \cdot (\mathbf{w} - \mathbf{w}_t)\end{aligned} \tag{5.7}$$

For this problem, we can derive a closed-form solution as:

$$\mathbf{w}_{t+1} = \frac{\mathbf{w}_t \odot \exp(-\eta_{eg}\nabla_{\mathbf{w}}\ell^{\rho}(\mathbf{w}_t; (\mathbf{f}_t(\mathbf{x}_t), y_t))}{\|\mathbf{w}_t \odot \exp(-\eta_{eg}\nabla_{\mathbf{w}}\ell^{\rho}(\mathbf{w}_t; (\mathbf{f}_t(\mathbf{x}_t), y_t))\|_1}, \tag{5.8}$$

where $\odot$ is element-wise product. It is easy to check that $\nabla_{\mathbf{w}}\ell^{\rho}(\mathbf{w}_t; (\mathbf{f}_t(\mathbf{x}_t), y_t)) = -\rho_t y_t \mathbf{f}_t(\mathbf{x}_t)$ when $\ell^{\rho}(\mathbf{w}; (\mathbf{f}(\mathbf{x}), y)) > 0$, and 0 otherwise, where ρ_t is set in the same manner as for learning a single kernel predictor. We refer to this approach as CSOMKC(EG).

OGD combination: Since CSOMKC(EG) learns a convex combination, we aim to increase the generality by learning the optimal cost-sensitive linear combination of multiple kernel predictors. Similar with the EG update (5.7), this can be cast into the following optimization:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|_2^2 + \eta_{ogd}\ell^{\rho}(\mathbf{w}, (\mathbf{f}_t(\mathbf{x}_t), y_t))$$

where $\mathbf{f}_t(\mathbf{x}_t)$ is the vector representing the predictions made by each individual kernel classifier. After replacing the loss function with its first order Taylor expansion, the update rule based on Online Gradient Descent can be derived as [134, 117]:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_{ogd}\nabla_{\mathbf{w}}\ell^{\rho}(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t)),$$

where the weights are updated only when the combined prediction suffers a loss, i.e., $\ell^{\rho}(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t)) > 0$; η_{ogd} represents the learning rate for the weight update of the combination; and ρ_t regulates the update to account for cost-sensitivity. This approach referred to as CSOMKC(OGD).

Both approaches are similar in the problem being addressed. However, EG uses multiplicative updates, whereas OGD uses additive updates. As a result, while EG may converge to a solution faster, in the long run OGD will outperform it. Both the approaches are outlined in Algorithm 8.

Algorithm 8 Cost-Sensitive Online Multiple Kernel Classification

```
INPUTS: Kernels: $k_i(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, i = 1, \dots, m$; Learning rates: $\eta > 0, \eta_{eg} > 0$; Cost Sensitive Parameter: $\rho > 0$
Initialization: $\mathbf{f}^1 = \mathbf{0}, \mathbf{w}^1 = \frac{1}{m}\mathbf{1}$
for $t = 1, 2, \dots$ do
  Receive an instance: $\mathbf{x}_t$
  Predict $\hat{y}_t = \mathrm{sign}\Big(\mathbf{w}_t \cdot \mathbf{f}_t(\mathbf{x}_t)\Big)$
  Receive the class label: $y_t$
  Set $\rho_t = \rho * \mathbb{I}(y_t = 1) + \mathbb{I}(y_t = -1)$
  for $i = 1, 2, \dots, m$ do
    Set $\ell^\rho(f_t^i; (\mathbf{x}_t, y_t)) = \rho_t \max(0, 1 - y_t f_t^i(\mathbf{x}_t))$
    if $\ell^\rho(f_t^i; (\mathbf{x}_t, y_t)) > 0$ then
      Update $f_{t+1}^i(\mathbf{x}) = f_t^i(\mathbf{x}) + \eta_i \rho_t y_t \kappa_i(\mathbf{x}_t, \mathbf{x})$
    end if
  end for
  Set $\ell^\rho(\mathbf{w}_t; (\mathbf{f}_t(\mathbf{x}_t), y_t)) = \rho_t \max(0, 1 - y_t \mathbf{w}_t \cdot \mathbf{f}_t(\mathbf{x}_t))$
  if $\ell^\rho(\mathbf{w}_t; (\mathbf{f}_t(\mathbf{x}_t), y_t)) > 0$ then
    Update $\mathbf{w}_{t+1} = \frac{\mathbf{w}_t \odot \exp(\eta_{eg} \rho_t y_t \mathbf{f}_t(\mathbf{x}_t))}{\|\mathbf{w}_t \odot \exp(\eta_{eg} \rho_t y_t \mathbf{f}_t(\mathbf{x}_t))\|_1}$ for update by EG OR
    Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_{ogd} \nabla_{\mathbf{w}} \ell^\rho(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t))$ for update by OGD
  end if
end for
```

5.3.4 Theoretical Analysis

In this section, we present the theoretical properties of the algorithm CSOMKC(EG). We derive the loss bound for Algorithm 8 when the kernel combination is learnt by EG algorithm. We assume $\kappa(\mathbf{x}, \mathbf{x}) \le 1$ for all κ and $\mathbf{x}$. We define the optimal regularized objective value for the kernel $\kappa_i(\cdot,\cdot)$ denoted by $O(\kappa_i, \ell^\rho, \mathcal{D})$ with respect to the dataset $\mathcal{D}$ as:

$$\min_{f_i \in \mathcal{H}_{\kappa_i}} \left(\sum_{t=1}^{T} \ell^\rho(f_i, (\mathbf{x}_t, y_t)) + \|f_i\|_{\mathcal{H}_{\kappa_i}} \sqrt{\rho^2 L_i^p + L_i^n} \right)$$

where $L_i^p = \sum_{y_t=1} \mathbb{I}(y_t f_i(\mathbf{x}_t) \le 1),\ L_i^n = \sum_{y_t=-1} \mathbb{I}(y_t f_i(\mathbf{x}_t) \le 1)$.

Lemma 2. *Assume $\|\mathbf{f}_t(\mathbf{x}_t)\|_\infty \le R$, and the CSOMKC(EG) algorithm is run with learning rate $\eta_{eg} = \sqrt{\frac{2\ln m}{R^2 T}}$ on a sequence of examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_T, y_T)\}$. Then for any combination of function $\mathbf{f}_t = \sum_{i=1}^m w^i f_t^i$,*

$\mathbf{w} \in \Delta$ *we have*

$$\sum_{t=1}^{T} \ell^{\rho}(\mathbf{w}_t, (\mathbf{f}(\mathbf{x}_t), y_t)) \leq \sum_{t=1}^{T} \ell^{\rho}(f_t, (\mathbf{x}_t, y_t)) + R\sqrt{\frac{T \ln m}{2}}.$$

Moreover, we have

$$\rho|\mathcal{M}_p| + |\mathcal{M}_n| \leq \min_{1 \leq i \leq m} O(\kappa_i, \ell^{\rho}, \mathcal{D}) + R\sqrt{\frac{T \ln m}{2}}.$$

Proof. The idea of this proof follows the principle of similar proof in [14]. We denote $l_t(\mathbf{w}_t) = \ell^{\rho}(\mathbf{w}_t, (\mathbf{f}(\mathbf{x}_t), y_t))$. Then, by convexity of l_t, we get

$$l_t(\mathbf{w}_t) - l_t(\mathbf{w}) \leq -(\mathbf{w} - \mathbf{w}_t) \cdot \nabla l_t(\mathbf{w}_t). \tag{5.9}$$

$$\begin{aligned}
&-(\mathbf{w} - \mathbf{w}_t) \cdot \mathbf{z} \\
&= -\mathbf{w} \cdot \mathbf{z} + \mathbf{w}_t \cdot \mathbf{z} - \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \\
&= -\mathbf{w} \cdot \mathbf{z} - \ln(\sum_{i=1}^{m} w_t^i e^{-z_i}) + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \\
&= \sum_{j=1}^{m} w^j \ln e^{-z_j} - \ln(\sum_{i=1}^{m} w_t^i e^{-z_i}) + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \\
&= \sum_{j=1}^{m} w^j \ln(\frac{1}{w_t^j} \frac{w_t^j e^{-z_j}}{\sum_{i=1}^{m} w_t^i e^{-z_i}}) + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \\
&= \sum_{j=1}^{m} w^j \ln \frac{w_{t+1}^j}{w_t^j} + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \\
&= D_{KL}(\mathbf{w} \| \mathbf{w}_t) - D_{KL}(\mathbf{w} \| \mathbf{w}_{t+1}) + \ln(\sum_{i=1}^{m} w_t^i e^{v_i}).
\end{aligned}$$

Plugging the above equality into the inequality (5.9) and summing over t, we get

$$\sum_{t=1}^{T} [l_t(\mathbf{w}_t) - l_t(\mathbf{w})] \leq \frac{1}{\eta_{eg}} [D_{KL}(\mathbf{w} \| \mathbf{w}_1) + \sum_{t=1}^{T} \ln(\sum_{i=1}^{m} w_t^i e^{v_i})],$$

by omitting $-D_{KL}(\mathbf{w} \| \mathbf{w}_{T+1})$. To bound the right hand side of the inequality, note that $D_{KL}(\mathbf{w} \| \mathbf{w}_1) \leq \ln m$, since $\mathbf{w}_1 = (1/m, \ldots, 1/m)$. We need bound the second term in the right hand side. Since $\|\mathbf{f}_t(\mathbf{x}_t)\|_\infty \leq R$, then $|z_i| \leq \eta_{eg} R$, and applying

Hoeffding's inequality:

$$\ln(\sum_{i=1}^{m} w_t^i e^{v_i}) \leq \eta_{eg}^2 R^2/2.$$

As a result, we get the following inequality,

$$\sum_{t=1}^{T}[l_t(\mathbf{w}_t) - l_t(\mathbf{w})] \leq \frac{\ln m}{\eta_{eg}} + \frac{\eta_{eg} R^2 T}{2} = R\sqrt{\frac{T \ln m}{2}}.$$

Re-arranging this concludes the first part of the theorem. If we set $w^i = 1$ and $w^j = 0$, for $j \neq i$, using $\ell^\rho(\mathbf{w}_t, (\mathbf{f}_t(\mathbf{x}_t), y_t)) \geq \rho_t$ when prediction is wrong, and combining with Lemma 1 of the paper [117], the above inequality will derive the second part. $\square$

Using this result, we now bound the *sum* incurred by CSOMKC.

Theorem 1. *After receiving a sequence of* T *training examples* $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$*, the weighted* $sum = \alpha(S_e) + (1-\alpha)(S_p)$ *achieved by Algorithm 8 for kernel combination update by EG, with* $\eta_i = \|f_i\|_{\mathcal{H}_{\kappa_i}} / \sqrt{\rho^2 M_i^p + M_i^n}$ *and* $\rho = \frac{\alpha T_n}{(1-\alpha)T_p}$*, is bounded as:*

$$sum \geq \frac{(1-\alpha)}{T_n}\left[\min_{1 \leq i \leq m} O(\kappa_i, \ell^\rho, \mathcal{D}) + R\sqrt{\frac{T \ln m}{2}}\right].$$

Proof. We know that $sum = 1 - \frac{1-\alpha}{|T_n|}\left[\frac{\alpha|T_n|}{(1-\alpha)|T_p|}|\mathcal{M}_p| + |\mathcal{M}_n|\right]$. Using $\rho = \frac{\alpha|T_n|}{(1-\alpha)|T_p|}$, and combining the above with Lemma 1 gives us the desired result. $\square$

We now derive a bound for the *cost* suffered, which unlike *sum* does not require the estimates of ratio $\frac{T_n}{T_p}$ (which may be unknown in advance). We set $\rho = \frac{1-c}{c}$, where $c \in (0, 1)$.

Theorem 2. *After receiving a sequence of* T *training examples* $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$*, the weighted* $cost = c(|\mathcal{M}_p|) + (1-c)(|\mathcal{M}_n|)$ *suffered by Algorithm 8 for kernel combination update by EG, with* $\eta_i = \|f_i\|_{\mathcal{H}_{\kappa_i}} / \sqrt{\rho^2 M_i^p + M_i^n}$ *and* $\rho = \frac{c}{1-c}$*,* *is bounded as:*

$$cost \leq (1-c)\left[\min_{1\leq i\leq m} O(\kappa_i, \ell^\rho, \mathcal{D}) + R\sqrt{\frac{T\ln m}{2}}\right].$$

Proof. From the definition of $cost$, we know that $cost = (1-c)(\frac{c}{1-c}|\mathcal{M}_p| + |\mathcal{M}_n|) = (1-c)(\rho|\mathcal{M}_p| + |\mathcal{M}_n|)$. Combining this with Lemma 1 proves this theorem. $\square$

Time Complexity: Traditional linear online algorithms execute in $O(T)$ where T is the number of instances. For an online kernel algorithm, in the worst case scenario, where every instance becomes a support vector, the algorithm would run in $O(T^2)$. However, applying budget techniques like the Randomized Budget Perceptron [13] reduces the running time to $O(BT)$ where B is the user-specified budget, and $B << T$. The multiple kernel variants with m kernels require time complexity of running m online kernel algorithms. Therefore, the time to run CSOMKC is in $O(mBT)$, i.e., the time complexity with budget approximations is linear in number of instances.

5.4 Experiments

We now present comprehensive empirical analysis of our proposed scheme, where we have evaluated algorithms' performance on imbalanced datasets, and anomaly detection tasks.

5.4.1 Datasets

We use a wide variety of datasets, across a wide spectrum of applications and varying number of instances, features, and imbalance ratios. All the datasets are publicly available and were retrieved from UCI repository, LIBSVM, and KDD Cup 2008. The datasets can be categorized into 6 regular imbalanced datasets, and 2 anomaly detection datasets. Among the imbalanced datasets we have: Spam and Webspam datasets which are self explanatory; Cod-rna is a bioinformatics dataset; Twitter dataset is about detecting buzz in social media, and is temporal in nature; Internet Ads is about predicting whether images in a given URL are ads or not; and Page-blocks attempts to classify page blocks into text or not. The anomaly detection datasets are KDD08 (from KDD Cup 2008 dataset on breast cancer); and Malware dataset built from Android Malware Genome Project which is about classifying apps as malware or not. The other details are given in Table 5.1.

Table 5.1: Details of the datasets used

Data ID	**Name of Dataset**	**Instances**	**Features**	$T_n : Tp$
Regular Imbalanced Datasets				
D1	Spam	4601	57	1.53
D2	Webspam	350000	254	1.54
D3	Cod-rna	59535	8	2.00
D4	Twitter	140707	77	4.07
D5	Internet Ads	3279	1556	6.14
D6	Page-blocks	21888	10	8.79
Highly Imbalanced Anomaly Detection Datasets				
D7	KDD08	102294	117	163.20
D8	Malware	208243	122	549.91

5.4.2 Kernels

Different kernels are suitable for different types of data. For example polynomial kernels which implicitly construct new polynomial features are more suited for NLP(among other tasks). Gaussian kernels have been the most widely used kernels for a variety of tasks. Additionally, depending on the data distribution, appropriate parameters need to be set, which is often done by validation techniques. To automatically select from a rich pool of kernels, we predefine a diverse set of 10 kernels which include three polynomial kernels $\kappa(x, y) = (x^T y)^p$ of degree parameter $p = 1, 2, 3, 4$, five RBF kernels ($\kappa(x, y) = e^{(\frac{-||x-y||^2}{2\sigma^2})}$) of kernel width parameter $\sigma = 2^{-2}, 2^{-1}, 2^0, 2^1, 2^2$, and a sigmoid kernel($\kappa(x, y) = \tanh(xy)$).

5.4.3 Algorithms Compared

Online Learning with Kernels suffers from an unbounded growth of support vectors. For large data, even with a good classifier, the number of support vectors keeps growing linearly. To make the computation realistically possible, a budget on the number of support vectors is required. We set a budget of 2000 support vectors per kernel classifier for all datasets with number of instances greater than $50,0000$, and apply the Randomized Budget Perceptron [13] by randomly discarding a support vector when the budget constraint is violated, and hence approximating the kernel predictions. This approximation is done for all algorithms that make kernel based predictions. In our experiments, OMKCSC-EG and OMKCSC-OGD are compared the following algorithms:

Linear Algorithms: We compare with the following linear algorithms:

1. Simple Linear Online Gradient Descent [134]
2. PAUM [74] Perceptron based method for uneven margins
3. CPA$_{PB}$ [21] which is a cost-sensitive variant of the popular online Passive Aggressive algorithms and
4. CSOL [117] that directly optimizes cost-sensitive measures.

Kernel Algorithms: We compare with the following online kernel methods:

1. Best Single Cost-Sensitive Kernel (CSC Kernel) determined by validation over first few samples of the data
2. Online Multiple Kernel Classification with hedge combination OMKC(H) [49]
3. OMKC with linear combination OMKC(OGD) [100]
4. Finally, we also compare with OMKCSC(U), where a uniform combination of the multiple cost-sensitive kernel predictors is used, i.e., our strategies of combining multiple kernels via EG or OGD are ignored.

All algorithms are evaluated on the basis of *sum* with $\alpha = 0.5$ (which is essentially the balanced accuracy); and *cost* with $c = 0.95$. For a comprehensive study, we also evaluate the results of sensitivity and specificity of each algorithm.

5.4.4 Parameters

There are 5 parameters required to be selected: Learning rate η for each kernel, cost-sensitive parameter ρ, and the combination learning rate for EG η_{eg} and for OGD η_{ogd}. For a fair comparison with OMKC, we set $\eta = 1$ for all algorithms. We set the combination parameters $\eta_{eg} = 0.1$, and $\eta_{ogd} = 0.1$ for all cases, and also perform sensitivity analysis for them. The cost-sensitive parameter ρ is set according to the objective being optimized. While trying to maximize *sum*, we set $\alpha = 0.5$, and accordingly ρ is set as $\rho = \frac{(1-\alpha)T_p}{\alpha T_n}$. While trying to minimize the *cost*, we set $c = 0.95$, and accordingly ρ is set as $\rho = \frac{1-c}{c}$.

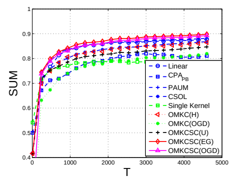
5.4.5 Results and Discussion

All results are reported as the average over multiple permutations, except the Twitter dataset which is temporal in nature (random permutations are meaningless). The details are in Table 5.2. Further, the *sum* and *cost* of all algorithms, as the number of instances grows can be visualized in Figure 5.1.

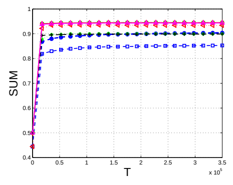
From Table 5.2, we see that our proposed CSOMKC(EG) and CSOMKC(OGD) almost always secure the highest *sum* (balanced accuracy), and the lowest *cost*. For *sum*, which in our case is a measure of balanced accuracy, CSOMKC(EG) and CSOMKC(OGD) get superior performances in all datasets with the exception CSOMKC(EG) in InternetAds. For the anomaly detection datasets, the proposed algorithms achieve excellent results, beating the benchmarks by a significant margin. In *cost* performance, with the exception of CSOMKC(EG) in InternetAds, we get a significant outperformance of the proposed techniques in all cases. However, for all other cases, the performance is phenomenal. InternetAds has a very high dimensionality, and with the usage of multiple kernels, suffers from a relatively slow convergence for a small number of instances. An interesting insight can be drawn from 5.1. Our proposed scheme usually picks up the best pattern, and achieves superior performance right from the very beginning. It is also worth noting that in some cases, single kernels give a good performance on the first few samples of the data, but eventually are not able to match OMKCSC algorithms. This demonstrates difficulty in kernel selection by validation, and hence it is imperative to have a dynamic technique that can automatically choose a combination of multiple kernels.

Linear vs Kernel Methods: The empirical results show that the introduction of kernels (and subsequently multiple kernels) have significantly increased the learning capacity of our models. In many cases, the balanced accuracy has improved by over 5% by using kernel methods as compared to the state of the art linear models. In the optimization of *cost*, have caused a great reduction of the cost e.g. in the Malware dataset, the cost suffered by kernel methods is a mere 7% of the cost suffered by the best linear methods.

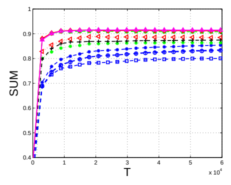
Comparisons between kernel methods: Among the kernel methods our proposed techniques out performed the others, due to their ability to learn multiple cost-sensitive kernel prediction models, followed by their cost-sensitive combination. Firstly, it should be noted that the addition of multiple kernels has in fact helped increase the predictive power of the model as compared to one single kernel. However, this raises a further question: which of the algorithms CSOMKC(EG) and CSOMKC(OGD) is more suitable? CSOMKC(EG) has a a more limited predictive power as it learns only a convex combination of multiple kernels (as compared to CSOMKC(OGD). Further, if the data is described by a single kernel function, CSOMKC(EG) is able to quickly converge to that kernel predictor via multiplicative updates; but if the optimal prediction depends on a combination of several kernel functions, CSOMKC(OGD) slowly approaches the best solution, and will probably outperform CSOMKC(EG) in the long run. It should be noted that the results are very robust, as indicated by a very small standard deviation in most cases. In fact, in several cases, due to the low standard deviation, upon rounding up, it is reported as 0.



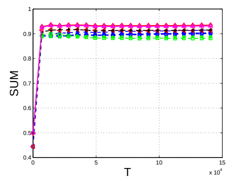
(a) SUM (Spam)



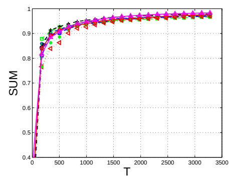
(b) SUM (Webspam)



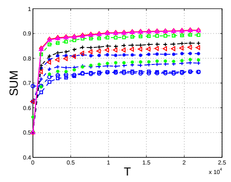
(c) SUM (COD-RNA)



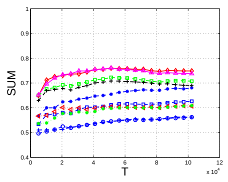
(d) SUM (Twitter)



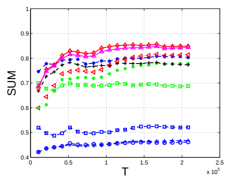
(e) SUM (InternetAds)



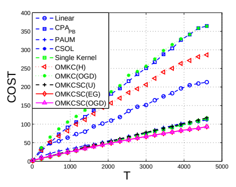
(f) SUM (Page-blocks)



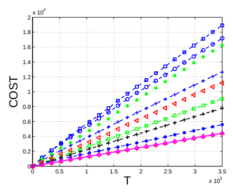
(g) SUM (KDD08)



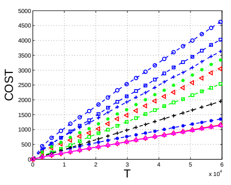
(h) SUM (Malware)



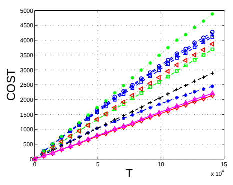
(i) COST (Spam)



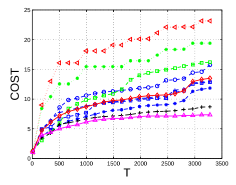
(j) COST (Webspam)



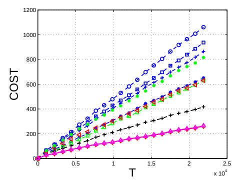
(k) COST (COD-RNA)



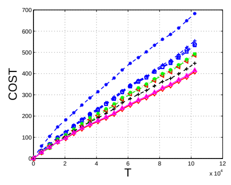
(l) COST (Twitter)



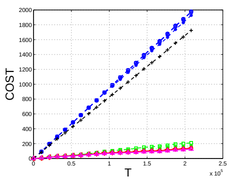
(m) COST (InternetAds)



(n) COST (Page-blocks)



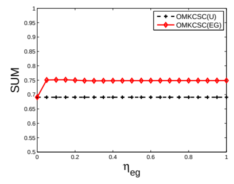
(o) COST (KDD08)



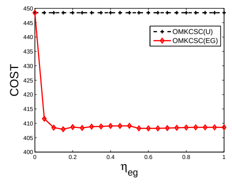
(p) COST (Malware)

Figure 5.1: Cost of different algorithms as the number of instances increases

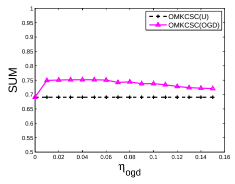
Sensitivity to learning rate parameters: We also analyzed the sensitivity of CSOMKC(EG) and CSOMKC(OGD) to their learning rates η_{eg} and η_{ogd} respectively. A sample of this analysis on the KDD08 dataset can be seen in Figure 5.2. As expected, both converge to the performance of OMKCSC(U) when the learning rates are set to 0. For a wide variety of other learning rates, CSOMKC(EG) and CSOMKC(OGD) substantially outperform OMKCSC(U) which for the KDD08 dataset is the next best performer. CSOMKC(EG) is more robust to the parameter choice and gives consistent results across a wide range, whereas CSOMKC(OGD) gives a good performance for small values of η_{ogd}, and when the learning rate is very large, its performance degrades. In our experiments, we had set $\eta_{ogd} = 0.1$ which is a conservative choice. A carefully chosen learning rate would further enhance the performance of CSOMKC(OGD).



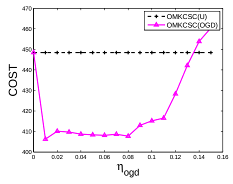
(a) CSOMKC(EG) SUM



(b) CSOMKC(EG) COST



(c) CSOMKC(OGD) SUM



(d) CSOMKC(OGD) COST

Figure 5.2: Parameter Sensitivity Analysis

5.5 Conclusion

In this chapter, we motivated the need for scalable techniques that can learn complex nonlinear patterns in imbalanced data and proposed a novel scheme of Cost-Sensitive Online Multiple Kernel Classification, which dynamically explores a diverse set of predefined kernel functions, and simultaneously learns both the kernel predictions and their optimal combinations. Both the kernel predictors and their combinations are learnt in a cost-sensitive manner. The kernel predictors are learnt by gradient descent, and for the kernel combinations, we demonstrated two approaches - first by exponentiated gradient, and second by online gradient descent. We discussed the application of the proposed techniques to online anomaly detection. We derived theoretical properties of the algorithms and have conducted extensive empirical analysis on imbalanced datasets and anomaly detection datasets. Our proposed techniques significantly outperformed several state of the art methods designed for cost-sensitive classification.

Table 5.2: Evaluation of all the algorithms on all datasets based on the *sum* and *cost*.

SUM performance of algorithms				
	Spam	Webspam	Cod-rna	Twitter
Linear	0.879 ± 0.002	0.903 ± 0.000	0.835 ± 0.003	0.904 ± 0.000
CPA$_{PB}$	0.809 ± 0.002	0.853 ± 0.000	0.801 ± 0.000	0.901 ± 0.000
PAUM	0.883 ± 0.002	0.902 ± 0.000	0.836 ± 0.002	0.904 ± 0.000
CSOL	0.870 ± 0.003	0.905 ± 0.000	0.854 ± 0.001	0.915 ± 0.000
CSC Kernel	0.846 ± 0.083	**0.945 ± 0.001**	0.910 ± 0.003	0.882 ± 0.000
OMKC(H)	0.866 ± 0.002	0.934 ± 0.000	0.885 ± 0.001	0.913 ± 0.000
OMKC(OGD)	0.825 ± 0.008	0.900 ± 0.000	0.863 ± 0.002	0.889 ± 0.000
CSOMKC(U)	0.850 ± 0.005	0.899 ± 0.000	0.874 ± 0.000	0.915 ± 0.000
CSOMKC(EG)	**0.902 ± 0.004**	**0.945 ± 0.001**	**0.912 ± 0.000**	**0.935 ± 0.000**
CSOMKC(OGD)	**0.896 ± 0.005**	**0.944 ± 0.001**	**0.914 ± 0.001**	**0.932 ± 0.000**
	Internet-Ads	Page-Blocks	KDD08	Malware
Linear	0.970 ± 0.001	0.743 ± 0.003	0.564 ± 0.003	0.466 ± 0.007
CPA$_{PB}$	0.971 ± 0.000	0.747 ± 0.003	0.625 ± 0.001	0.524 ± 0.005
PAUM	0.974 ± 0.002	0.780 ± 0.000	0.558 ± 0.006	0.472 ± 0.005
CSOL	0.976 ± 0.002	0.819 ± 0.001	0.684 ± 0.006	0.804 ± 0.001
CSC Kernel	0.967 ± 0.096	0.904 ± 0.014	0.649 ± 0.083	0.688 ± 0.000
OMKC(H)	0.972 ± 0.002	0.845 ± 0.002	0.612 ± 0.007	0.814 ± 0.000
OMKC(OGD)	0.975 ± 0.001	0.793 ± 0.001	0.616 ± 0.010	0.781 ± 0.003
CSOMKC(U)	0.979 ± 0.000	0.859 ± 0.002	0.696 ± 0.008	0.771 ± 0.004
CSOMKC(EG)	0.976 ± 0.000	**0.913 ± 0.000**	**0.748 ± 0.001**	**0.836 ± 0.018**
CSOMKC(OGD)	**0.985 ± 0.002**	**0.912 ± 0.001**	0.733 ± 0.007	**0.84 ± 0.007**

COST performance of algorithms				
	Spam	Webspam	Cod-rna	Twitter
Linear	205.2 ± 11.172	17233.975 ± 88.282	4560.625 ± 86.373	4275.9 ± 0.000
CPA$_{PB}$	367.625 ± 4.914	18856.15 ± 81.388	4002.85 ± 31.466	4112.05 ± 0.000
PAUM	113.175 ± 2.369	12611.25 ± 57.134	3669.075 ± 55.402	4177.05 ± 0.000
CSOL	109.35 ± 3.041	5601.2 ± 29.628	1349.675 ± 3.147	2449 ± 0.000
CSC Kernel	114.05 ± 1.131	9055.8 ± 3.041	2489.375 ± 63.534	3690 ± 0.000
OMKC(H)	282.725 ± 5.409	11183.925 ± 8.309	3055.125 ± 15.167	3872.65 ± 0.000
OMKC(OGD)	351.25 ± 17.607	16265.4 ± 7.637	3391.325 ± 71.17	4883.6 ± 0.000
CSOMKC(U)	116.45 ± 1.131	7805.025 ± 36.946	1950.975 ± 4.066	2889.25 ± 0.000
CSOMKC(EG)	**95.725 ± 3.359**	**4407.325 ± 29.097**	1157.225 ± 0.530	**2141 ± 0.000**
CSOMKC(OGD)	**95.975 ± 4.207**	**4412.903 ± 30.618**	**1141.525 ± 0.813**	2210.7 ± 0.000
	Internet-ads	Page-blocks	KDD08	Malware
Linear	15.275 ± 0.672	1069.425 ± 12.127	535.9 ± 3.323	1973.6 ± 4.596
CPA$_{PB}$	13.45 ± 0.849	940.525 ± 5.48	528.975 ± 6.116	1977.8 ± 6.223
PAUM	11.875 ± 1.591	848.525 ± 22.451	550.175 ± 5.551	1970.95 ± 5.091
CSOL	10.9 ± 1.344	645.85 ± 6.435	674.975 ± 11.208	1924.85 ± 4.525
CSC Kernel	14.125 ± 2.934	445.35 ± 260.569	607.175 ± 165.852	208.525 ± 3.076
OMKC(H)	22.075 ± 1.52	623.4 ± 9.546	480.575 ± 8.945	141.925 ± 0.247
OMKC(OGD)	18.85 ± 0.778	819.575 ± 5.48	483.775 ± 12.763	169.1 ± 2.546
CSOMKC(U)	7.75 ± 1.344	425.425 ± 11.49	444.05 ± 6.223	1725.525 ± 2.157
CSOMKC(EG)	12.5 ± 1.414	**262.15 ± 0.919**	**419.6 ± 15.556**	136.675 ± 4.137
CSOMKC(OGD)	**6.000 ± 1.909**	266.4 ± 2.192	**421.625 ± 9.016**	**129.3 ± 4.313**

Chapter 6

Online Deep Learning

Deep Neural Networks (DNNs) are typically trained by backpropagation in a batch learning setting, which requires the entire training data to be made available prior to the learning task. This is not scalable for many real-world scenarios where new data arrives sequentially in a stream form. We aim to address an open challenge of "Online Deep Learning" (ODL) for learning DNNs on the fly in an online setting. Unlike traditional online learning that often optimizes some convex objective function with respect to a shallow model (e.g., a linear/kernel-based hypothesis), ODL is significantly more challenging since the optimization of the DNN objective function is non-convex, and regular backpropagation does not work well in practice, especially for online learning settings. In this chapter, we present a new online deep learning framework that attempts to tackle the challenges by learning DNN models of adaptive depth from a sequence of training data in an online learning setting. In particular, we propose a novel Hedge Backpropagation (HBP) method for online updating the parameters of DNN effectively, and validate the efficacy of our method on large-scale data sets, including both stationary and concept drifting scenarios.

6.1 Introduction

Recent years have witnessed tremendous success of deep learning techniques in a wide range of applications [68, 8, 9, 63, 46]. Learning Deep Neural Networks (DNN) faces many challenges, including (but not limited to) vanishing gradient, diminishing feature reuse [112], saddle points (and local minima) [18, 28], immense number of parameters to be tuned, internal covariate shift during training [56], difficulties in choosing a good regularizer, choosing hyperparameters, etc. Despite many promising advances [89, 56, 46, 112], etc., which are designed to address specific problems for optimizing deep neural networks, most of these existing approaches assume that the DNN models are trained in a batch learning setting which requires the entire training data set to be made available prior to the learning task. This is not possible for many real world tasks where data arrives sequentially in a stream, and may be too large to be stored in memory. Moreover, the data may exhibit concept drift [39]. Thus, a more desired option is to learn the models in an online setting.

Unlike batch learning, online learning [134, 14] represents a class of learning algorithms that learn to optimize predictive models over a stream of data instances in a sequential manner. The nature of on-the-fly learning makes online learning highly scalable and memory efficient. However, most existing online learning algorithms are designed to learn shallow models (e.g., linear or kernel methods [99, 134, 21, 61, 49]) with online convex optimization, which cannot learn complex nonlinear functions in complicated application scenarios.

In this work, we attempt to bridge the gap between deep learning and online learning by addressing the open problem of "Online Deep Learning" (ODL) — how to learn Deep Neural Networks (DNNs) from data streams in an online setting. A possible way to do ODL is to put the process of training DNNs online by directly applying a standard Backpropagation training on only a single instance at each online round. Such an approach is simple but falls short due to some critical limitations in practice. One key challenge is how to choose a proper model capacity (e.g., depth of the network) before starting to learn the DNN online. If the model is too complex (e.g., very deep networks), the learning process will converge too slowly (vanishing gradient and diminishing feature reuse), thus losing the desired property of online learning. On the other extreme, if the model is too simple, the learning capacity will be too restricted, and without the power of depth, it would be difficult to learn complex patterns. In batch learning literature, a common way to address this issue is to do model selection on validation data. Unfortunately, it is not realistic to have validation data in online settings, and is thus infeasible to apply traditional model selection in online learning scenarios. In this work, we present a novel framework for online deep learning, which is able to learn DNN models from data streams sequentially, and more importantly, is able to adapt its model capacity from simple to complex over time, nicely combining the merits of both online learning and deep learning.

We aim to devise an online learning algorithm that is able to start with a shallow network that enjoys fast convergence; then gradually switch to a deeper model (meanwhile sharing certain knowledge with the shallow ones) automatically when more data has been received to learn more complex hypotheses, and effectively improve online predictive performance by adapting the capacity of DNNs. To achieve this, we first amend the existing DNN architecture by attaching every hidden layer representation to an output classifier. Then, instead of using a standard Backpropagation, we propose a novel *Hedge Backpropagation* method, which evaluates the online performance of every output classifier at each online round, and extends the Backpropagation algorithm to train the DNNs online by exploiting the classifiers of different depths with the Hedge algorithm [37]. This allows us to train DNN of adaptive capacity meanwhile enabling knowledge sharing between shallow and deep networks.

6.2 Related Work

6.2.1 Online Learning

Online Learning represents a family of scalable and efficient algorithms that learn to update models from data streams sequentially [14, 105, 51]. Many techniques are based on maximum-margin classification, starting from Perceptron [99] to Online Gradient Descent [134], Passive Aggressive (PA) algorithms [21], Confidence-Weighted (CW) Algorithms, [33] etc. These are primarily designed to learn linear models. Online Learning with kernels [61] offered a solution for online learning with nonlinear models. These methods received substantial interest from the community, and models of higher capacity such as Online Multiple Kernel Learning were developed [58, 49, 100]. While these models learn nonlinearity, they are still shallow models. Moreover, deciding the number and type of kernels is non-trivial; and these methods are not explicitly designed to *learn* a feature representation.

Online Learning can be directly applied to DNNs ("online backpropagation") but they suffer from convergence issues, such as vanishing gradient and diminishing feature reuse. Moreover, the optimal depth to be used for the network is usually unknown, and cannot be validated easily in the online setting. Further, networks of different depth would be suitable for different number of instances to be processed, e.g., for small number of instances - a quick convergence would be of high priority, and thus shallow networks would be preferred, whereas, for a large number of instances, the long run performance would be enhanced by using a deeper network. There have been attempts at making deep learning compatible with online learning [132, 70]. However, they operate via a sliding window approach with a (mini)batch training stage, making them unsuitable for a streaming data setting.

6.2.2 Deep Learning

Due to the difficulty in training deep networks, there has been a large body of emerging works adopting the principle of ”shallow to deep”. This is similar to the principle we adopt in our work. This approach exploits the intuition that shallow models converge faster than deeper models, and this idea has been executed in several ways. Some approaches do this explicitly by Growing of Networks via the function preservation principle [16, 119], where the (student) network of higher capacity is guaranteed to be at least as good as the shallower (teacher) network. Other approaches perform this more implicitly by modifying the network architecture and objective functions to enable the network to allow the input to flow through the network, and slowly adapt to deep representation learning, e.g., Highway Nets[112], Residual Nets[46], Stochastic Depth Networks [54] and Fractal Nets [66].

However, they are all designed to optimize the loss function based on the output obtained from the deepest layer. Despite the improved batch convergence, they cannot yield good online performances (particularly for the instances observed in early part of the stream), as the inference made by the deepest layer requires substantial time for convergence. For the online setting, such existing deep learning techniques could be trivially beaten by a very shallow network. Deeply Supervised Nets [69], shares a similar architecture as ours, which uses companion objectives at every layer to address vanishing gradient and to learn more discriminative features at shallow layers. However, the weights of companions are set heuristically, where the primary goal is to optimize the classification performance based on features learnt by the deepest hidden layer, making it suitable only for batch settings, which suffers from the same drawbacks as others.

Recent years have also witnessed efforts in learning the architecture of the neural networks [111, 1], which incorporate architecture hyperparameters into the optimization objective. Starting from an overcomplete network, they use regularizers that help in eliminating neurons from the network. For example, [1] use a group sparsity regularization to reduce the width of the network. [135] use reinforcement learning to search for the optimal architecture. Our proposed technique is related to these in the sense the we are adapting the depth of the network during the learning phase, making the network learn the appropriate capacity as and when it observes more data. Unlike other model selection techniques, our method is designed to operate in a pure online learning setting.

6.3 Online Deep Learning

6.3.1 Problem Setting

Without loss of generality, consider an online classification task. The goal of online deep learning is to learn a function $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^C$ based on a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, that arrive sequentially, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in \{0,1\}^C$ is the class label assigned to $\mathbf{x}_t$ and C is the number of classes. The prediction is denoted by $\hat{y}_t$, and the performance of the learnt functions are usually evaluated based on the cumulative prediction error: $\epsilon_T = \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}_{(\hat{y}_t \neq y_t)}$, where $\mathbb{I}$ is the indicator function resulting in 1 if the condition is true, and 0 otherwise. To minimize the classification error over the sequence of T instances, a loss function (e.g., squared loss, cross-entropy, etc.) is often chosen for minimization. In every online iteration, when an instance $\mathbf{x}_t$ is observed, and the model makes a prediction, the environment then reveals the ground truth of the class label, and finally the learner makes an update to the model (e.g., using online gradient descent).

6.3.2 Backpropagation

For typical online learning algorithms, the prediction function $\mathbf{F}$ is either a linear or kernel-based model. In the case of Deep Neural Networks (DNN), it is a set of stacked linear transformations, each followed by a nonlinear activation. Given an input $\mathbf{x} \in \mathbb{R}^d$, the prediction function of DNN with L hidden layers $(\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)})$ is recursively given by:

$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= \text{softmax}(W^{(L+1)}\mathbf{h}^{(L)}) \quad \text{where} \\ \mathbf{h}^{(l)} &= \sigma(W^{(l)}\mathbf{h}^{(l-1)}) \quad \forall l = 1, \ldots, L \\ \mathbf{h}^{(0)} &= \mathbf{x} \end{aligned}$$

where σ is an activation function, e.g., sigmoid, tanh, ReLU, etc. This equation represents a feedforward step of a neural network. The hidden layers $\mathbf{h}^{(l)}$ are the feature representations learnt during the training procedure. To train a model with such a configuration, we use the cross-entropy loss function denoted by $\mathcal{L}(\mathbf{F}(\mathbf{x}), y)$. We aim to estimate the optimal model parameters W_i for $i = 1, \ldots (L + 1)$ by applying Online Gradient Descent (OGD) on this loss function. Following the online learning setting, the update of the model in each iteration by OGD is given by:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \nabla_{W_t^{(l)}} \mathcal{L}(\mathbf{F}(\mathbf{x}_t), y_t) \qquad \forall l = 1, \ldots, L + 1$$

where η is the learning rate. Using backpropagation, the chain rule of differentiation is applied to compute the gradient of the loss with respect to $W^{(l)}$ for $l \leq L$. This is Online Backpropagation.

Unfortunately, using such a model for an online learning (i.e. Online Backpropagation) task faces several issues with convergence. Most notably: (i) For such models, a fixed depth of the neural network has to be decided a priori, and this cannot be changed during the training process. This is problematic, as determining the depth is a difficult task. Moreover, in an online setting, different depths may be suitable for a different number of instances to be processed, e.g. because of convergence reasons, shallow networks maybe preferred for small number of instances, and deeper networks for large number of instances. Our aim is to exploit the fast convergence of shallow networks at the initial stages, and the power of deep representation at a later stage; (ii) vanishing gradient is well noted problem that slows down learning. This is even more critical in an online setting, where the model needs to make predictions and learn simultaneously; (iii) diminishing feature reuse, according to which many useful features are lost in the feedforward stage of the prediction. This again is very critical for online learning, where it is imperative to quickly find the important features, so as to not suffer from poor performance for the initial training instances.

To address these issues, we design a novel training scheme for *Online Deep Learning* through a Hedging strategy for Deep Neural Networks: Hedge Backpropagation (HBP).

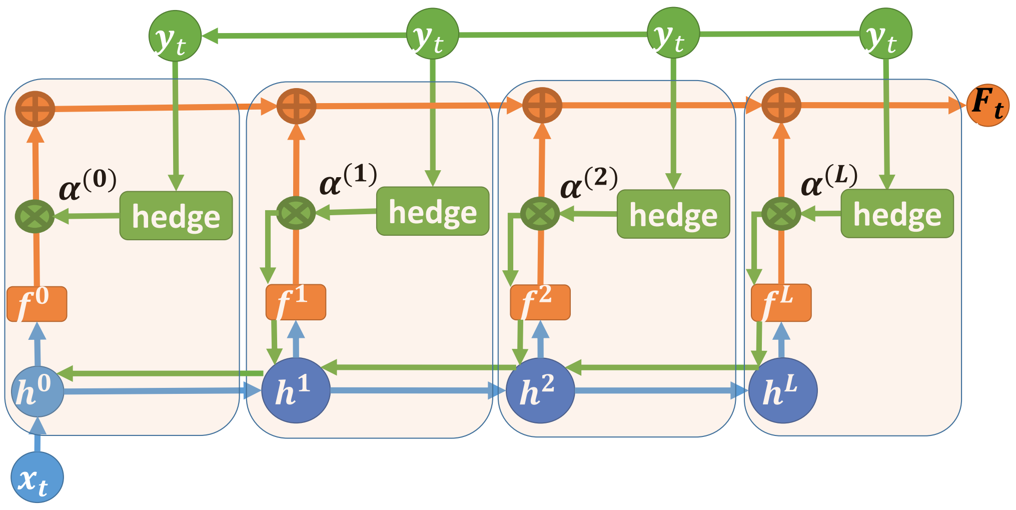


Figure 6.1: Illustration of the proposed Online Deep Learning framework using Hedge Backpropagation (HBP). The blue lines represent feedforward flow for computing hidden layer features. The orange lines indicate softmax output followed by the hedging combination at prediction time. The green lines indicate the online updating flows with the hedge backpropagation approach.

6.3.3 Hedge Backpropagation (HBP)

Figure 6.1 gives an overview of the proposed online deep learning framework, which illustrates the feedforward and backprop flows for a Hedged Deep Neural Network using the proposed Hedge Backpropagation.

Consider a deep neural network with L hidden layers (i.e. the maximum capacity of the network that can be learnt is one with L hidden layers), the prediction function for the proposed Hedged Deep Neural Network is given by:

$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= \sum_{l=0}^{L} \alpha^{(l)} \mathbf{f}^{(l)} \quad \text{where} \qquad (6.1) \\ \mathbf{f}^{(l)} &= \operatorname{softmax}(\mathbf{h}^{(l)} \Theta^{(l)}),\ \forall l = 0, \ldots, L \\ \mathbf{h}^{(l)} &= \sigma(W^{(l)} \mathbf{h}^{(l-1)}),\ \forall l = 1, \ldots, L \\ \mathbf{h}^{(0)} &= \mathbf{x} \end{aligned}$$

Here we have designed a new architecture, and introduced two sets of new parameters $\Theta^{(l)}$ (parameters for $\mathbf{f}^{(l)}$) and α, that have to be learnt. Unlike the original network, in which the final prediction is given by a classifier using the feature representation $\mathbf{h}^{(L)}$, here the prediction is weighted combination of classifiers learnt using the feature representations from $\mathbf{h}^{(0)}, \ldots, \mathbf{h}^{(L)}$. Each of these classifiers $\mathbf{f}^{(l)}$ is parameterized by $\Theta^{(l)}$. Note that there are a total of $L + 1$ classifiers. The final prediction of this model is a weighted combination of the predictions of all these classifiers, where the weight of each classifier is denoted by $\alpha^{(l)}$, and the loss suffered by the model is $\mathcal{L}(\mathbf{F}(\mathbf{x}), y) = \sum_{l=0}^{L} \alpha^{(l)} \mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}), y)$. During the online learning procedure, we need to learn $\alpha^{(l)}$, $\Theta^{(l)}$ and $W^{(l)}$.

We propose to learn $\alpha^{(l)}$ using the Hedge Algorithm [37]. At the first iteration, all the weights α are uniformly distributed, i.e., $\alpha^{(l)} = \frac{1}{L+1}, l = 0, \ldots, L$. At every iteration, the classifier $\mathbf{f}^{(l)}$ makes a prediction $\hat{y_t}^{(l)}$. When the ground truth is revealed, the classifier's weight is updated based on the loss suffered by the classifier. This is given by:

$$\alpha_{t+1}^{(l)} \leftarrow \alpha_t^{(l)} \beta^{\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}), y)}$$

where $\beta \in (0, 1)$ is the discount rate parameter, and $\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}), y) \in (0, 1)$ [37]. Thus, a classifier's weight is discounted by a factor of $\beta^{\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}), y)}$ in every iteration. At the end of every round, the weights α are normalized such that $\sum_l \alpha_t^{(l)} = 1$.

Learning the parameters $\Theta^{(l)}$ for all the classifiers can be done via online gradient descent [134], where the input to the l^{th} classifier is $\mathbf{h}^{(l)}$. This is similar to the update of the weights of the output layer in the original feedforward networks. This update is given by:

$$\Theta_{t+1}^{(l)} \leftarrow \Theta_t^{(l)} - \eta \nabla_{\Theta_t^{(l)}} \mathcal{L}(\mathbf{F}(\mathbf{x}_t, y_t)) = \Theta_t^{(l)} - \eta \alpha^{(l)} \nabla_{\Theta_t^{(l)}} \mathcal{L}(\mathbf{f}^{(l)}, y_t)$$

Updating the feature representation parameters $W^{(l)}$ is more tricky. Unlike the original backpropagation scheme, where the error derivatives are backpropagated from the output layer, here, the error derivatives are backpropagated from *every* classifier $\mathbf{f}^{(l)}$. Thus, using the adaptive loss function $\mathcal{L}(\mathbf{F}(\mathbf{x}), y) = \sum_{l=0}^{L} \alpha^{(l)} \mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}), y)$ and applying OGD rule, the update rule for $W^{(l)}$ is given by:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \sum_{j=l}^{L} \alpha^{(j)} \nabla_{W^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t)$$

where $\nabla_{W^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t)$ is computed via backpropagation from error derivatives of $\mathbf{f}^{(j)}$. Note that the summation (in the gradient term) starts at $j = l$ because the shallower classifiers do not depend on $W^{(l)}$ for making predictions. In effect, we are computing the gradient of the final prediction with respect to the backpropagated derivatives of a predictor at every depth weighted by $\alpha^{(l)}$ (which is an indicator of the performance of the classifier). Hedge enjoys a regret of $R_T \leq \sqrt{T \ln N}$, where N is the number of experts [38], which in our case is the depth of the network. This gives an effective model selection approach to adapt to the optimal network depth, automatically during online learning.

Based on the intuition that shallower models tend to converge faster than deeper models [16, 66, 45], using the hedging strategy would lower the α weights of the deeper classifiers to a very small value (due to poor initial performance as compared to shallower classifiers), which would consequently affect the update in Eq. (6.3.3), and result in deeper classifiers having very slow learning. To alleviate this, we introduce a *smoothing* parameter $s \in (0,1)$ which is used to set a minimum weight for each classifier. After the weight update of the classifiers in each iteration, the weights are set as: $\alpha^{(l)} \leftarrow \max\left(\alpha^{(l)}, \frac{s}{L}\right)$ This maintains a minimum weight for a classifier of every depth and helps us achieve a tradeoff between exploration and exploitation. s encourages all classifiers at every depth to have an impact on the backprop update (allowing exploring high capacity deep classifiers, and enabling deep classifiers to perform as good as shallower classifiers), and simultaneously, hedging the model exploits the best performing classifier. Similar strategies have been used in Multi-arm bandit setting, and online learning with expert advice to trade off exploration and exploitation [4, 49].

The entire algorithm of Online Deep Learning using Hedge Backpropagation is outlined below.

Algorithm 9 Online Deep Learning (ODL) using HBP

```
INPUTS: Discounting Parameter: $\beta \in (0,1)$;
Learning rate Parameter: $\eta$; Smoothing Parameter: $s$
**Initialize**: $\mathbf{F}(\mathbf{x}) =$ DNN with $L$ hidden layers and $L+1$ classifiers $\mathbf{f}^{(l)}, \forall l = 0,\dots,L$; $\alpha^{(l)} = \frac{1}{L+1}, \forall l = 0,\dots,L$
  **for** t = 1,...,T **do**
    Receive instance: $\mathbf{x}_t$
    Predict $\hat{y}_t = \mathbf{F}_t(\mathbf{x}_t) = \sum_{l=0}^{L} \alpha_t^{(l)} \mathbf{f}_t^{(l)}$ as per Eq. (6.1)
    Reveal true value $y_t$
    Set $\mathcal{L}_t^{(l)} = \mathcal{L}(\mathbf{f}_t^{(l)}(\mathbf{x}_t), y_t), \forall l, \dots, L$;
    Update $\Theta_{t+1}^{(l)}, \forall l = 0,\dots,L$ as per Eq. (6.3.3);
    Update $W_{t+1}^{(l)}, \forall l = 1,\dots,L$ as per Eq. (6.3.3);
    Update $\alpha_{t+1}^{(l)} = \alpha_t^{(l)} \beta^{\mathcal{L}_t^{(l)}}, \forall l = 0,\dots,L$;
    Smoothing $\alpha_{t+1}^{(l)} = \max(\alpha_{t+1}^{(l)}, \frac{s}{L}), \forall l = 0,\dots,L$ ;
    Normalize $\alpha_{t+1}^{(l)} = \frac{\alpha_{t+1}^{(l)}}{Z_t}$ where $Z_t = \sum_{l=0}^{L} \alpha_{t+1}^{(l)}$
  **end for**
```

6.3.4 Discussion

HBP has the following properties: (i) it identifies a neural network of an appropriate depth based on the performance of the classifier at that depth. This is a form of Online Learning with Expert Advice,[14], where the experts are DNNs of varying depth, making the DNN robust to depth. (ii) it offers a good initialization for deeper networks, which are encouraged to match the performance of shallower networks (if unable to beat them). This facilitates knowledge transfer from shallow to deeper networks ([16, 119]), thus simulating student-teacher learning; (iii) it makes the learning robust to vanishing gradient and diminishing feature reuse by using a multi-depth architecture where gradients are backpropagated from shallower classifiers, *and* the low level features are used for the final prediction (by hedge weighted prediction); (iv) it can be viewed as an ensemble of multi-depth networks which are competing and collaborating to improve the final prediction performance. The competition is induced by Hedging , whereas the collaboration is induced by sharing feature representations; (v) This allows our algorithms to continuously learn and adapt as and when it sees more data, enabling a form of life-long learning [70]; and (vi) In concept drifting scenarios [39], traditional online backpropagation would suffer from slow convergence for deep networks (when the concepts would change), whereas, HBP is able to adapt quickly due to hedging. (vii) Our work is also similar to LSTMs [47] in appearance of the learning architecture, however, LSTMs aim to backpropagate through time, while our method backpropagates through depth. This way HBP learns the appropriate depth capacity.

6.4 Experiments

6.4.1 Datasets

Table 6.1: Datasets

Data	#Features	#Instances	Type
Higgs	28	5m	Stationary
Susy	18	5m	Stationary
i-mnist	784	5m	Stationary
Syn8	50	5m	Stationary
CD1	50	4.5m	Concept Drift
CD2	50	4.5m	Concept Drift

We consider several large scale datasets. Higgs and Susy are Physics datasets from UCI repository. For Higgs, we sampled 5 million instances for our analysis. We generated 5 million instances from Infinite MNIST generator [77]. We also evaluated on 3 synthetic datasets. The first (Syn8) is generated from a randomly initialized DNN comprising 8-hidden layers (of width 100 each). The other two are concept drift datasets CD1 and CD2. In CD1, 2 concepts (C1 and C2), appear in the form C1-C2-C1, with each segment comprising a third of the data stream. Both C1 and C2 were generated from a 8-hidden layer network. CD2 has 3 concepts with C1-C2-C3, where C1 and C3 are generated from a 8-hidden layer network, and C2 from a shallower 6-hidden layer network. Other details are summarized in Table 6.1.

6.4.2 Performance Variation with Depth: Limitations of traditional Online BP

First we demonstrate the difficulty of DNN model selection in the online setting. We compare the error rate of DNNs of varying depth, in different segments of the data. All models were trained online, and we evaluate their performance in different windows (or stages) of the learning process. See Table 6.2. In the first 0.5% of the data, the shallowest network obtains the best performance indicating faster convergence - which would indicate that we should use the shallow network for the task. In the segment from [10-15]%, a 4-layer DNN seems to have the best performance in most cases. And in the segment from [60-80]% of the data, an 8-layer network gives a better performance. This indicates that deeper networks took a longer time to converge, but at a later stage gave a better performance. Looking at the final error, it does not give us conclusive evidence of what depth of network would be the most suitable. Furthermore, if the datastream had more instances, a deeper network may have given an overall better performance. This demonstrates the difficulty in model selection for learning DNNs online, where typical validation techniques are ineffective. Ideally we want to exploit the convergence of the shallow DNNs in the beginning and the power of deeper representations later.

Table 6.2: Online error rate of DNNs of varying Depth. All models were trained online $t = 1, \ldots, T$. We evaluate the performance in different segments of the data. L is the number of layers in the DNN.

	Final Error			**[0-0.5]% Error**			**[10-15]% Error**			**[60-80]% Error**		
L	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**
3	0.272	0.201	0.3936	**0.358**	**0.215**	**0.426**	0.279	**0.202**	0.400	0.266	0.200	0.390
4	0.268	**0.201**	**0.392**	0.372	0.219	0.433	**0.277**	**0.203**	**0.398**	0.261	0.200	**0.387**
8	**0.268**	0.201	0.3936	0.380	0.221	0.452	0.279	0.203	0.401	**0.261**	**0.199**	0.388
16	0.273	0.203	0.402	0.455	0.231	0.472	0.283	0.205	0.412	0.264	0.202	0.392

6.4.3 Baselines

We aim to learn a 20 layer DNN in the online setting, with 100 units in each hidden layer. As baselines, we learn the 20 layer network online using OGD (Online Backpropagation), OGD Momentum, OGD Nesterov, and Highway Networks. Since a 20 layer network would be very difficult to learn in the online setting, we also compare the performance of shallower networks — DNNs with fewer layers (2,3,4,8,16), trained using Online BP. We used ReLU Activation, and a fixed learning rate of 0.01 (chosen, as it gave the best final error rate for all DNN-based baselines). For momentum techniques, a fixed learning rate of 0.001 was used, and we finetuned the momentum parameter to obtain the best performance for the baselines. For HBP, we attached a classifier to each of the 19 hidden layers (and not directly to the input). This gave 19 classifiers each with depth from $2, \ldots, 20$. We set $\beta = 0.99$ and the smoothing parameter $s = 0.2$. Implementation was in Keras [17]. We also compared with representative state of the art linear (OGD, Adaptive Regularization of Weights (AROW), Soft-Confidence Weighted Learning (SCW) [51]) and kernel (Fourier Online Gradient Descent (FOGD) and Nystrom Online Gradient Descent (NOGD)[78]) online learning algorithms.

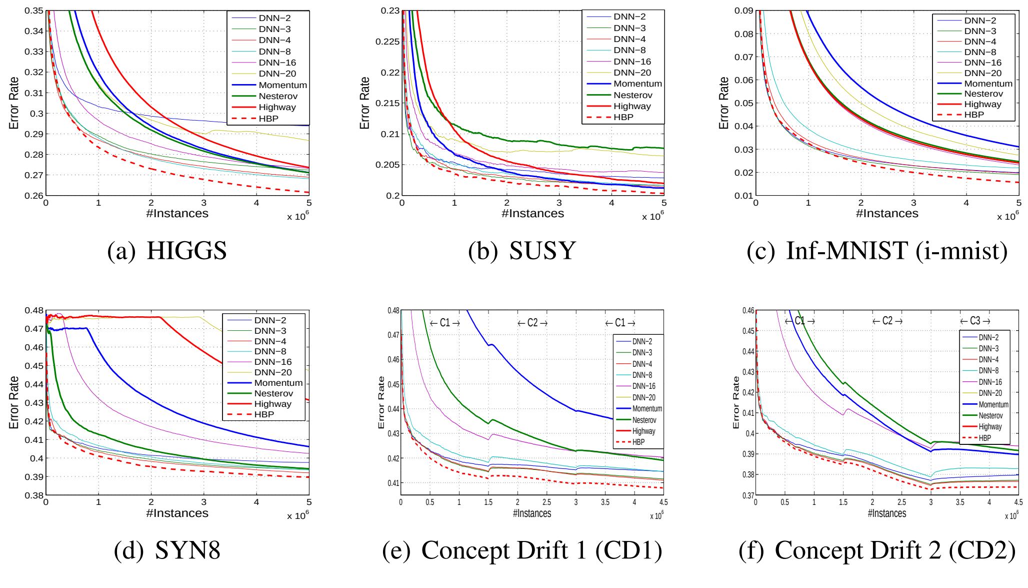
6.4.4 Evaluation of Online Deep Learning Algorithms

The final cumulative error obtained by all the baselines and the proposed HBP technique can be seen in Table 6.3. First, traditional online learning algorithms (linear and kernel) have relatively poor performance on complex datasets. Next, in learning with a 20-layer network, the convergence is slow, resulting in poor overall performance. While second order methods utilizing momentum and highway networks are able to offer some advantage over simple Online Gradient Descent, they can be easily beaten by a relatively shallower networks in the online setting. We observed before that relatively shallower networks could give a competitive performance in the online setting, but lacked the ability to exploit the power of depth at a later stage. In contrast, HBP enjoyed the best of both worlds, by allowing for faster convergence initially, and making use of the power of depth at a later stage. This way HBP was able to do automatic model selection online, enjoying merits of both shallow and deep networks, and this resulted in HBP outperforming all the DNNs of different depths, in terms of online performance. It should be noted that the optimal depth for DNN is not known before the learning process, and even then HBP outperforms all DNNs at any depth.

In Figure 6.2, we can see the convergence behavior of all the algorithms on the stationary as well as concept drift datasets. In the stationary datasets, HBP shows consistent outperformance over all the baselines. The only exception is in the very initial stages of the online learning phase, where shallower baselines are able to outperform HBP. This is not a surprising result, as HBP has many more parameters to learn. However, HBP is able to quickly outperform the shallow networks. The performance of HBP in concept drifting scenarios demonstrates its ability to adapt to the change fairly quickly, enabling usage of DNNs in the concept drifting scenarios. Looking at the performance of simple 20-layer (and 16-layer) networks on concept drifting data, we can see that utilizing deep representation for such scenarios can be difficult.

Table 6.3: Final Online Cumulative Error Rate obtained by the algorithms. L represents the number of Layers. Best performance is in bold.

Model	Method	L	Higgs	Susy	i-mnist	Syn8	CD1	CD2
Linear	**OGD**	1	0.3620	0.2160	0.1230	0.4070	0.4360	0.4270
	AROW	1	0.3630	0.2160	0.1250	0.4050	0.4340	0.4250
	SCW	1	0.3530	0.2140	0.1230	0.4050	0.4340	0.4250
Kernel	**FOGD**	2	0.2973	0.2021	0.0495	0.3962	0.4329	0.4193
	NOGD	2	0.3488	0.2045	0.1045	0.4146	0.4455	0.4356
DNNs	**OGD (Online BP)**	2	0.2938	0.2028	0.0199	0.3976	0.4146	0.3797
	OGD (Online BP)	3	0.2724	0.2016	0.0190	0.3936	0.4115	0.3772
	OGD (Online BP)	4	0.2688	0.2014	0.0196	0.3920	0.4110	0.3766
	OGD (Online BP)	8	0.2682	0.2016	0.0219	0.3936	0.4145	0.3829
	OGD (Online BP)	16	0.2731	0.2037	0.0232	0.4025	0.4204	0.3939
	OGD (Online BP)	20	0.2868	0.2064	0.0274	0.4472	0.4928	0.4925
	OGD+Momentum	20	0.2711	0.2012	0.0310	0.4062	0.4312	0.3897
	OGD+Nesterov	20	0.2711	0.2076	0.0247	0.3942	0.4191	0.3917
	Highway	20	0.2736	0.2019	0.0241	0.4313	0.4928	0.4925
	Hedge BP	20	**0.2615**	**0.2003**	**0.0156**	**0.3896**	**0.4079**	**0.3739**



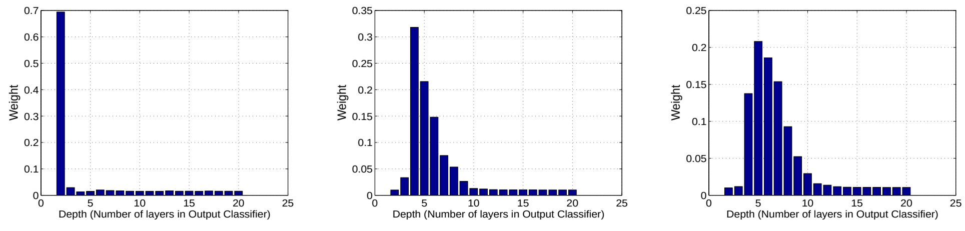
(a) HIGGS (b) SUSY (c) Inf-MNIST (i-mnist)

(d) SYN8 (e) Concept Drift 1 (CD1) (f) Concept Drift 2 (CD2)

Figure 6.2: Convergence behavior of DNNs in Online Setting on stationary and concept drifting data.

6.4.5 Weight Distribution for Depth Adaptation

In this section we look at the weight distribution learnt by HBP over time. We analyse the mean weight distribution in different segments of the Online Learning phase in Figure 6.3. In the initial phase (first 0.5%), the maximum weight has gone to the shallowest classifier (with just one hidden layer). In the second phase (10-15%), slightly deeper classifiers (classifiers with 4-5 layers) have picked up some weight, and in the third segment (60-80%), even deeper classifiers have gotten more weight (classifiers with 5-7 layers). The shallow and the very deep classifiers receive little weight in the last segment showing HBPs ability to perform model selection. Few classifiers having similar depth indicates that the intermediate features learnt are themselves of discriminatory nature, which are being used by the deeper classifiers to potentially learn better features.

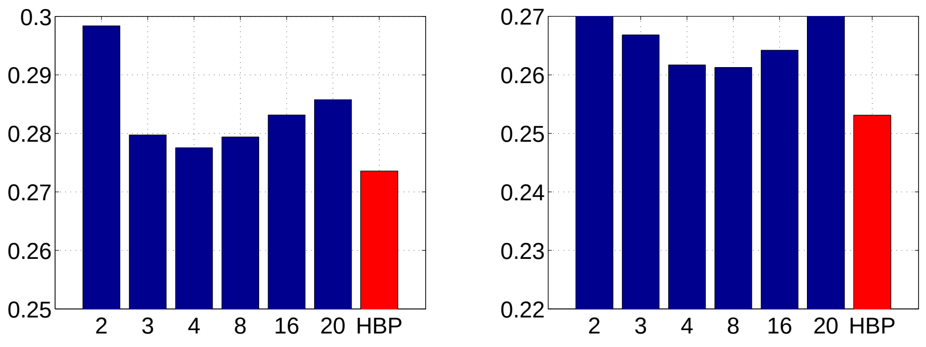


(a) First 0.5% of Data (b) 10-15% of Data (c) 60-80% of Data

Figure 6.3: Evolution of weight distribution of the classifiers over time using HBP on HIGGS dataset.

6.4.6 Performance with Depth Adaptation

Here we evaluate the performance of HBP in different segments of the data to see how the proposed HBP algorithm performed as compared to the DNNs of different depth in different segments of the data. In Figure 6.4, we can see, HBP matches (and even beats) the performance of the best depth network in both the beginning and at a later stage of the training phase. This shows its ability to exploit faster convergence of shallow networks in the beginning, and power of deep representation towards the end. Not only is it able to do automatic model selection, but also it is able to offer



(a) Error in 10-15% of data

(b) Error in 60-80% of data

Figure 6.4: Error Rate in different segments of the Data. Red represents HBP using a 20-layer network. Blue are OGD using DNN with layers = 2,3,4,8, 16 and 20.

a good initialization for the deeper representation, so that the depth of the network can be exploited sooner, thus beating a DNN of every depth.

6.4.7 Depth Robustness

Table 6.4: Robustness of HBP to depth of the base network compared to traditional DNN

Depth	**12**	**16**	**20**	**30**
Online BP	0.2692	0.2731	0.2868	0.4770
HBP	0.2609	0.2613	0.2615	0.2620

We evaluate the performance of HBP with varying DNN depth. We consider a 12, 16, 20, and a 30-layer HBP based DNN and compare their performance on Higgs against simple Online BP. See Table 6.4: in which we see that the performance variation with depth does not significantly alter HBPs performance, while for simple Online BP, significant increase in depth hurts the learning process.

6.5 Conclusion

We identified several issues which prevented existing DNNs from being used in an online setting, which meant that they could not be used for streaming data, and necessarily required storage of the entire data in memory. These issues primarily arose from difficulty in model selection (an appropriate depth DNN), and convergence difficulties from vanishing gradient and diminishing feature reuse. We used the ”shallow to deep” principle, and designed an algorithm for Online Deep Learning: Hedge Backpropagation, which enabled the usage of Deep Neural Networks in an online setting. The proposed scheme used a hedging strategy to make predictions with multiple outputs from different hidden layers of the network, and the backpropagation algorithm was modified to allow for knowledge sharing among the deeper and shallower networks. We validated the performance improvements offered by the proposed algorithms through experiments on large datasets.

Chapter 7

Conclusions and Future Work

7.1 Summary of Contributions

In this dissertation we proposed novel algorithms for Online Learning with Non-linear models. Specifically we identified limitations of existing works in Online Multiple Kernel Learning, and proposed novel algorithms to address those issues. We also identified limitations of existing works that prevented usage of DNNs in the online setting, and developed novel Online Deep Learning algorithms:

We made the following contributions in the field of Online Multiple Kernel Learning:

- **Online Multiple Kernel Regression**: We propose Online Multiple Kernel Regression Algorithms, which learn a kernel-based regressor in an online fashion, and dynamically explore a pool of diverse kernels to enhance the model performance. This builds on the existing OMKL work which is primarily designed for classification tasks [58, 49] or for structured prediction[81]. We extend the OMKL principles for regression tasks, and improve the kernel combination methods over the previous work. We also demonstrate the application to Auto-Regressive Time Series Modeling.
- **Temporal Kernel Descriptors for Learning with Time-Sensitive Patterns**:

Successful performance of kernel based methods depends on the choice of the kernel. We consider scenarios where patterns are sensitive to time (or time-stamps), and accordingly we propose Temporal Kernel Descriptors. These kernel descriptors are able to automatically learn the association of time-stamps at different resolutions (e.g. hourly, daily, etc.) with different data modalities. This enables more effective capturing of the information, and results in superior performance as compared to directly using traditional kernels.

- **Cost-Sensitive Online Multiple Kernel Classification**: Often data streams are can be highly imbalanced, which may make evaluation of algorithms on common metrics such as accuracy unreliable. Consequently, for imbalanced data streams, we consider evaluation over cost-sensitive metrics. We then propose Cost-Sensitive Online Multiple Kernel Classification and also demonstrate the application of the proposed methods to online anomaly detection. To achieve this we develop new cost-sensitive kernel learning approaches and cost-sensitive kernel combination approaches.

We also developed a novel approach for learning Deep Neural Networks on the fly - Online Deep Learning:

- **Online Deep Learning**: We highlighted the limitation of existing approaches which made it very difficult to use Deep Neural Networks online. We leveraged on the ”shallow to deep principle”, according to which shallow networks converge faster than deeper networks, and proposed a novel *Hedge Backpropagation* algorithm to learn Deep Neural Networks online. Hedge Backpropagation attached a classifier to each hidden layer, and in every online iteration evaluated the performance of the algorithms using Hedge Algorithm, and appropriately amended the overproportioned algorithm to enable faster learning.

7.2 Future Directions

In this dissertation, while we have addressed several challenges faced while online learning of nonlinear models using multiple kernels (in particular we tackled regression problems, challenges associated with temporal properties of the data, and streams with imbalanced data streams). However multiple kernel learning models have some restrictions with regard to learning nonlinear models. Most notably they have limited capacity models as compared to Deep Neural Networks (kernel methods are effectively 2-layer Deep Neural Networks). Consequently we developed online learning algorithms for deep neural networks (Online Deep Learning). While Online Deep Learning clearly has the capacity to learn more complex functions than multiple kernel learning, it still suffers from convergence challenges, and does not have rigorous theoretical guarantees to an optimum. There are several directions to improve Online Learning with nonlinear models, particularly in the domain of Online Deep Learning. Next we briefly discuss two main directions:

Advanced Online Deep Learning: There have been very limited efforts in the field of Online Deep learning. Ours is among the first such efforts. There is tremendous scope to further improve the algorithms, particularly from 2 angles: (i) Usage of second order information; and (ii) Improved computation speed. For the first direction, using second order information such as momentum to improve convergence can possibly improve convergence speed. However, for Hedge Backpropagation, applying momentum techniques may be non trivial, since the objective function is not fixed, and adapts at every online iteration. Another weakness of the current method is the possibly unnecessary computations performed for the deeper layers, which do not really impact the prediction performance of the network. A more ideal approach would be to grown the network layer by layer during the online learning phase, so that unnecessary computation of the deeper layers can be avoided.

Online Transfer Learning: Transfer learning deals with learning a model in one domain and transferring the knowledge from the learnt domain to a new domain where we usually do not have sufficient labeled examples to train a model from scratch [92]. Further, Deep Learning Models when trained on large datasets are known to produce hidden layer features, that have varying degrees of transferability to other applications [129]. Unfortunately, deciding which layer's features are the ideal features to use in a new application (or how much of the network to freeze during training in a new domain) is an unaddressed problem. We believe a more principled approach to select the best performing hidden layer features for transfer learning can significantly help improve both transfer learning and online transfer learning. In particular, a Hedge Backpropagation inspired approach can allow us to directly evaluate the performance of the different layers, enabling automatic selection of transferable features.

Bibliography

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[2] O. Anava, E. Hazan, S. Mannor, and O. Shamir. Online learning for time series prediction. In *COLT*, volume 30, pages 172–184, 2013.

[3] C. M. Antunes and A. L. Oliveira. Temporal data mining: An overview. In *KDD workshop on temporal data mining*, pages 1–13, 2001.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[5] F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the 21st ICML*, page 6. ACM, 2004.

[6] W. Barbakh and C. Fyfe. Online clustering algorithms. *International Journal of Neural Systems*, 18(03):185–194, 2008.

[7] A. Ben-Hur and W. S. Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl 1):i38–i46, 2005.

[8] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[9] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. *An MIT Press book in preparation. Draft chapters available at http://www. iro. umontreal. ca/ bengioy/dl-book*, 2015.

[10] K. P. Bennett, M. Momma, and M. J. Embrechts. Mark: A boosting algorithm for heterogeneous kernel models. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 24–31. ACM, 2002.

[11] O. Bousquet and D. J. Herrmann. On the complexity of learning the kernel matrix. *Advances in NIPS*, 2003.

[12] D. Brugger, W. Rosenstiel, and M. Bogdan. Online svr training by solving the primal optimization problem. *Journal of Signal Processing Systems*, 65(3):391–402, 2011.

[13] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 2007.

[14] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[15] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[16] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *ICLR*, 2016.

[17] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[18] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

[19] M. Christoudias, R. Urtasun, and T. Darrell. Bayesian localized multiple kernel learning. *Univ. California Berkeley, Berkeley, CA*, 2009.

[20] C. Cortes, M. Mohri, and A. Rostamizadeh. Two-stage learning kernel algorithms. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 239–246, 2010.

[21] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, 7:551–585, Dec. 2006.

[22] K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Advances in NIPS 16*. MIT Press, Cambridge, MA, 2004.

[23] K. Crammer and D. D. Lee. Learning via gaussian herding. In *Advances in neural information processing systems*, pages 451–459, 2010.

[24] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *JMLR*, 2003.

[25] M. Cuturi and A. Doucet. Autoregressive kernels for time series. *arXiv preprint arXiv:1101.0673*, 2011.

[26] A. Daniely, A. Gonen, and S. Shalev-Shwartz. Strongly adaptive online learning. *arXiv preprint arXiv:1502.07073*, 2015.

[27] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[28] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[29] I. M. de Diego, A. Muñoz, and J. M. Moguerza. Methods for the combination of kernel matrices within a support vector framework. *Machine learning*, 78(1-2):137, 2010.

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

[31] Y. Ding, P. Zhao, S. C. Hoi, and Y.-S. Ong. An adaptive gradient method for online auc maximization. *AAAI*, 2015.

[32] S. Donoho. Early detection of insider trading in option markets. *ACM SIGKDD*, 2004.

[33] M. Dredze, K. Crammer, and F. Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.

[34] H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2-3):113–127, 2014.

[35] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[36] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, volume 904, pages 23–37. Springer Berlin Heidelberg, 1995.

[37] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[38] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

[39] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[40] C. Gentile. A new approximate maximal margin classification algorithm. *JMLR*, 2002.

[41] S. Gezici, H. Kobayashi, and H. V. Poor. A new approach to mobile position tracking. In *Proc. IEEE Sarnoff Symp. Advances in Wired and Wireless Communications*, 2003.

[42] M. Girolami and M. Zhong. Data integration for classification problems employing gaussian process priors. *Advances in Neural Information Processing Systems*, 19:465, 2007.

[43] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *JMLR*, 2011.

[44] S. Guha and A. McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–279. ACM, 2006.

[45] C. Gulcehre, M. Moczulski, F. Visin, and Y. Bengio. Mollifying networks. *arXiv preprint arXiv:1608.04980*, 2016.

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[47] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] S. C. Hoi, R. Jin, and M. R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *ICML*, pages 361–368. ACM, 2007.

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

[50] S. C. Hoi, M. R. Lyu, and E. Y. Chang. Learning the unified kernel machines for classification. In *KDD*, pages 187–196. ACM, 2006.

[51] S. C. Hoi, J. Wang, and P. Zhao. Libol: A library for online learning algorithms. *JMLR*, 15:495–499, 2014.

[52] S. C. Hoi and P. Zhao. Online learning methods for big data analytics. *Tutorial of IEEE ICDM*, 2014.

[53] D. Hovy, J. Fan, A. Gliozzo, S. Patwardhan, and C. Welty. When did that happen?: linking events and relations to timestamps. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 185–193. Association for Computational Linguistics, 2012.

[54] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[55] Z. Huang, H. Chen, C. Hsu, W. Chen, and S. Wu. Credit rating analysis with support vector machines and neural networks: a market comparative study. *DSS*, 2004.

[56] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[57] T. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. *ACM Transactions on Database Systems (TODS)*, 33(4):26, 2008.

[58] R. Jin, S. C. Hoi, and T. Yang. Online multiple kernel learning: Algorithms and mistake bounds. In *Algorithmic Learning Theory*. Springer Berlin Heidelberg, 2010.

[59] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, volume 3, pages 321–328, 2003.

[60] V. Kekatos, Y. Zhang, and G. Giannakis. Electricity market forecasting via low-rank multi-kernel learning. *Selected Topics in Signal Processing, IEEE Journal of*, 8(6):1182–1193, 2014.

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

[62] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[64] J. T. Kwok and I. W. Tsang. Learning with idealized kernels. In *ICML*, pages 400–407, 2003.

[65] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.*, 5:27–72, Dec. 2004.

[66] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[67] S. Laxman and P. S. Sastry. A survey of temporal data mining. *Sadhana*, 2006.

[68] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[69] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, volume 2, page 6, 2015.

[70] S.-W. Lee, C.-Y. Lee, D. H. Kwak, J. Kim, J. Kim, and B.-T. Zhang. Dual-memory deep learning architectures for lifelong learning of everyday human behaviors. In *IJCAI*, 2016.

[71] B. Li and S. C. Hoi. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 46(3):35, 2014.

[72] B. Li, D. Sahoo, and S. Hoi. Olps: A toolbox for online portfolio selection. *Journal of Machine Learning Research (JMLR)*, 2015.

[73] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *ML*, 2002.

[74] Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. The perceptron algorithm with uneven margins. In *ICML*, volume 2, pages 379–386, 2002.

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

[76] C. Liu, S. C. Hoi, P. Zhao, and J. Sun. Online arima algorithms for time series prediction. 2016.

[77] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007.

[78] J. Lu, S. C. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu. Large scale online kernel learning. *The Journal of Machine Learning Research*, 17(1):1613–1655, 2016.

[79] J. Luo, F. Orabona, M. Fornoni, B. Caputo, and N. Cesa-Bianchi. Om-2: An online multi-class multi-kernel learning algorithm. Technical report, Idiap, 2010.

[80] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 681–688. ACM, 2009.

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

[82] A. McGregor. Finding graph matchings in data streams. In *APPROX-RANDOM*, volume 3624, pages 170–181. Springer, 2005.

[83] M. Michalak. Time series prediction with periodic kernels. *Advances in Intelligent and Soft Computing*, 95:137–146, 2011.

[84] M. Michalak. On selection of periodic kernels parameters in time series prediction. 2014.

[85] S. A. Mirroshandel, M. Khayyamian, and G. Ghassem-Sani. Syntactic tree kernels for event-time temporal relation learning. In *Human Language Technology. Challenges for Computer Science and Linguistics*, pages 213–223. Springer, 2011.

[86] T. Mitsa. *Temporal data mining*. CRC Press, 2010.

[87] J. M. Moguerza, A. Munoz, and I. M. de Diego. Improving support vector classification via the combination of multiple sources of information. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 592–600. Springer, 2004.

[88] E. Moroshko and K. Crammer. A last-step regression algorithm for non-stationary online learning. *CoRR*, abs/1303.3754, 2013.

[89] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[90] F. Orabona and K. Crammer. New adaptive algorithms for online classification. In *Advances in neural information processing systems*, pages 1840–1848, 2010.

[91] F. Orabona, J. Keshet, and B. Caputo. The projectron: a bounded kernel-based perceptron. In *ICML*, pages 720–727. ACM, 2008.

[92] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[93] P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the fifth annual international conference on Computational biology*, pages 249–255. ACM, 2001.

[94] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[95] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *24th ICML*, pages 775–782. ACM, 2007.

[96] A. Rakotomamonjy, F. Bach, S. Canu, Y. Grandvalet, et al. Simplemkl. *JMLR*, 9:2491–2521, 2008.

[97] C. Richard, J. C. M. Bermudez, and P. Honeine. Online prediction of time series data with kernels. *Signal Processing, IEEE Transactions on*, 57(3):1058–1067, 2009.

[98] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

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

[100] D. Sahoo, S. C. Hoi, and B. Li. Online multiple kernel regression. In *ACM SIGKDD*, 2014.

[101] D. Sahoo, C. Liu, and S. C. Hoi. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.

[102] N. Sapankevych and R. Sankar. Time series prediction using support vector machines: a survey. *Computational Intelligence Magazine, IEEE*, 4(2):24–38, 2009.

[103] B. Schölkopf and A. J. Smola. *Learning with kernels*. The MIT Press, 2002.

[104] L. C. S. V. D. Schuurmans and S. W. Caelli. Implicit online learning with kernels. In *NIPS*, volume 19, page 249. MIT Press, 2007.

[105] S. Shalev-Shwartz. Online learning: Theory, algorithms, and applications. 2007.

[106] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[107] K. C. J. K. Y. Singer. Kernel design using boosting. *Advances in neural information processing systems*, 2002.

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

[109] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[110] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *JMLR*, 7:1531–1565, 2006.

[111] S. Srinivas and R. V. Babu. Learning neural network architectures using backpropagation. *arXiv preprint arXiv:1511.05497*, 2015.

[112] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[113] F. E. Tay and L. Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[114] F. E. Tay and L. Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1):847–861, 2002.

[115] U. Thissen, R. Van Brakel, A. De Weijer, W. Melssen, and L. Buydens. Using support vector machines for time series prediction. *Chemometrics and intelligent laboratory systems*, 69(1):35–49, 2003.

[116] V. Vovk. A game of prediction with expert advice. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 51–60. ACM, 1995.

[117] J. Wang, P. Zhao, and S. C. Hoi. Cost-sensitive online classification. *Knowledge and Data Engineering, IEEE Transactions on*, 26(10):2425–2438, 2014.

[118] J. Wang, P. Zhao, and S. C. Hoi. Soft confidence-weighted learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):15, 2016.

[119] T. Wei, C. Wang, R. Rui, and C. W. Chen. Network morphism. *ICML*, 2016.

[120] B. WIDROW, M. E. HOFF, et al. Adaptive switching circuits. 1960.

[121] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

[122] H. Xia and S. C. Hoi. Mkboost: A framework of multiple kernel boosting. *IEEE Transactions on knowledge and data engineering*, 25(7):1574–1586, 2013.

[123] T. Xiang and S. Gong. Video behavior profiling for anomaly detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(5):893–908, 2008.

[124] Z. Xu, R. Jin, I. King, and M. R. Lyu. An extended level method for efficient multiple kernel learning. In *NIPS*, pages 1825–1832, 2008.

[125] H. Yang, Z. Xu, I. King, and M. R. Lyu. Online learning for group lasso. In *Proceedings of the 27th ICML*, pages 1191–1198, 2010.

[126] J. Yang and Y. Zhang. Application research of support vector machines in condition trend prediction of mechanical equipment. In *Advances in Neural Networks–ISNN 2005*, pages 857–864. Springer, 2005.

[127] L. Yang, R. Jin, and J. Ye. Online learning by ellipsoid method. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1153–1160. ACM, 2009.

[128] Y. Yao, A. Sharma, L. Golubchik, and R. Govindan. Online anomaly detection for sensor systems: A simple and efficient approach. *Performance Evaluation*, 67(11):1059–1075, 2010.

[129] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[130] P. Zhao, R. Jin, T. Yang, and S. C. Hoi. Online auc maximization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011.

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

[132] G. Zhou, K. Sohn, and H. Lee. Online incremental feature learning with denoising autoencoders. *Ann Arbor*, 1001:48109, 2012.

[133] J. Zhuang, I. W. Tsang, and S. C. Hoi. A family of simple non-parametric kernel learning algorithms. *JMLR*, 12:1313–1347, 2011.

[134] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. 2003.

[135] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.