2017 IEEE International Conference on Data Mining

Large Scale Kernel Methods for Online AUC Maximization

Yi Ding*, Chenghao Liu†, Peilin Zhao‡ and Steven C.H. Hoi†
*Department of Computer Science, The University of Chicago, USA
†School of Information Systems, Singapore Management University, Singapore
‡School of Software Engineering, South China University of Technology, Guangzhou, China
Email: *dingy@uchicago.edu, †chliu@smu.edu.sg, ‡peilinzhao@hotmail.com, †chhoi@smu.edu.sg,

***Abstract*—Learning to optimize AUC performance for classifying label imbalanced data in online scenarios has been extensively studied in recent years. Most of the existing work has attempted to address the problem directly in the original feature space, which may not be suitable for non-linearly separable datasets. To solve this issue, some kernel-based learning methods are proposed for non-linearly separable datasets. However, such kernel approaches have been shown to be inefficient and failed to scale well on large datasets in practice. Taking this cue, in this work, we explore the use of scalable kernel-based learning techniques as surrogates to existing approaches: random Fourier features and Nyström method, for tackling the problem and bringing new insights to the differences between the two methods based on their online performance. In contrast to the conventional kernel-based learning methods which suffer from high computational complexity of the kernel matrix, our proposed approaches elevate this issue with linear features that approximate the kernel function/matrix. Specifically, two different surrogate kernel-based learning models are presented for addressing the online AUC maximization task: (i) the Fourier Online AUC Maximization (FOAM) algorithm that samples the basis functions from a data-independent distribution to approximate the kernel function; and (ii) the Nyström Online AUC Maximization (NOAM) algorithm that samples a subset of instances from the training data to approximate the kernel matrix by a low rank matrix. Another novelty of the present work is the proposed mini-batch Online Gradient Descent method for model updating to control the noise and reduce the variance of gradients. We provide theoretical analyses for the two proposed algorithms. Empirical studies on commonly used large scale datasets show that the proposed algorithms outperformed existing state-of-the-art methods in terms of both AUC performance and computational efficiency.**

I. Introduction

AUC (Area Under ROC curve) [1] has played a significant role in evaluating machine learning performance, including the spam detection dataset wherein the spam emails occupy only a tiny fraction of all emails, and the medical diagnosis dataset where cancer cases are treated with higher bias in the learning process than the benign ones. In recent years, AUC, which measures the probability for a randomly drawn positive instance to have a higher decision value than a randomly sample negative instance, has become one of the most widely-used performance measurements for handling the label imbalance phenomenon. Given its important role in theory and application, current research on AUC has escalated to a point where it has become the direct learning objective of interest in both batch and online scenarios [2]–[7].

To achieve high efficiency and scalability in real-world applications, online AUC maximization (OAM) for streaming data has attracted increasing research interests over the years. The key challenge of OAM is that the objective function involved is represented by the sum of pairwise losses between instances of different classes, making conventional online learning algorithms unsuitable to optimize it directly. To address this issue, two popular types of online AUC maximization frameworks have been proposed. The first type is the buffer-based sampling method [4], [7], [8], which maintains some historical instances in the buffer to represent the observed data with opposite class. The other type is the one-pass AUC optimization method [5] using squared loss as the loss function and a new adaptive gradient method for OAM to exploit the second order information [6], which has achieved improved online performance at comparable time costs over the first order OAM methods.

Non-separability of data is a common characteristic found in many real-world applications. Therefore, in order to investigate the nonlinearity of the data and conduct effective model updates, it is essential to introduce kernel tricks and learn kernel-based models that are capable of handling non-separable datasets. A kernelized online imbalanced learning method was proposed recently in [7]. It considered conventional online kernel learning technique with a bounded support vector size. However, this approach requires high computational and storage costs for the kernel matrix and thus unsuitable for large scale learning tasks in practice.

In contrast to the work in [7], here we present a novel framework with the introduction of surrogate kernel-based learning models to address the non-separability and scalability problems involving OAM tasks. In particular, we use functional approximation strategy to approximate the kernel function by mapping data in the original space to the new feature space which is often of high dimensions, and then conducting OAM in the new feature space, which is referred as the Kernel-based Online AUC Maximization (KOAM). Specifically, we propose two new algorithms for KOAM: (i) KOAM with Random Fourier Features (FOAM) algorithm that uses the random Fourier features to approximate the shift-invariant kernels; and (ii) KOAM with Nyström method that applies the Nyström method to approximate the kernel matrix. Both theoretical and empirical results are provided to examine

2374-8486/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDM.2017.18

the efficacy of both algorithms. In addition, we adopts a mini-batch Online Gradient Descent method for model updating to control the noise and reduce the variance efficiently.

The rest of this paper is organized as follows. We first review some related work. Secondly, we present the two proposed algorithms that serve as suitable surrogate kernel-based learning models for online AUC maximization involving large scale datasets. Then, we offer their theoretical analyses and empirical studies, respectively. Finally, we conclude the paper with a brief summary of the present work.

II. Related Work

The present work is closely related to three core topics of data mining and machine learning, namely, online learning, AUC maximization, and kernel-based online learning. In what follows, we briefly review some of the important related work in each of the topics.

Online Learning. Online learning represents a family of efficient and scalable machine learning algorithms that has enjoyed solid theoretical guarantees and reported notable empirical performances in many real-world applications [9]–[11]. The first and most well-known online algorithm is the Perceptron [12]. After introducing the principle of "maximum margin" for classification, the Passive-Aggressive (PA) algorithm [13] was proposed followed by several second order online algorithms [14], [15], which attempt to use parameter confidence information to improve online performance. In recent years, many studies have tried to explore the connections between online learning and stochastic optimization [16], [17]. This has led to an enhancement on the theoretical foundation of online learning and attracted new research interests from other communities such as theoretical computer science and statistics.

AUC Maximization. Due to its significance for measuring the label imbalanced classification task, AUC has been deemed recently as the objective function to optimize [2]–[5]. In the initial efforts, the general framework for optimizing multivariate performance measurements including AUC [3] was presented in the setup of batch learning. For online setting, two categories of online AUC maximization framework have been proposed very recently. The first framework is based on the idea of buffer sampling [4], [7], [8] using a fixed-size buffer to store the observed data with opposite class label for loss calculation. A typical buffer update policy for this framework is available in [4], which leverages the reservoir sampling technique to solve oblivious problem for streaming data. Instead of using pairwise hinge loss [4] as the surrogate loss function, [5] and [6] considered the square loss as the loss function and introduced the one-pass AUC optimization mode, which maintained the mean vector and covariance matrices for model updating purposes.

Kernel-based Online Learning. Compared to conventional kernel-based batch learning [18], kernel-based online learning [19] is more computationally favorable in that it only needs to go through the training instances once in the learning process. The conventional kernel-based online learning algorithms maintain a set of support vectors to form the kernel-based prediction model. Using the principle of "budget online kernel classification" proposed in [20] and subsequently extended by several others [21], the kernel matrix is computed during the online phase. To address the high storage requirement of the support vectors and high kernel matrix computational cost of conventional kernel-based online learning approaches, one way is to solve the kernel learning problem by approximating the kernel function with a suitable linear data representation in the new feature space. The most popular methods for such approximation tasks are the random Fourier features [22] and the Nyström method [23]. For instance, both of these methods have been successfully used to speed up kernel-based batch learning without any deterioration reported in the accuracy performance [24].

Despite the extensive work in these three different fields, to the best of our knowledge, the present work represents a first attempt to explore the use of surrogate kernel-based learning models for improving online AUC maximization performance involving large scale datasets.

III. Algorithms

A. Problem Setting

Consider a kernel-based online binary classification task over an incoming sequence of $(\mathbf{x}_t, y_t)$, $t = 1, 2, \ldots, T$, where $\mathbf{x}_t \in \mathbb{R}^d$ is the observed instance in the original feature space received at the t-th trial and $y_t \in \{-1, +1\}$ is the true class label of $\mathbf{x}_t$ that is only revealed by the environment when each online prediction round ends. Without loss of generality, we assume positive class is the minority class while the negative class is majority. In the kernel-based online AUC maximization task, our goal is to learn a kernel-based prediction model $f(\mathbf{x})$ to correctly classify an incoming instance $\mathbf{x}_t \in \mathbb{R}^d$ given by: $f(\mathbf{x}) = \sum_{i=1}^{S} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$, where S is the number of support vectors, α_i is the coefficient of the i-th support vector, and $\kappa(\mathbf{x}_i, \mathbf{x})$ is the kernel function between two data points. Noted that this formulation is the general framework of existing kernel-based online learning on a budget, which tries to bound the number of support vectors in order to find a trade-off between the computational efficiency and classification performance. However, here we introduce a functional approximation scheme to avoid such sophisticated budget maintenance efforts. Instead of relying on inner product between data points, our idea is to explicitly construct a new feature representation $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^K$ such that the inner product between new data vectors is capable of approximating the kernel function: $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j)$. In this way, we can simply project $\mathbf{x}$ in the original feature space to $\mathbf{z}$ in another new feature space before applying linear learning methods to approximate nonlinear kernel-based models. Thus, the above formulation can be reformulated as $f(\mathbf{x}) = \sum_{i=1}^{S} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \sum_{i=1}^{S} \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}(\mathbf{x})$, where $\mathbf{w} = \sum_{i=1}^{S} \alpha_i \mathbf{z}(\mathbf{x}_i)$ is the weight vector learned in the

new feature space. So far, we have transformed the problem of kernel-based online AUC maximization to a regular online AUC maximization task in the new feature space. Based on this result, we introduce the general framework of online AUC maximization in the next paragraphs.

To begin with, we define the AUC performance measurement [1] explicitly for binary classification task on the new feature space from the input. Given a dataset $\mathcal{D} = \{(\mathbf{z}_i, y_i) \in \mathbb{R}^K \times \{-1, +1\} | \ i \in [n]\}$, where $[n] = \{1, 2, \ldots, n\}$, it is natural to divide it into two groups: the set of positive instances $\mathcal{D}_+ = \{(\mathbf{z}_i^+, +1) | \ i \in [n_+]\}$ and the set of negative instances $\mathcal{D}_- = \{(\mathbf{z}_j^-, -1) | \ j \in [n_-]\}$, where n_+ and n_- are the numbers of positive and negative instances, respectively. For a linear classifier $\mathbf{w} \in \mathbb{R}^K$ and dataset $\mathcal{D}$, its AUC performance measurement is written as:

$$\text{AUC}(\mathbf{w}) = \frac{\sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \mathbb{I}_{(\mathbf{w}^\top \mathbf{z}_i^+ > \mathbf{w}^\top \mathbf{z}_j^-)} + \frac{1}{2} \mathbb{I}_{(\mathbf{w}^\top \mathbf{z}_i^+ = \mathbf{w}^\top \mathbf{z}_j^-)}}{n_+ n_-},$$

where $\mathbb{I}_\pi$ is the indicator function that outputs a $'1'$ if the prediction π holds and $'0'$ the other way round. Since we apply the buffer-based approach for online AUC maximization, the indicator function $\mathbb{I}_\pi$ is replaced with the convex surrogate, i.e., the pairwise hinge loss function $\ell(\mathbf{w}^\top(\mathbf{z}_i^+ - \mathbf{z}_j^-)) = \max\{0, 1 - \mathbf{w}^\top(\mathbf{z}_i^+ - \mathbf{z}_j^-)\}$, and then the optimal classifier is the one minimizing the following objective function

$$\mathcal{P}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{\eta \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \ell(\mathbf{w}^\top(\mathbf{z}_i^+ - \mathbf{z}_j^-))}{2 n_+ n_-}, \tag{1}$$

where $\frac{1}{2}\|\mathbf{w}\|_2^2$ is a regularization term to control the complexity of the model and η is the positive penalty parameter to trade off the cumulative loss and regularization term.

The main challenge of optimizing the loss function (1) lies in that it belongs to the pairwise loss function involving the current instance and all the observed training instances with the opposite label, making it undesirable to store all the observed instances for each class in online scenarios. In order to address this issue, two fixed-size buffers [4], [8] are introduced for each class in loss calculation, B_+ with size N_+ and B_- with size N_-, respectively. With this, we show the general framework of kernel-based online AUC maximization in Algorithm 1.

In Algorithm 1, there are two core components, i.e., **UpdateBuffer** and **UpdateClassifier**. When the instance $(\mathbf{x}_t, y_t)$ arrives at trial t, the two buffers are updated and then the weight model $\mathbf{w}_t$ is updated by comparing $\mathbf{z}_t$ to the instances in B_+^t if $y_t = -1$ and to the instances in B_-^t if $y_t = +1$.

For the **UpdateBuffer** part, we apply the buffer update policy known as **Reservoir Sampling** [25] which is similar to [4]. The reservoir sampling technique can guarantee that the buffer maintains a uniform sampling from the preceding data stream at any time. Specifically, when an instance $(\mathbf{z}_t, y_t)$ arrives, we add it to the buffer $B_{y_t}^t$ if $|B_{y_t}^t| < N_{y_t}$. Otherwise, with probability $\frac{N_{y_t}}{N_{y_t}^{t+1}}$, we update the buffer $B_{y_t}^t$ by randomly replacing one instance in $B_{y_t}^t$ with $\mathbf{z}_t$. Algorithm 2 gives the details of the reservoir sampling technique.

Finally, we arrive at the routine of **UpdateClassifier**. The method used in [4] is the sequential Online Gradient Descent [26], i.e., $\mathbf{w}_t$ as $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{P}_t(\mathbf{w})$, where η is the learning rate parameter and $\nabla \mathcal{P}_t(\mathbf{w})$ is the gradient term of the loss function. Although it is an efficient and commonly used scheme for online model updates, its key limitation lies in that it uses noisy gradient estimated from a random instance of the dataset, which probably leads to slow convergence and poor performance especially when the variance of the noisy gradient is large [27]. To overcome this limitation, we adopt the mini-batch Online Gradient Descent (**mini-batch OGD**) updating strategy. Specifically, $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{L}_t(\mathbf{w}_t)$, where $\mathcal{L}_t(\mathbf{w}_t) = \frac{1}{|B|} \sum_{\mathbf{z}_i \in B} \ell(y_t \mathbf{w}_t^\top (\mathbf{z}_t - \mathbf{z}_i))$. So, the update is performed on an average of the gradients with reference to all the instances in the buffer at a time, rather than only single instance. The idea of mini-batch OGD is motivated by the variance reduction principle such that using multiple instances per iteration for computing the gradient could help reduce the variance of model updating per iteration. The procedure described above is detailed in Algorithm 3.

Algorithm 1 The Framework for Kernel-based Online AUC Maximization (KOAM)

Input: the penalty parameter η, the maximum buffer size N_+ and N_-, the Gaussian kernel function $\kappa(\cdot, \cdot)$ with parameter σ.
Initialize $\mathbf{w}_1 = \mathbf{0}$, $B_+^1 = B_-^1 = \emptyset$, $N_+^1 = N_-^1 = 0$.
for $t = 1, 2, \ldots, T$ **do**
  Receive a training instance $(\mathbf{x}_t, y_t)$.
  `KernelRepresentation`: Project input $\mathbf{x}_t$ to new feature representation $\mathbf{z}_t$ by kernel-based learning models.
  if $y_t = +1$ **then**
    $N_+^{t+1} = N_+^t + 1$, $N_-^{t+1} = N_-^t$, $B_-^{t+1} = B_-^t$,
    $B_+^{t+1} = \textbf{UpdateBuffer}(B_+^t, \mathbf{z}_t, N_+, N_+^{t+1})$,
    $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_-^{t+1})$.
  else
    $N_-^{t+1} = N_-^t + 1$, $N_+^{t+1} = N_+^t$, $B_+^{t+1} = B_+^t$,
    $B_-^{t+1} = \textbf{UpdateBuffer}(B_-^t, \mathbf{z}_t, N_-, N_-^{t+1})$,
    $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_+^{t+1})$.
  end if
end for

Algorithm 2 UpdateBuffer by Reservoir Sampling

Input: B^t, $\mathbf{z}_t$, N, N^{t+1}.
Output: updated buffer B^{t+1}.
if $|B^t| < N$ **then**
  $B^{t+1} = B^t \cup \{\mathbf{z}_t\}$.
else
  Sample Z from a Bernoulli distribution with $\Pr(Z = 1) = N/N^{t+1}$.
  if $Z = 1$ **then**
    Randomly delete an instance from B^t.
    $B^{t+1} = B^t \cup \{\mathbf{z}_t\}$.
  end if
end if
Return B^{t+1}.

B. Random Fourier Features for KOAM

In this subsection, we present the kernel-based online AUC maximization algorithm using random Fourier features with the general framework described above. Random Fourier projection maps the original input data into a low dimensional space spanned by the vectors drawn from the Fourier transform of a shift-invariant kernel function. The key to the random Fourier strategy lies in the Bochners theorem which links the positive definite kernels to their Fourier transform [28]. For a positive definite shift-invariant kernel defined as $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \kappa(\mathbf{x}_1 - \mathbf{x}_2)$ where κ is the kernel function, the Bochners theorem ensures that each kernel is the inverse Fourier transform resulting from a proper probability distribution. Let $p(\mathbf{u})$ be the proper probability distribution denoting the Fourier transform of $\kappa(\mathbf{x}_1 - \mathbf{x}_2)$,

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \kappa(\mathbf{x}_1 - \mathbf{x}_2) = \int p(\mathbf{u}) e^{i\mathbf{u}^\top(\mathbf{x}_1 - \mathbf{x}_2)} d\mathbf{u}. \tag{2}$$

When we expand this formulation by expressing it as an expectation of function with variable $\mathbf{u}$, we have

$$\begin{aligned} &\kappa(\mathbf{x}_1, \mathbf{x}_2) = E_{\mathbf{u}}[e^{i\mathbf{u}^\top \mathbf{x}_1} \cdot e^{-i\mathbf{u}^\top \mathbf{x}_2}] \\ &= E_{\mathbf{u}}[\cos(\mathbf{u}^\top \mathbf{x}_1)\cos(\mathbf{u}^\top \mathbf{x}_2) + \sin(\mathbf{u}^\top \mathbf{x}_1)\sin(\mathbf{u}^\top \mathbf{x}_2)] \\ &= E_{\mathbf{u}}[[\cos(\mathbf{u}^\top \mathbf{x}_1), \sin(\mathbf{u}^\top \mathbf{x}_1)] \cdot [\cos(\mathbf{u}^\top \mathbf{x}_2), \sin(\mathbf{u}^\top \mathbf{x}_2)]]. \end{aligned} \tag{3}$$

From this derivation, (2) can be obtained by only keeping the real part of the complex function. From (3), it is drawn that the shift-invariant kernel function can be expressed by the expectation of inner product of the new feature representation for the original feature vector, where the new feature representation is $\mathbf{z}(\mathbf{x}) = [\cos(\mathbf{u}^\top \mathbf{x}), \sin(\mathbf{u}^\top \mathbf{x})]^\top$. Therefore, we can approximate the expectation in (3) by sampling multiple random Fourier elements $\mathbf{u}_1, \cdots, \mathbf{u}_m$ independently from the distribution $p(\mathbf{u})$ to obtain a new representation of input $\mathbf{x}$: $\mathbf{z}_t(\mathbf{x}) = (\cos(\mathbf{u}_1^\top \mathbf{x}), \sin(\mathbf{u}_1^\top \mathbf{x}), \cdots, \cos(\mathbf{u}_m^\top \mathbf{x}), \sin(\mathbf{u}_m^\top \mathbf{x}))^\top$, where m is the number of sampled Fourier components. With this projection, the OAM in the original space can be solved by the KOAM in the new feature space. In this paper, we apply the shift-invariant Gaussian kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2/2\sigma^2)$ [22], and thus obtain the corresponding random Fourier component $\mathbf{u}$ with the distribution $p(\mathbf{u}) = \mathcal{N}(0, \sigma^{-2}I)$. In online setting, we construct the new feature representation of an incoming instance in the `KernelRepresentation` step of the KOAM shown in Algorithm 1. Note that although the random Fourier feature technique has been applied to several machine learning research tasks such as conventional classification [22], [29]–[31], kernel-based clustering [32], and data projection [33], the current work represents the first attempt to adopt it in kernel-based online AUC maximization task. We refer to the proposed kernel-based OAM algorithm using random Fourier features as the Fourier Online AUC Maximization (FOAM) with the details given in Algorithm 4.

C. Nyström Method for KOAM

In contrast to the FOAM algorithm that approximates the kernel function directly before applying the general framework of KOAM, the following Nyström method for online AUC maximization (NOAM) algorithm represents an attempt to approximate the kernel matrix indirectly during the online updating phases. The key motivation of exploring the Nyström method [23] for KOAM task lies in that the random Fourier components for the new feature vector representation have been randomly sampled from a data-independent distribution, hence failing to take full advantage of the important characteristics available in the original data. To address this, we introduce a data-dependent sampling technique, namely the Nyström method, to approximate the kernel function indirectly by constructing a low rank matrix from sampling a subset of training instances. Before we present our algorithm based on the Nyström method for kernel-based online AUC maximization task, we briefly review the Nyström method in what follows. To begin with, we provide some notations. Suppose the kernel matrix is $\mathbf{K} \in \mathbb{R}^{T \times T}$ with rank r. The Singular Value Decomposition (SVD) of $\mathbf{K}$ is $\mathbf{K} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$, where the columns of $\mathbf{V}$ are orthogonal and $\mathbf{D} = \text{diag}(\sigma_1, \cdots, \sigma_r)$ is the diagonal matrix containing the singular values of $\mathbf{K}$ in non-increasing order. For $k < r$, $\mathbf{K}_k = \sum_{i=1}^{k} \sigma_i V_i V_i^\top = \mathbf{V}_k \mathbf{D}_k \mathbf{V}_k^\top$ is the best rank-k approximation of $\mathbf{K}$, where $\mathbf{V}_i$ is the i-th column of matrix $\mathbf{V}$.

The Nyström method used here is to approximate a kernel matrix $\mathbf{K}$ by randomly sampling Q columns from $\mathbf{K}$, where $Q \ll T$. Then, a much smaller kernel matrix $\mathbf{W} \in \mathbb{R}^{Q \times Q}$ is

Algorithm 3 UpdateClassifier by mini-batch OGD

Input: $\mathbf{w}_t$, $(\mathbf{z}_t, y_t)$, B, η.
Output: updated classifier $\mathbf{w}_{t+1}$.
Define $A = \{\mathbf{z}_i | \ell(y_t \mathbf{w}_t^\top (\mathbf{z}_t - \mathbf{z}_i)) > 0, \mathbf{z}_i \in B\}$.
Let $\mathcal{L}_t(\mathbf{w}_t) = \frac{1}{|B|} \sum_{\mathbf{z}_i \in A} \ell(y_t \mathbf{w}_t^\top (\mathbf{z}_t - \mathbf{z}_i))$.
Calculate gradient $\nabla \mathcal{L}_t(\mathbf{w}_t) = -\frac{1}{|B|} \sum_{\mathbf{z}_i \in A} y_t (\mathbf{z}_t - \mathbf{z}_i)/2$.

$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{L}_t(\mathbf{w}_t)$.
Return $\mathbf{w}_{t+1}$.

Algorithm 4 Kernel-based OAM Algorithm with Random Fourier Features (FOAM)

Input: the penalty parameter η, the number of sampled random Fourier components m, the maximum buffer size N_+ and N_-, the Gaussian kernel function $\kappa(\cdot, \cdot)$ with parameter σ.
Initialize $\mathbf{w}_1 = \mathbf{0}, B_+^1 = B_-^1 = \emptyset$, $N_+^1 = N_-^1 = 0$.
Draw m i.i.d random Fourier components: $\mathbf{u}_1, \cdots, \mathbf{u}_m$ sampled from distribution $p(\mathbf{u}) = \mathcal{N}(0, \sigma^{-2}I)$.
for $t = 1, 2, \ldots, T$ **do**
  Receive a training instance $(\mathbf{x}_t, y_t)$.
  $\mathbf{z}_t(\mathbf{x}_t) = (\cos(\mathbf{u}_1^\top \mathbf{x}), \sin(\mathbf{u}_1^\top \mathbf{x}), \cdots, \cos(\mathbf{u}_m^\top \mathbf{x}), \sin(\mathbf{u}_m^\top \mathbf{x}))$.
  if $y_t = +1$ **then**
    $N_+^{t+1} = N_+^t + 1$, $N_-^{t+1} = N_-^t$, $B_-^{t+1} = B_-^t$,
    $B_+^{t+1} =$ **UpdateBuffer**$(B_+^t, \mathbf{z}_t, N_+, N_+^{t+1})$,
    $\mathbf{w}_{t+1} =$ **UpdateClassifier**$(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_-^{t+1})$.
  else
    $N_-^{t+1} = N_-^t + 1$, $N_+^{t+1} = N_+^t$, $B_+^{t+1} = B_+^t$,
    $B_-^{t+1} =$ **UpdateBuffer**$(B_-^t, \mathbf{z}_t, N_-, N_-^{t+1})$,
    $\mathbf{w}_{t+1} =$ **UpdateClassifier**$(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_+^{t+1})$.
  end if
end for

formed based on the sampled matrix $\mathbf{C} \in \mathbb{R}^{T\times Q}$. The columns and rows of $\mathbf{K}$ can be rearranged based on this sampling so that $\mathbf{K}$ and $\mathbf{C}$ be written as follows:

$$\mathbf{K} = \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{W} \\ \mathbf{K}_{21} \end{bmatrix}.$$

With this, the rank-k Nyström approximation becomes $\hat{\mathbf{K}} = \mathbf{C}\mathbf{W}_k^\dagger\mathbf{C}^\top \approx \mathbf{K}$, where $\mathbf{W}_k$ is the best rank-k approximation of $\mathbf{W}$, and $\mathbf{W}_k^\dagger$ is the pseudo inverse of the matrix $\mathbf{W}_k$. After approximating the kernel matrix using rank-k Nyström method, the kernel function between two data points can thus be approximated as: $\hat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{C}_i\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}})(\mathbf{C}_j\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}})^\top = (\kappa(\hat{\mathbf{x}}_1, \mathbf{x}_i), \cdots, \kappa(\hat{\mathbf{x}}_Q, \mathbf{x}_i)\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}})(\kappa(\hat{\mathbf{x}}_1, \mathbf{x}_j), \cdots, \kappa(\hat{\mathbf{x}}_Q, \mathbf{x}_j)\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}})^\top$, where $\hat{x}_a$, $a \in \{1, 2\cdots, |S|\}$ is the a-th support vector index. Therefore, a new instance can be represented naturally as:

$$\mathbf{z}_t(\mathbf{x}) = \left([\kappa(\hat{\mathbf{x}}_1, \mathbf{x}), \cdots, \kappa(\hat{\mathbf{x}}_Q, \mathbf{x})]\mathbf{V}_k\mathbf{D}_k^{-\frac{1}{2}}\right)^\top. \tag{4}$$

Now, we can plug the Nyström data representation into the KOAM framework. It is not difficult to notice that the main challenge is how to construct the new feature representation for every newly arrived training instance and formulate the kernel matrix $\hat{\mathbf{K}}$ simultaneously in online scenarios. To tackle this, we propose the following strategy to achieve both goals. In the very beginning, we simply run the regular kernel-based online AUC maximization to maintain and update a kernel-based model $\mathbf{w}_t = \sum_{\mathbf{x}_i\in\mathcal{S}_t}\alpha_i\phi(\mathbf{x}_i)$, where $\mathcal{S}_t$ is the indices of the support vector (SV) set, ϕ is the feature map, i.e., $\langle\phi(\mathbf{x}_1), \phi(\mathbf{x}_2)\rangle = \kappa(\mathbf{x}_1, \mathbf{x}_2)$ (note, we only use $\phi(x_i)$ for convenience), until the size of the SV set reaches the pre-defined sample size Q. Then, we use the maintained Q SVs to approximate the kernel values of other new instances, which is equivalent to applying the first Q columns to approximate the whole kernel matrix. After the low rank approximated kernel matrix is obtained, we construct new feature representation for the incoming instance determined using (4) before applying the KOAM framework. Noted that we should not drop old $\hat{B}_+$ and $\hat{B}_-$ here. We just convert the instances in $\hat{B}_+$ and $\hat{B}_-$ using (4) to get new B_+ and B_-. We label this proposed approach as Nyström Online AUC Maximization (NOAM) as summarized in Algorithm 5.

Algorithm 5 Kernel-based OAM Algorithm with Nyström Method (NOAM)

Input: the penalty parameter η, the sample budget Q, rank approximation k, the maximum buffer size N_+ and N_-, the Gaussian kernel function $\kappa(\cdot,\cdot)$ with parameter σ.
Initialize SV set $\mathcal{S} = \emptyset$, coefficient set of SVs $\alpha = \emptyset$, model $f_1 = 0$, $\mathbf{w}_1 = \mathbf{0}$, $\hat{B}_+^1 = \hat{B}_-^1 = \emptyset$ before Nyström approximation, $B_+^1 = B_-^1 = \emptyset$ after Nyström approximation, $N_+^1 = N_-^1 = 0$.
if $|\mathcal{S}_t| < Q$ **then**
  for $t = 1, 2, \ldots, T_0$ **do**
    Receive a training instance $(\mathbf{x}_t, y_t)$.
    $\mathbf{z}_t = \phi(\mathbf{x}_t)$.
    if $y_t = +1$ **then**
      $N_+^{t+1} = N_+^t + 1$, $N_-^{t+1} = N_-^t$, $\hat{B}_-^{t+1} = \hat{B}_-^t$,
      $\hat{B}_+^{t+1} = \textbf{UpdateBuffer}(\hat{B}_+^t, \mathbf{z}_t, N_+, N_+^{t+1})$,
      $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, \hat{B}_-^{t+1})$.
    else
      $N_-^{t+1} = N_-^t + 1$, $N_+^{t+1} = N_+^t$, $\hat{B}_+^{t+1} = \hat{B}_+^t$,
      $\hat{B}_-^{t+1} = \textbf{UpdateBuffer}(\hat{B}_-^t, \mathbf{z}_t, N_-, N_-^{t+1})$,
      $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, \hat{B}_+^{t+1})$.
    end if
  end for
end if
$\mathbf{w}_{T_0+1} = \sum_{\mathbf{x}_i\in\mathcal{S}_{T_0+1}}\alpha_i\phi(\mathbf{x}_i)$.
Construct the kernel matrix $\mathbf{W}$ from $\mathcal{S}_t$.
$[\mathbf{V}_t, \mathbf{D}_t] = \text{eigs}(\mathbf{W}, k)$, where $\mathbf{V}_t$ and $\mathbf{D}_t$ are Eigenvectors and Eigenvalues of $\mathbf{W}$, respectively.
Initialize $\mathbf{w}_{T_0+1} = (\alpha_1, \cdots, \alpha_Q)(\mathbf{D}_t^{-\frac{1}{2}}\mathbf{V}_t^\top)^{-1}$.
Convert the instances in $\hat{B}_+^{T_0+1}$ and $\hat{B}_-^{T_0+1}$ using (4) to get new $B_+^{T_0+1}$ and $B_-^{T_0+1}$.
for $t = T_0 + 1, \ldots, T$ **do**
  Receive a training instance $(\mathbf{x}_t, y_t)$.
  $\mathbf{z}_t(\mathbf{x}_t) = \mathbf{D}_t^{-\frac{1}{2}}\mathbf{V}_t^\top(\kappa(\mathbf{x}_1, \mathbf{x}_t), \cdots, \kappa(\mathbf{x}_Q, \mathbf{x}_t))^\top$.
  if $y_t = +1$ **then**
    $N_+^{t+1} = N_+^t + 1$, $N_-^{t+1} = N_-^t$, $B_-^{t+1} = B_-^t$,
    $B_+^{t+1} = \textbf{UpdateBuffer}(B_+^t, \mathbf{z}_t, N_+, N_+^{t+1})$,
    $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_-^{t+1})$.
  else
    $N_-^{t+1} = N_-^t + 1$, $N_+^{t+1} = N_+^t$, $B_+^{t+1} = B_+^t$,
    $B_-^{t+1} = \textbf{UpdateBuffer}(B_-^t, \mathbf{z}_t, N_-, N_-^{t+1})$,
    $\mathbf{w}_{t+1} = \textbf{UpdateClassifier}(\mathbf{w}_t, \mathbf{z}_t, y_t, \eta, B_+^{t+1})$.
  end if
end for

IV. Theoretical Analyses

A. Regret Bound for FOAM

Theorem 1. Assume we learn with a Gaussian kernel of bandwidth σ, i.e., $\kappa(\mathbf{x}_1, \mathbf{x}_2) = exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2/2\sigma^2)$, and the input data is bounded by a ball $\mathbb{R}^d$ of diameter R. Let $\ell(\cdot) : \mathbb{R} \to \mathbb{R}$ be a convex loss function that is Lipschitz continuous with Lipschitz constant L. Let $\mathbf{w}_t, t \in [T]$ be the sequence of classifier generated by FOAM in Algorithm 4. For any $f^* = \sum_{t=1}^T \alpha_t\kappa(\mathbf{x}, \mathbf{x}_t)$, we have the following regret bound with probability at least $1 - 2^8(\frac{dR}{\sigma^2\epsilon})^2\exp(-\frac{m\epsilon^2}{4(d+1)})$

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

where $\|f^*\|_1 = \sum_{t=1}^T |\alpha_t|$.

Remark. Generally, the higher the dimensionality m, the higher the probability of the bound to be achieved, which means that sampling more random Fourier features could approximate the kernel function more accurately. Also, setting $\epsilon = \frac{1}{2\sqrt{T}}$ requires sampling $m = O(T)$ random components to achieve a high probability, leading to high-dimensionality in the new feature space. Even this, the learning time for

each instance $\mathcal{O}(c_1 T)$ is much less than the time cost for classification by regular kernel learning framework $\mathcal{O}(c_2 T)$, where c_1 is the time complexity for a scalar product in FOAM while c_2 is the time complexity for computing the kernel function including the kernel matrix. Since $c_1 \gg c_2$, the proposed FOAM is still much faster than the KOAM based on regular kernel learning techniques.

B. Regret Bound for NOAM

Theorem 2. Assume we learn with kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) \leq 1$. Let $\ell_t(\cdot) : \mathbb{R} \to \mathbb{R}$ be a convex loss function that is Lipschitz continuous with Lipschitz constant L. Let the sequence of T instances $\mathbf{x}_1, \cdots, \mathbf{x}_T$ form a kernel matrix $\mathbf{K} \in \mathbb{R}^{T \times T}$, and $\mathbf{K}_k$ is the best rank-k approximation of $\mathbf{K}$, $\mathbf{K}_{max} = \max_i \mathbf{K}_{ii}$, $d^{\mathbf{K}}_{\max} = \max_{ij} \sqrt{\mathbf{K}_{ii} + \mathbf{K}_{jj} - 2\mathbf{K}_{ij}}$. Let $\mathbf{w}_t, t \in [T]$ be the sequence of classifier generated by NOAM in Algorithm 5 with budget size Q. For f^* that minimize $\frac{1}{2}\|f\|_{\mathcal{H}} + \frac{\lambda}{T}\sum_{t=1}^{T} \mathcal{L}_t(f)$ with probability at least $1 - \epsilon$, we have

$$\sum_{t=1}^{T}[\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(f^*)] \leq (\frac{\|f^*\|_1 + L^2}{2} + L)\sqrt{T} + \mathcal{O}(\sqrt{T}).$$

Remark. The above sun-linear regret $O(\sqrt{T})$ may be better than that in FOAM since NOAM does not need a large value of Q to achieve the $O(\sqrt{T})$ bound.

V. Experimental Results

In this section, we evaluate the empirical performance of the proposed algorithms: FOAM and NOAM, by comparing them to existing state-of-the-art algorithms in terms of AUC performance and parameter sensitivity. In addition, we also examine the performance of both methods with varied number of random samples so as to verify the theoretical differences between these two methods.

A. Algorithms Considered for Comparison

The algorithms considered in our experiments include:

- **OAM$_{\text{seq}}$**: The buffer-based OAM algorithm with reservoir sampling and sequential updating method [4].
- **OAM$_{\text{gra}}$**: The buffer-based OAM algorithm with reservoir sampling and online gradient updating method [4].
- **OPAUC**: The one-pass AUC optimization algorithm with square loss function [5].
- **AdaOAM**: The adaptive gradient method for online AUC maximization [6].
- **KOIL**: The buffer-based kernelized online imbalanced learning algorithm for AUC maximization [7].
- **FOAM**: The proposed random Fourier feature method for surrogate kernel-based online AUC maximization.
- **NOAM**: The proposed Nyström method for surrogate kernel-based online AUC maximization.

B. Experimental Testbed and Setup

To examine the performance of FOAM and NOAM compared with other existing state-of-the-art methods, we conduct extensive experiments on 12 benchmark datasets by maintaining consistency to the previous studies on online AUC maximization. Table I gives the detailed characteristics of the 12 binary-class datasets used in our experimental studies. All of these datasets can be downloaded from LIBSVM [1], UCI machine learning repository [2], and KDD CUP competition website [3]. These datasets are chosen fairly randomly to cover a variety of domains with different characteristics. Note that several datasets (vowel, vehicle, shuttle, satimage, pendigits) are originally multi-class, which are converted to class-imbalanced binary datasets in our experiments.

TABLE I: Details of benchmark datasets.

datasets	# inst	# dim	T_-/T_+
vowel	528	10	10.000
vehicle	846	18	3.251
svmguide3	1,243	22	3.199
spambase	4,601	56	1.538
satimage	6,435	36	3.137
pendigits	10,992	16	8.620
magic04	19,020	10	1.843
shuttle	43,500	9	3.6316
KDDCUP08	102,294	117	163.196
ijcnn1	141,691	22	9.303
covtype	581,012	54	1.051
KDDCUP99	1,131,571	127	6.628

In the experiments, the features have been normalized, i.e., $\mathbf{x}_t \leftarrow \mathbf{x}_t/\|\mathbf{x}_t\|$, which is reasonable since instances are received sequentially in online setting. Each dataset has been randomly divided into 5 folds, in which 4 folds are for training and the remaining is for testing. We also generate 4 independent 5-fold partitions per dataset to further reduce the effects of random partition on the algorithms. Therefore, the reported AUC results are averaged of the 20 runs for each dataset. 5-fold cross validation is conducted on the training sets to decide on the appropriate learning rate $\eta \in 2^{[-10:10]}$. For all the algorithms except OPAUC and AdaOAM, the buffer size is fixed at 100. The parameter m in FOAM is set to Q in NOAM, and the parameter k in NOAM is set to $0.4Q$. All experiments for online setting comparisons were conducted on the MATLAB platform and a workstation with 16GB memory and 3.20GHz CPU.

C. Evaluation on Benchmark Datasets

Table II summarizes the average AUC performance of the algorithms in comparison, across all the 12 datasets in online settings. Note that 'N/A' means that no result was attained after a running time of 10^5 seconds or the memory overflow occurs due to heavy memory requirements of some algorithms. From Table II, several observations have been drawn.

First of all, in terms of AUC performance for online AUC maximization tasks, we found that kernel-based methods including KOIL, FOAM, and NOAM, generally fare better than their non-kernelized counterparts, which justifies the motivation behind the use of kernel techniques for solving OAM

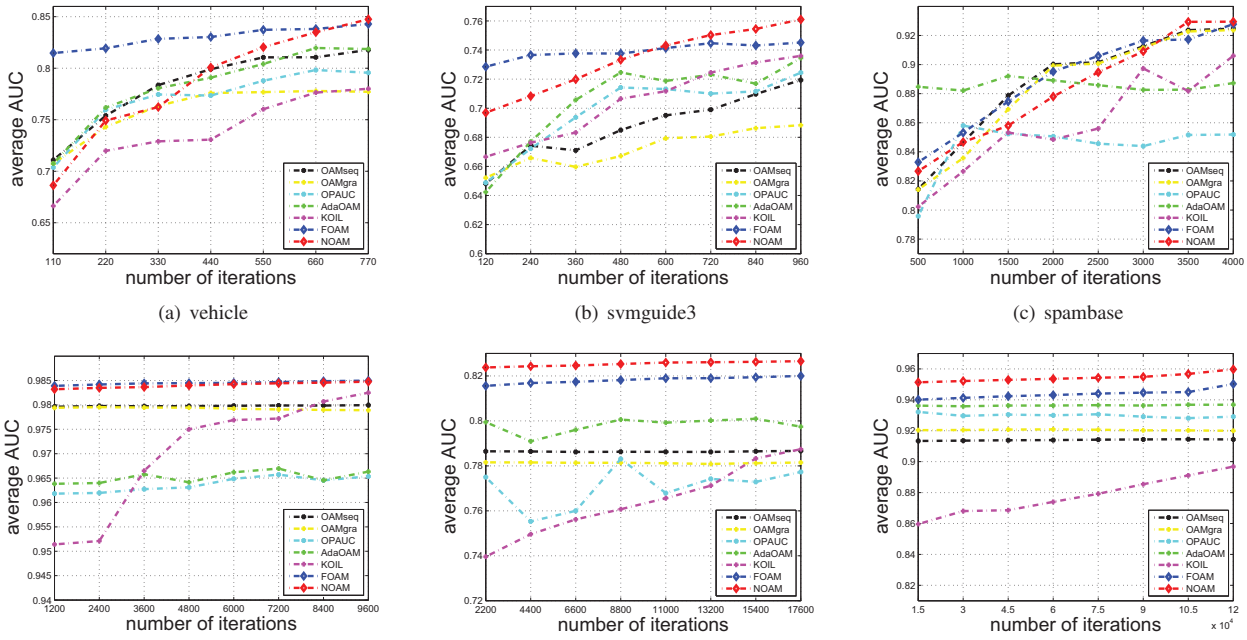
[1] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/
[2] http://www.ics.uci.edu/~mlearn/MLRepository.html
[3] http://www.sigkdd.org/kddcup/index.php

TABLE II: Evaluation of AUC performance on benchmark datasets.

Algorithm	vowel		vehicle		svmguide3	
	AUC	Time Cost(s)	AUC	Time Cost(s)	AUC	Time Cost(s)
OAM$_{seq}$	0.9013± 0.0415	0.4878	0.8071± 0.0372	0.9222	0.7171 ± 0.0366	1.3507
OAM$_{gra}$	0.9075± 0.0394	0.4514	0.8090± 0.0246	0.8413	0.7207 ± 0.0392	1.2742
OPAUC	0.8892± 0.0503	0.0235	0.8168± 0.0257	0.0428	0.7090 ± 0.0432	0.0682
AdaOAM	0.9032± 0.0471	0.0381	0.8196± 0.0264	0.075	0.7310 ± 0.0382	0.0980
KOIL	**0.9945± 0.0027**	0.4677	0.7909± 0.0389	4.4244	0.7184 ± 0.0548	7.8424
FOAM	0.9300± 0.0360	0.6219	0.8418± 0.0248	1.2407	0.7476 ± 0.0352	1.9704
NOAM	0.9198± 0.0384	0.3235	**0.8423± 0.0251**	0.5852	**0.7528 ± 0.0344**	1.1113
Algorithm	spambase		satimage		pendigits	
	AUC	Time Cost(s)	AUC	Time Cost(s)	AUC	Time Cost(s)
OAM$_{seq}$	0.9254± 0.0089	5.8854	0.9955± 0.0016	5.8843	0.9775 ± 0.0034	13.0094
OAM$_{gra}$	0.9251± 0.0092	5.7011	0.9956± 0.0020	5.4609	0.9778 ± 0.0032	12.4870
OPAUC	0.8419± 0.0141	0.4924	0.9947± 0.0020	0.3074	0.9604 ± 0.0061	0.6616
AdaOAM	0.8874± 0.0093	0.6519	0.9955± 0.0018	0.4581	0.9664 ± 0.0056	0.6383
KOIL	0.9065± 0.0224	68.5431	0.9642± 0.0036	9.8999	0.9830 ± 0.0035	174.7840
FOAM	0.9271± 0.0081	8.0887	0.9970± 0.0014	7.8983	0.9838 ± 0.0041	18.0551
NOAM	**0.9294± 0.0088**	6.4919	**0.9977± 0.0016**	5.5976	**0.9854 ± 0.0044**	13.0201
Algorithm	magic04		shuttle		KDDCUP08	
	AUC	Time Cost(s)	AUC	Time Cost(s)	AUC	Time Cost(s)
OAM$_{seq}$	0.7889± 0.0116	23.3121	0.9812± 0.0014	55.1749	0.9062 ± 0.0133	116.1822
OAM$_{gra}$	0.7800± 0.0126	22.2374	0.9814± 0.0013	52.5216	0.9094 ± 0.0148	114.1183
OPAUC	0.7636± 0.0084	1.1869	0.9844± 0.0015	2.9681	0.9040 ± 0.0122	22.3601
AdaOAM	0.7980± 0.0072	1.6779	0.9846± 0.0015	4.1887	0.9079 ± 0.0122	26.7904
KOIL	0.7956± 0.0227	266.9989	0.9863± 0.0021	503.6254	0.8687 ± 0.0115	903.7449
FOAM	0.8227± 0.0089	33.8771	0.9929± 0.0016	69.4108	0.9188 ± 0.0134	164.5357
NOAM	**0.8294± 0.0133**	24.8760	**0.9931± 0.0014**	51.7314	**0.9214 ± 0.0127**	143.7298
Algorithm	ijcnn1		covtype		KDDCUP99	
	AUC	Time Cost(s)	AUC	Time Cost(s)	AUC	Time Cost(s)
OAM$_{seq}$	0.9211± 0.0083	176.3466	0.7978± 0.0138	2272.5596	0.9956 ± 0.0013	4519.3686
OAM$_{gra}$	0.9224± 0.0078	174.8347	0.7913± 0.0163	2195.9358	0.9953 ± 0.0017	4448.1765
OPAUC	0.9363± 0.0022	63.4248	0.7839± 0.0093	533.4639	0.9954 ± 0.0007	915.4200
AdaOAM	0.9366± 0.0022	69.0924	0.8059± 0.0087	547.1813	0.9956 ± 0.0018	979.1088
KOIL	0.8968± 0.0280	1468.2975	N/A	N/A	N/A	N/A
FOAM	0.9502± 0.0048	243.5067	0.8087± 0.0082	2286.1813	**0.9960 ± 0.0021**	4466.7716
NOAM	**0.9599± 0.0029**	193.8069	**0.8105± 0.0075**	2041.5380	0.9956 ± 0.0018	4103.3958

tasks. Among all the algorithms considered, the proposed FOAM and NOAM reported the best AUC values on almost all the 12 datasets, thus highlighting the efficacy of surrogate kernel-based learning models for solving OAM tasks.

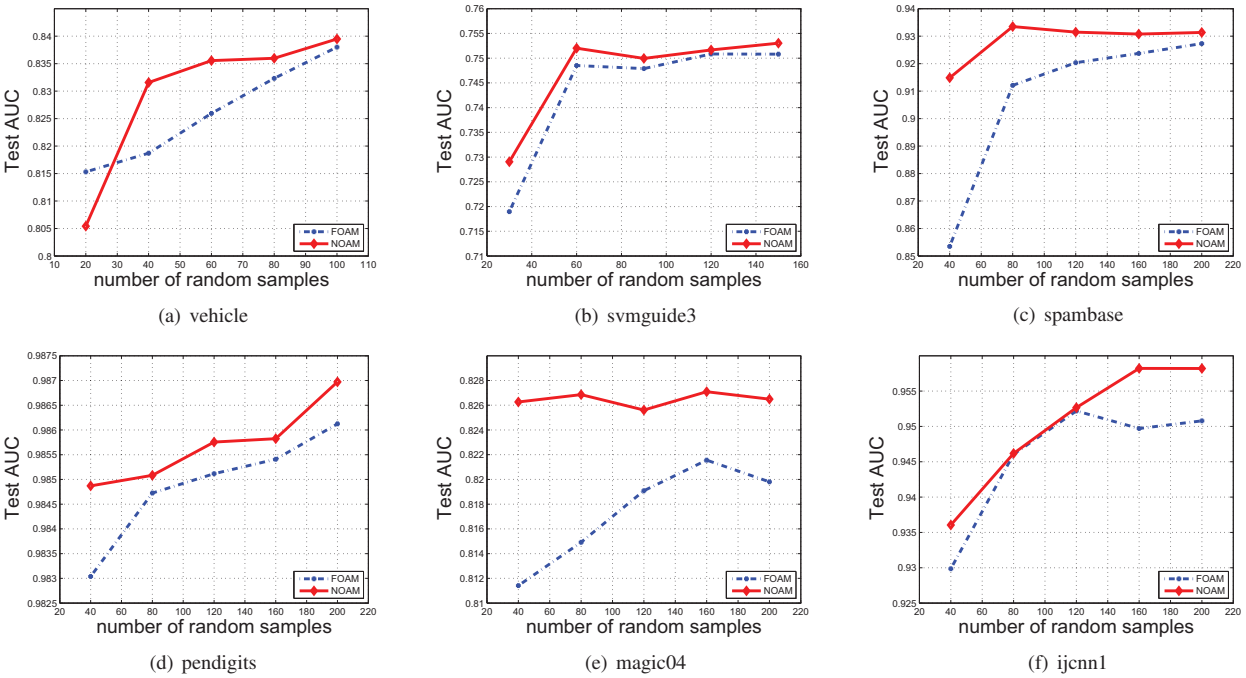
Second, in terms of efficiency, the time costs of the non-buffer based algorithms (OPAUC and AdaOAM) are much lower than the other five buffer-based algorithms. This can be mainly attributed to their differences in loss functions and thus the differing updating and maintenance schemes considered. However, for the buffer-based algorithms, it is noted that the functional kernel approximation approaches (FOAM and NOAM) are more efficient than the regular kernelized AUC learning approaches with fixed size budgets (KOIL), while comparable with the non-kernelized methods (OAM$_{seq}$ and



(a) vehicle (b) svmguide3 (c) spambase

(d) pendigits (e) magic04 (f) ijcnn1

Fig. 1: Evaluation of convergence rate on selected benchmark datasets.



(a) vehicle (b) svmguide3 (c) spambase

(d) pendigits (e) magic04 (f) ijcnn1

Fig. 2: Comparison of FOAM and NOAM with varied number of random samples.

OAM_{gra}), especially on the large datasets. This observation highlights the higher efficiency of the random Fourier features and Nyström method over conventional kernel-based online learning methods. Between FOAM and NOAM on time costs, NOAM generally consumes less time to train than FOAM, mainly due to the lower dimensionality of the weight model ($0.4Q$) than that of FOAM (Q). This observation coincides well with our theoretical analysis that NOAM does not require

a large Q to achieve high performance. On the very large datasets such as covtype and KDDCUP99, the regular kernelized method KOIL fails to return any useful results within the available computational budget considered in the experimental studies. This shows that KOIL does not scale well on large datasets in practice due to the high computational time and memory incurred on the large kernel matrix.

Furthermore, it is also observed that the NOAM almost always outperforms the FOAM (except on the "vowel" and "KDDCUP99" datasets). This can be explained by the fundamentally different approximation strategies between the random Fourier features and the Nyström method. Specifically, random Fourier features approach samples basis functions from a data-independent distribution to form a data-independent representation, while the Nyström method samples a subset of the training instances which constitute to the basis functions that form a data-dependent representation. Based on this deduction, it is natural for the empirical AUC performance of the NOAM to exceed that of the FOAM, which is consistent to the theoretical comparison made between the random Fourier features and the Nyström method for large scale kernel learning [24].

We now study the online performance of all the algorithms considered in the present work. In particular, the results of these algorithms on 6 of the datasets are summarized in Figure 1 for further discussions. Specifically, the reported AUC performance is the average AUC values of the online models on the test datasets, based on 20 runs. From the results, once again both the proposed FOAM and NOAM algorithms are observed to significantly outperform the other methods in online mode.

D. Evaluation of FOAM and NOAM with Varied Number of Random Samples

To analyze the differences between FOAM and NOAM, the resultant AUC performance with varied number of random samples are studied. The results are summarized in Figure 2. Note that in our study we restrict the maximum number of random samples to vary from 100 to 200, even for large datasets, which is empirically verified to guarantee good AUC performance. From the curve trends of the figures, it is observed that a higher number of the random features leads to the higher AUC values. This indicates that a higher number of random features in the algorithms can be beneficial to the AUC performance. It is also observed that on all datasets, the NOAM based on the Nyström method outperforms the FOAM based on the random Fourier features method with different sizes of random samples. Notably, the results further confirms the superiority of the data-dependent sampling method over the data-independent sampling counterpart.

VI. Conclusion

This paper investigates a series of kernel-based online AUC maximization (KOAM) algorithms, which aim to address OAM issue by projection the data into the new feature space. We have proposed two surrogate kernel-based learning models, namely, (i) the Fourier Online AUC Maximization (FOAM) algorithm which samples the basis functions from a data-independent distribution to approximate the kernel function; and (ii) the Nyström Online AUC Maximization (NOAM) algorithm which samples a subset of instances from the training data to approximate the kernel matrix with a low rank matrix. To further control the noise and variance, we adopted the mini-batch OGD method to update the model. We compared the differences between the two proposed algorithms both theoretically and empirically on a variety of large scale datasets. The encouraging results highlighted the high efficacy and efficiency of the proposed algorithms over existing state-of-the-art approaches for online learning .

VII. Appendix

A. Proof of Theorem 1

Proof. Given $f^*(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \kappa(\mathbf{x}, \mathbf{x}_t)$, according to the Representer Theorem [34], we have a corresponding linear model: $\mathbf{w}^* = \sum_{t=1}^{T} \alpha_t \mathbf{z}_t(\mathbf{x}_t)$, where $\mathbf{z}_t(\mathbf{x}) = (\cos(\mathbf{u}_1^\top \mathbf{x}), \sin(\mathbf{u}_1^\top \mathbf{x}), \cdots, \cos(\mathbf{u}_m^\top \mathbf{x}), \sin(\mathbf{u}_m^\top \mathbf{x}))$, To prove our theorem, the first step is to bound the regret between the sequence of linear model $\mathbf{w}_t$ learned by our online learner and the linear model $\mathbf{w}^*$ learned in the new feature space.

To begin with, we have

$$\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 = \|\mathbf{w}_t - \eta \nabla \mathcal{L}_t(\mathbf{w}_t) - \mathbf{w}^*\|^2$$
$$= \|\mathbf{w}_t - \mathbf{w}^*\|^2 + \eta^2 \|\nabla \mathcal{L}_t(\mathbf{w}_t)\|^2 - 2\eta \nabla \mathcal{L}_t^\top(\mathbf{w}_t)(\mathbf{w}_t - \mathbf{w}).$$

Combining this with the convexity of the loss function: $\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(\mathbf{w}) \leq \nabla \mathcal{L}_t^\top(\mathbf{w}_t)(\mathbf{w}_t - \mathbf{w})$, we have the following

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

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

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

Now, we study the relationship between $\sum_{t=1}^{T} \mathcal{L}_t(\mathbf{w}^*)$ and $\sum_{t=1}^{T} \mathcal{L}_t(f^*)$. According to *Claim 1* on the uniform convergence of Fourier features in [22], we have a high probability bound for the difference between the approximated kernel value and the true kernel value, i.e., with probability at least $1 - 2^8 (\frac{dR}{\sigma^2 \epsilon})^2 \exp(-\frac{m\epsilon^2}{4(d+1)})$, we have $\forall i, j$

$$|\mathbf{z}_i^\top \mathbf{z}_j - \kappa(\mathbf{x}_i, \mathbf{x}_j)| < \epsilon. \tag{6}$$

When (6) holds, we have

$$\begin{aligned} &\sum_{t=1}^{T} [\mathcal{L}_t(\mathbf{w}^*) - \mathcal{L}_t(f^*)] \\ &= \sum_{t=1}^{T} \sum_{\mathbf{z}_i \in B^t_{-y_t}} \frac{[\ell_t(y_t \mathbf{w}_t^{*\top}(\mathbf{z}_t - \mathbf{z}_i)) - \ell_t(y_t(f^*(\mathbf{x}_t) - f^*(\mathbf{x}_i)))]}{|B^t_{-y_t}|} \\ &\leq \sum_{t=1}^{T} \sum_{\mathbf{z}_i \in B^t_{-y_t}} \frac{|\ell_t(y_t \mathbf{w}_t^{*\top}(\mathbf{z}_t - \mathbf{z}_i)) - \ell_t(y_t(f^*(\mathbf{x}_t) - f^*(\mathbf{x}_i)))|}{|B^t_{-y_t}|} \\ &\leq \sum_{t=1}^{T} \sum_{\mathbf{z}_i \in B^t_{-y_t}} L \sum_{j=1}^{T} \frac{|\alpha_j \mathbf{z}_j^\top(\mathbf{z}_t - \mathbf{z}_i) - \alpha_j(\kappa(\mathbf{x}_j, \mathbf{x}_t) - \kappa(\mathbf{x}_j, \mathbf{x}_i))|}{|B^t_{-y_t}|} \\ &\leq \sum_{t=1}^{T} \sum_{\mathbf{z}_i \in B^t_{-y_t}} L \sum_{j=1}^{T} \frac{|\alpha_j||\mathbf{z}_j^\top \mathbf{z}_t - \kappa(\mathbf{x}_j, \mathbf{x}_t)| + |\alpha_j||\mathbf{z}_j^\top \mathbf{z}_i - \kappa(\mathbf{x}_j, \mathbf{x}_i)|}{|B^t_{-y_t}|} \\ &\leq \sum_{t=1}^{T} \sum_{\mathbf{z}_i \in B^t_{-y_t}} L \sum_{j=1}^{T} \frac{2|\alpha_j|}{|B^t_{-y_t}|} \leq 2\epsilon L T \|f^*\|_1. \end{aligned} \tag{7}$$

Combining (5) and (7), we obtain

$$\sum_{t=1}^{T}[\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(f^*)] \leq \frac{\|\mathbf{w}\|^2}{2\eta} + \frac{\eta}{2}L^2T + 2\epsilon LT\|f^*\|_1.$$

By setting $\eta = \frac{1}{\sqrt{T}}$ and $\epsilon = \frac{1}{2\sqrt{T}}$, we have

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

which completes the proof with sub-linear regret $O(\sqrt{T})$. □

B. Proof of Theorem 2

Proof. Similar with 1, we bound the regret concerning $\mathbf{w}^*$ in new feature space as in (5) before bounding $\sum_{t=1}^{T}[\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(f^*)]$. As *Corollary 1* shown in [35], we could bound the gap between the suffered loss above by approximation with respect to the spectral norm of kernel approximation gap:

$$\sum_{t=1}^{T}[\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(\mathbf{w}^*)] \leq \sqrt{2}\lambda L\|\hat{\mathbf{K}} - \mathbf{K}\|_2^{\frac{1}{4}}(1 + (\frac{\|\hat{\mathbf{K}} - \mathbf{K}\|_2}{4})^{\frac{1}{4}}). \quad (8)$$

Then, we should bound the spectral norm of the approximated kernel gap $\|\hat{\mathbf{K}} - \mathbf{K}\|_2$. As *Theorem 2* in [35] shows, this could be bounded by the spectral norm of the best rank-k approximated kernel gap $\|\mathbf{K} - \mathbf{K}_k\|_2 = \sigma_{k+1}$: $\|\hat{\mathbf{K}} - \mathbf{K}\|_2 \leq \sigma_{k+1} + \Delta_Q$, where Δ_Q is

$$\Delta_Q = \frac{2T}{\sqrt{Q}}\mathbf{K}_{max}(1 + \sqrt{\frac{T-Q}{T-0.5}\frac{1}{\beta(Q,T)}\log\frac{1}{\epsilon}\frac{d_{\max}^{\mathbf{K}}}{\mathbf{K}_{max}^{\frac{1}{2}}}}). \quad (9)$$

$\beta(Q,T) = 1 - \frac{1}{2\max\{Q,T-Q\}}$ and $\|\mathbf{K}\|_2$ is the spectral norm of $\mathbf{K}$. Combining (7), (8), (9), we obtain

$$\sum_{t=1}^{T}[\mathcal{L}_t(\mathbf{w}_t) - \mathcal{L}_t(f^*)] \leq \frac{\|f^*\|_1^2}{\eta} + \frac{\eta}{2}L^2T + \sqrt{2}\lambda L(\sigma_{k+1} + \Delta_Q)^{\frac{1}{4}}(1 + \frac{\sigma_{k+1}+\Delta_Q}{4}).$$

Obviously, the larger the size of Q, the smaller value of Δ_Q, leading to a tighter bound. Mostly $\Delta_Q = O(T)$ and thus $\sqrt{2}\lambda L(\sigma_{k+1} + \Delta_Q)^{\frac{1}{4}}(1 + \frac{\sigma_{k+1}+\Delta_Q}{4}) = O(\sqrt{T})$. By setting $\eta = \frac{1}{\sqrt{T}}$, we complete the proof. □

ACKNOWLEDGMENT

This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative.

REFERENCES

[1] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under of receiver operating characteristic (roc) curve." in *Radiology*, 1982.

[2] C. Cortes and M. Mohri, "Auc optimization vs. error rate minimization," in *NIPS*, 2003.

[3] T. Joachims, "A support vector method for multivariate performance measures," in *ICML*, 2005, pp. 377–384.

[4] P. Zhao, S. C. H. Hoi, R. Jin, and T. Yang, "Online AUC maximization," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2*, 2011, pp. 233–240.

[5] W. Gao, R. Jin, S. Zhu, and Z.-H. Zhou, "One-pass auc optimization," in *ICML (3)*, 2013, pp. 906–914.

[6] Y. Ding, P. Zhao, S. C. H. Hoi, and Y. Ong, "An adaptive gradient method for online AUC maximization," in *AAAI 2015, January 25-30, Austin, Texas, USA.*, 2015, pp. 2568–2574.

[7] J. Hu, H. Yang, I. King, M. R. Lyu, and A. M. So, "Kernelized online imbalanced learning with fixed budgets," in *AAAI 2015, January 25-30, Austin, Texas, USA.*, 2015, pp. 2666–2672.

[8] P. Kar, B. K. Sriperumbudur, P. Jain, and H. Karnick, "On the generalization ability of online learning algorithms for pairwise loss functions," in *ICML 2013, Atlanta, GA, USA, 16-21 June*, 2013, pp. 441–449.

[9] S. C. H. Hoi, J. Wang, and P. Zhao, "LIBOL: a library for online learning algorithms," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 495–499, 2014.

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

[11] S. Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

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

[14] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in *ICML*, Helsinki, 2008, pp. 264–271.

[15] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," in *NIPS 2009, 7-10 December, Vancouver, British Columbia, Canada.*, 2009, pp. 414–422.

[16] J. C. Duchi and Y. Singer, "Efficient online and batch learning using forward backward splitting," *Journal of Machine Learning Research*, vol. 10, pp. 2899–2934, 2009.

[17] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 567–599, 2013.

[18] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[19] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," in *NIPS*, 2001, pp. 785–792.

[20] K. Crammer, J. S. Kandola, and Y. Singer, "Online classification on a budget," in *NIPS 2003, December 8-13,Vancouver and Whistler, British Columbia, Canada*, 2003, pp. 225–232.

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

[22] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS 2007, Vancouver, British Columbia, Canada, December 3-6*, 2007, pp. 1177–1184.

[23] C. K. I. Williams and M. W. Seeger, "Using the nyström method to speed up kernel machines," in *NIPS*, 2000, pp. 682–688.

[24] T. Yang, Y. Li, M. Mahdavi, R. Jin, and Z. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison"," in *NIPS2012*, 2012, pp. 485–493.

[25] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[26] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *ICML 2003, August 21-24, 2003, Washington, DC, USA*, 2003, pp. 928–936.

[27] C. Wang, X. Chen, A. J. Smola, and E. P. Xing, "Variance reduction for stochastic gradient optimization," in *NIPS 2013, December 5-8, Lake Tahoe, Nevada, United States.*, 2013, pp. 181–189.

[28] W. Rudin, *Fourier Analysis on Groups*, ser. A Wiley-interscience publication. Wiley, 1990.

[29] L. Bo and C. Sminchisescu, "Efficient match kernel between sets of features for visual recognition," in *NIPS 2009, Vancouver, British Columbia, Canada, December 7-10*, 2009, pp. 135–143.

[30] S. C. H. Hoi, J. Wang, P. Zhao, J. Zhuang, and Z. Liu, "Large scale online kernel classification," in *IJCAI 2013, Beijing, China, August 3-9*, 2013.

[31] J. Lu, S. C. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu, "Large scale online kernel learning," *Journal of Machine Learning Research*, vol. 17, no. 47, pp. 1–43, 2016.

[32] R. Chitta, R. Jin, and A. K. Jain, "Efficient kernel clustering using random fourier features," in *ICDM 2012, Brussels, Belgium, December 10-13*, 2012, pp. 161–170.

[33] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *NIPS 2009, Vancouver, British Columbia, Canada, December 7-10*, 2009, pp. 1509–1517.

[34] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *COLT*, 2001, pp. 416–426.

[35] C. Cortes, M. Mohri, and A. Talwalkar, "On the impact of kernel approximation on learning accuracy," in *AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15*, 2010, pp. 113–120.