DISSERTATION

SCALABLE LARGE MARGIN PAIRWISE LEARNING ALGORITHMS

Submitted by

Majdi Khalid Alnnfiai

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2019

ABSTRACT

SCALABLE LARGE MARGIN PAIRWISE LEARNING ALGORITHMS

Classification is a major task in machine learning and data mining applications. Many of these applications involve building a classification model using a large volume of imbalanced data. In such an imbalanced learning scenario, the area under the ROC curve (AUC) has proven to be a reliable performance measure to evaluate a classifier. Therefore, it is desirable to develop scalable learning algorithms that maximize the AUC metric directly.

The kernelized AUC maximization machines have established a superior generalization ability compared to linear AUC machines. However, the computational cost of the kernelized machines hinders their scalability. To address this problem, we propose a large-scale nonlinear AUC maximization algorithm that learns a batch linear classifier on approximate feature space computed via the k-means Nyström method. The proposed algorithm is shown empirically to achieve comparable AUC classification performance or even better than the kernel AUC machines, while its training time is faster by several orders of magnitude.

However, the computational complexity of the batch linear model compromises its scalability when training sizable datasets. Hence, we develop a second-order online AUC maximization algorithms based on a confidence-weighted model. The proposed algorithms exploit the second-order information to improve the convergence rate and implement a fixed-size buffer to address the multivariate nature of the AUC objective function. We also extend our online linear algorithms to consider an approximate feature map constructed using random Fourier features in an online setting. The results show that our proposed algorithms outperform or are at least comparable to the competing online AUC maximization methods.

Despite their scalability, we notice that online first and second-order AUC maximization methods are prone to suboptimal convergence. This can be attributed to the limitation of the hypothesis space. A potential improvement can be attained by learning stochastic online variants. However, the vanilla stochastic methods also suffer from slow convergence because of the high variance introduced by the stochastic process.

We address the problem of slow convergence by developing a fast convergence stochastic AUC maximization algorithm. The proposed stochastic algorithm is accelerated using a unique combination of scheduled regularization update and scheduled averaging. The experimental results show that the proposed algorithm performs better than the state-of-the-art online and stochastic AUC maximization methods in terms of AUC classification accuracy.

Moreover, we develop a proximal variant of our accelerated stochastic AUC maximization algorithm. The proposed method applies the proximal operator to the hinge loss function. Therefore, it evaluates the gradient of the loss function at the approximated weight vector. Experiments on several benchmark datasets show that our proximal algorithm converges to the optimal solution faster than the previous AUC maximization algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Machine learning is fundamentally about building statistical models that are able to learn and forecast in an automatic manner. Classification is one of the primary machine learning tasks. The development of efficient, effective, and robust classification algorithms is of paramount importance in designing intelligent systems that can be deployed in a variety of applications, such as recommender systems, bioinformatics, information retrieval, computer vision, fraud detection, social network analysis, and medical diagnosis.

In the era of "big data," most of the real-world applications involve building a predictive model using a tremendous amount of data. This large volume of data makes most classical learning algorithms impractical. For example, the complexity of training nonlinear kernel support vector machines grows quadratically with the number of instances. Further, the complexity of their predictive models depend on the set of support vectors, which grows linearly with the number of training instances.

The support vector machines (SVM) literatures contain different approaches to scale up classical kernel SVM from different perspectives. This includes methods that approximate the kernel matrix using low-rank approximation, such as Nyström methods [1, 2] and incomplete Cholesky factorization [3]. Instead of approximating the kernel matrix, other methods such as the random kitchen sink [4] approximate the kernel function by projecting the input space into a higher dimensional space via a random matrix. The works in [5, 6] scale up nonlinear kernel SVM by building sparse kernels. Recent approaches exploit the local structure of the data (i.e., clusters and manifolds) [7, 8, 9] to approximate complex nonlinear decision boundaries using linear classifiers.

Online learning has also been used to approximate kernel SVM by training an online model using a predefined number of support vectors [10] or using online kernel approximation techniques [11]. Another recent method addresses the scalability of kernel SVM by integrating random features and a functional gradient [12], where random features are computed on the fly for a sample of data in each iteration. In this stochastic setting, the random features can be increased in each iteration.

Most of the recent scalable learning algorithms focus on optimizing the error rate or accuracy. These algorithms fail to construct a robust model when applied to imbalanced datasets. For imbalanced learning, the AUC has been shown to be a reliable measure to evaluate the performance of classifiers [13]. The direct optimization of the AUC measure requires a learning algorithm that can deal with pairwise loss functions instead of classical univariate loss functions that are defined based on a single instance.

In the following subsections, we discuss the importance of developing learning algorithms that directly optimize the AUC measure. We then briefly summarize our contributions in this line of research.

1.1 Motivation

This research aims to devise supervised classification algorithms that address both large-scale and heavily imbalanced class distribution data. The thesis focuses on maximum margin classifiers, such as SVM [14, 15]. The concept of maximum margin has a sound theoretical foundation in reducing the uncertainty of the predictive model [16], which in turns improves the generalization capability of the classifier. Among several metrics that can be optimized by the SVM classifier, this thesis is restricted to optimizing the area under the ROC curve (AUC) metric [17] to deal with imbalanced datasets.

For binary classification task, the vanilla SVM, which minimizes the error rate, trained on an extremely imbalanced dataset is prone to a bias problem, which results in a low classification accuracy for the minority class. This bias phenomenon is demonstrated in Figure 1.1. The figure shows a linear SVM classifier trained to maximize the accuracy on one-dimensional imbalanced data. The classifier is accurate in classifying the majority class (blue points) but at the expense of the minority class (red points). Further, the error rate or accuracy becomes a misleading indicator of the quality of the predictive model as the ratio of one class label outnumbers the another class label. In Figure 1.2, we can notice that as the ratio of the positive to negative is increasing as the accuracy of the vanilla SVM classifier decreases, while the SVM classifier trained to maximize the AUC shows a reliable classification accuracy, obtained at the optimal operating point.

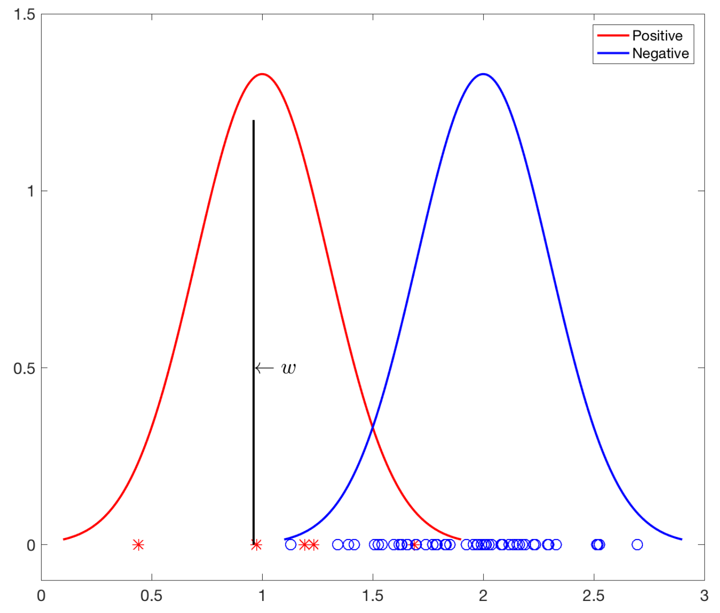


Figure 1.1: Linear classifier trained on one-dimensional imbalanced data is shown to be biased. The one-dimensional data comprise positive instances, the minority, and negative instances, the majority. Each class is generated from normal distributions. The underlying distributions of both positive and negative instances are shown. The minority class includes five instances while the majority has 100 instances. The solid line represents the linear SVM classifier. The authors of [18] describe a similar figure to support their argument about the bias problem.

Re-sampling techniques such as under-sampling [20] and over-sampling [21] have been proposed to deal with imbalanced datasets, but these methods are not efficient. Cost sensitive classifiers [22, 23] have also been developed for imbalanced learning. However, these methods require prior knowledge about the class distribution, which is hard to obtain in real-world applications. Therefore, it is important to develop learning algorithms that can directly optimize the proper measures for imbalanced learning.

Unlike error rate, the AUC metric does not consider the class distribution when assessing the performance of classifiers. Therefore, the AUC metric is a threshold-independent measure. In fact,

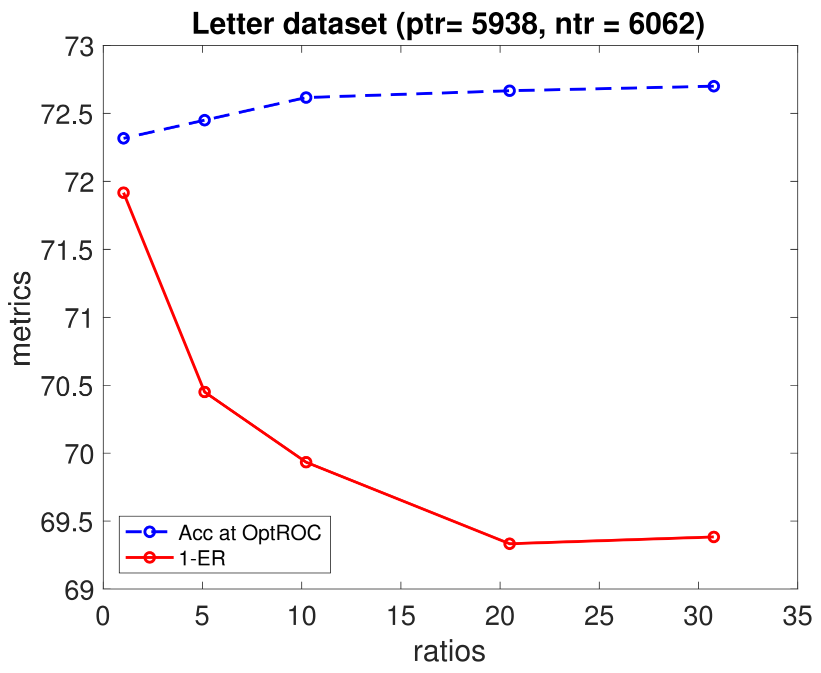


Figure 1.2: Comparison of classification accuracy for two linear SVM classifiers on the letter test set. The letter dataset is obtained from [9]. Both classifiers are trained on the training dataset with varying ratios of positive to negative classes. The number of positive and negative training instances is 5938 and 6062, respectively. The first classifier optimizes the AUC metric, and its classification accuracies, shown by the blue dashed line, are obtained using the threshold corresponding to the optimal operating point. The second classifier optimizes the accuracy, and its classification accuracies are shown by the solid red line.

it evaluates a classifier over all possible thresholds, hence eliminating the effect of imbalanced class distribution. This property renders the AUC a reliable measure to evaluate classification performance on datasets with strongly unbalanced classes [13], which are not uncommon in real-world applications such as recommender systems [24], bioinformatics[25], continuous integration systems [26] ,and anomaly detection [27].

A typical recommender system consists of a graphical user interface (GUI) and an internal recommendation policy. A high level paradigm of recommender systems is shown in Figure 1.3 The recommendation policy can be constructed using a machine learning algorithm, which makes predictions that will be displayed by the GUI. The GUI also feeds the learning algorithm with any feedback solicited from the users (e.g., click, rating). For example, in personalized advertisement recommendation (PAR), a user interacts with the system by visiting a web page where the system displays a predicted ad from a set of ads for the user.

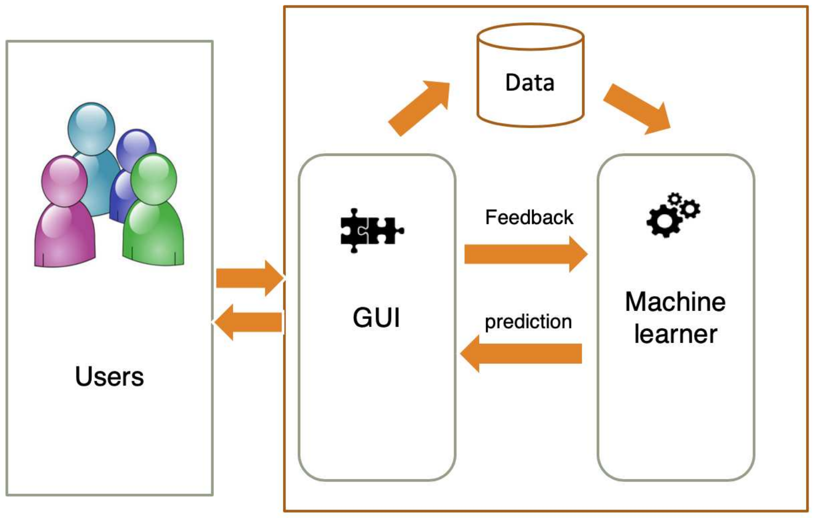


Figure 1.3: High level structure of recommender systems. In the front-end, there is a graphical user interface (GUI) that displays the prediction (e.g., ads) to users while sending information (e.g., clicks) to the machine learner in the back-end. The draw of this figure is inspired by [19].

The step of predicting the ad is performed by the predictive model trained on the users' profiles, which contains pertinent information about the users, such as demographics, geo-location, the frequency of visits, and browser type, etc.. Then, the recommendation policy is updated whether the user clicks or ignores that ad. The main problem with such PAR systems is that the users are usually ignoring the ads, which results in sparse feedback. This problem is known as click sparsity [28]. In relating to our context, this problem eventually poses an imbalanced learning problem, where the AUC is the proper measure to evaluate the learning algorithm.

Another effective application of the AUC measure is multiple instance learning (MIL). MIL is a variant of supervised learning, where labels are assigned to groups (bags) of instances rather than individual instances. In a binary classification task, the positive bags contain at least one positive instance while the negative bags hold all negative instances. The ultimate objective of MIL is to classify novel bags or instances based on a classifier trained on the labeled bags. Many problems with weakly annotated labels can be formulated as MIL (e.g., drug activity prediction [29], diagnosis of neurological diseases [30]). One successful formulation of MIL is to transform the objective function into an AUC maximization problem [31].

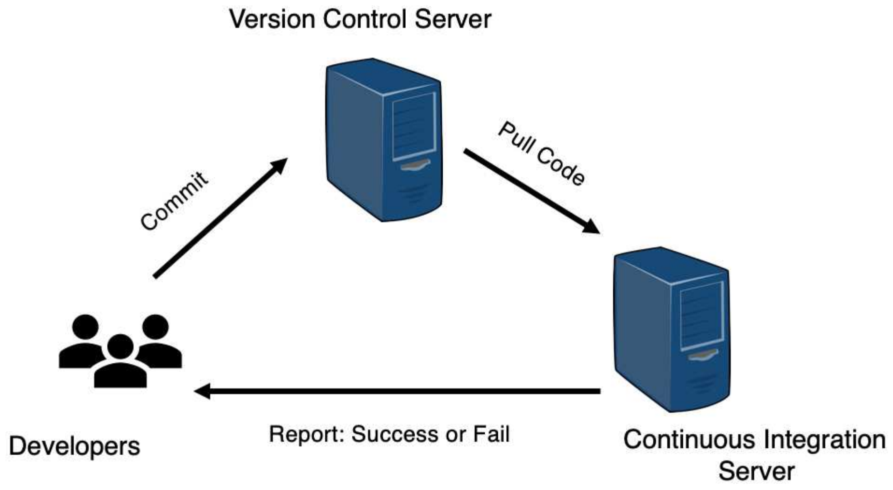


Figure 1.4: Typical Architecture of continuous integration systems

The AUC is also a proper metric for continuous integration systems. In software engineering, continuous integration is a widely used system to facilitate and expedite the software development. A paradigm architecture of continuous integration is illustrated in Figure 1.4. In this development cycle, the continuous integration executes an automated build whenever code changes are committed by a developer through the version control server. The continuous integration server then reports to the developers if the build is successful or failed.

For large systems, the build step requires impractical time, which degrades the efficiency of the continuous integration [32]. To address this issue, the outcome prediction [26, 33, 34] can be involved to forecast the success or the fail of the build, hence, reducing the time required by the build step. The outcome prediction task is similar to just-in-time defect prediction [35, 36].

1.2 Problem Statement

The area under the ROC curve (AUC) is an accurate measure to evaluate a model applied to highly imbalanced data. It is, therefore, a measure of interest for wide range of applications. The main challenge in developing a robust learning algorithm for AUC maximization is to accommodate the scalability of optimizing a pairwise loss function and the optimality of the solution. This work is devoted to devising scalable and robust AUC maximization learning algorithms to deal with large-scale imbalanced data. In particular, we propose robust algorithms for AUC maximization in different learning settings.

1.3 Summary of Contributions

To address the scalability problem of the batch kernel AUC machines, we develop nonlinear AUC maximization algorithms that use Nyström and random Fourier features to approximate a nonlinear kernel map. In an online setting, we develop linear and nonlinear second-order AUC maximization algorithms to address the suboptimality of the solutions achieved by the state-of-the-art online methods. In the stochastic regime, we devise an accelerated stochastic AUC maximization algorithm that improves the convergence rate by scheduling both the regularization and the averaging steps. We extend our accelerated AUC maximization algorithm to deal with nonlinear decision boundaries using Nyström and random Fourier approaches. We also develop a proximal AUC maximization algorithm that promotes the convergence rate of our accelerated stochastic AUC maximization algorithm.

1.4 Thesis Outline

This dissertation is organized as follows. Chapter 2 reviews the definition of the AUC measure and different formulations for the AUC maximization problem. It also reviews the mathematical derivation of Nyström and random Fourier approaches, which are popular methods for approximating kernel maps. In Chapter 3, we propose a large-scale batch nonlinear AUC machines using Nyström and random Fourier methods. We solve the batch nonlinear AUC maximization algorithm using truncated Newton optimization, which minimizes the pairwise squared hinge loss function. In Chapter 4, we develop a linear online confidence-weighted bipartite ranking algorithm. We also develop a diagonal variation of the proposed confidence-weighted bipartite ranking algorithm to deal with high-dimensional data. Chapter 5 extends the confidence-weighted bipartite ranking algorithm to address nonlinear problems. Chapter 6 proposes a linear and nonlinear accelerated stochastic AUC maximization algorithms. In Chapter 7 we develop a proximal stochastic AUC maximization algorithm. Appendix A presents the derivation for the closed-form solution of the proximal operator for the pairwise hinge loss function.

Chapter 2

Preliminaries and Background

In this section, we seek to detail the definition and formulation of the AUC maximization problem. We begin by defining the AUC metric. We then briefly describe the linear and nonlinear support vector machine formulations that directly maximize the AUC metric. We finish this chapter by reviewing Nyström low-rank kernel approximation and random Fourier features, which are widely used approaches to scale up kernel machines.

2.1 Definition of AUC metric

We borrow the definition of the AUC metric from [13, 37]. The AUC is principally the performance measure of a bipartite ranking function. To define the AUC metric, we consider the task at hand is a binary problem, which can be generalized to a multi-class problem using different techniques, such as one-vs-one or one-vs-all.

Let $\mathcal{X} \in \mathbb{R}^d$ denote the input space that contains positive and negative instances, where d is the dimension of the input space. Given a training set $\mathcal{S} = \left\{\mathcal{S}_+ \cup \mathcal{S}_- \ \in \mathcal{X}_{n^+} \times \mathcal{X}_{n^-}\right\}$, where $\mathcal{S}_+ = \{x_1, \ldots, x_{n^+}\}$ is the positive instances of size n^+, and $\mathcal{S}_- = \{x_1, \ldots, x_{n^-}\}$ is the negative instances of size n^-. The positive and negative instances are drawn i.i.d. according to $\mathcal{D}_+$ and $\mathcal{D}_-$, respectively. The goal is to learn a real-valued function $f : \mathcal{X} \rightarrow \mathbb{R}$ that scores a random positive instance higher than any negative instance. In binary classification task, the new instance is classified based on the score function along with an appropriate threshold. In bipartite ranking, the score function provides an accurate rank for a new instance.

The AUC yields a single value totalizing the performance of a classifier on average over a set of possible thresholds. For a given classifier f and positive threshold ρ, the true positive rate (TPR) of the classifier is defined as the probability of correctly classifying a random positive instance as positive.

$$\text{TPR}_f(\rho) = \text{Pr}_{x^+ \sim \mathcal{D}_+}\big[f(x^+) > \rho\big],$$

and the false positive rate (FPR) is defined as the probability of incorrectly classifying a random negative instance as positive.

$$\text{FPR}_f(\rho) = \text{Pr}_{x^- \sim \mathcal{D}_-}\big[f(x^-) > \rho\big].$$

The results of plotting $\text{TPR}_f(\rho)$ versus $\text{FPR}_f(\rho)$ for different values of the threshold ρ is the ROC curve, which is shown in Figure 2.1. The AUC is defined as [37]:

$$\text{AUC}_f = \int_0^1 \text{TPR}_f(\text{FPR}_f^{-1}(u))du,$$

where $\text{FPR}_f^{-1}(u) = \inf\{\rho \in \mathbb{R} | \text{FPR}_f(\rho) \le u\}$. Therefore, the AUC can be defined as the probability of scoring a random positive instance higher than a random negative instance [13]. This definition can be written as the following:

$$\text{AUC}_f = \text{Pr}_{(x^+,x^-)\sim(\mathcal{D}_+\mathcal{D}_-)}\big[f(x^+) > f(x^-)\big],$$

where the assumption of assigning the same score to random positive and negative instances is ignored.

The empirical ROC curve of the classifier f can be plotted by computing the TPR and FPR for all instances of the given sample S with respect to each distinct value of threshold ρ as follows:

$$\widehat{\text{TPR}}_f(\rho) = \frac{1}{n^+}\sum_{i=1}^{n^+} \text{I}(f(x_i^+) > \rho) \quad \text{and} \quad \widehat{\text{FPR}}_f(\rho) = \frac{1}{n^-}\sum_{j=1}^{n^-} \text{I}(f(x_j^-) > \rho),$$

where $\text{I}(\cdot)$ is an indicator function that returns one if its argument is true and zero otherwise. The empirical AUC can be written as follows:

$$\widehat{\text{AUC}}_f = \frac{1}{n^+n^-}\sum_{i=1}^{n^+}\sum_{j=1}^{n^-} \mathtt{I}(f(x_i^+) > f(x_j^-)) \tag{2.1}$$

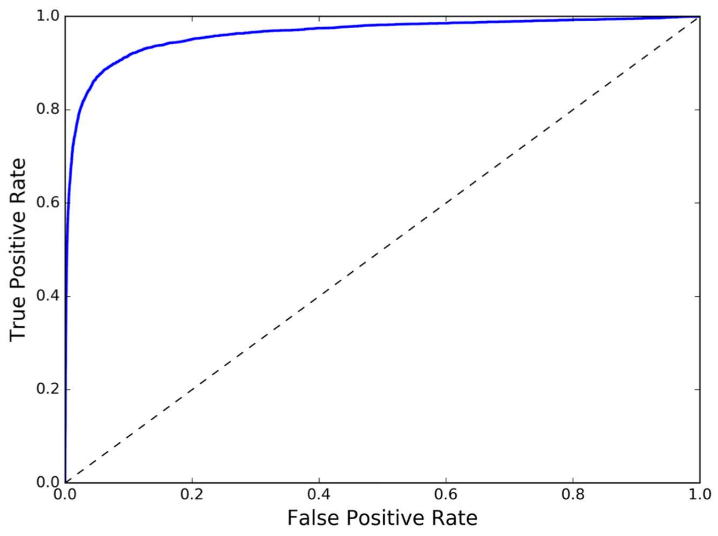


Figure 2.1: Receiver Operating Characteristic curve (ROC Curve). The ROC plots the true positive rate (TPR) versus the false positive rate (FPR). The computation of the AUC results in a single value assessing the performance of the model on average over multiple thresholds (the whole region under the curve). The lower left and the upper right corners correspond to classifying all instances as negative and positive, respectively. The ideal classifier has TPR=1 and FPR=0, which corresponds to the upper left corner.

2.2 Linear and Nonlinear AUC Maximization

The support vector machines are widely used to optimize the empirical AUC loss function. We name the SVM algorithms that optimize the AUC loss function as pairwise classifiers. The pairwise classifier builds a large margin bipartite ranking model that can be employed to rank a set of instances based on their relevance. In other words, the minimization of a pairwise SVM can be boiled down to a minimization of a binary classification problem. Given the training set S, and assuming the model is a linear classifier $f(x) = w^T x$ for some $w \in \mathbb{R}^d$, the empirical pairwise learning is defined as the minimization of the following objective function:

$$\widehat{\mathrm{R}}_{\mathrm{AUC}}(w, S) = 1 - \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} \mathrm{I}(w^T x_i^+ \leq w^T x_j^-). \tag{2.2}$$

The minimization of this objective function is equivalent to maximizing the AUC. In general, the optimization of the indicator function $I(\cdot)$ is difficult because of its discontinuous nature [37]. A common practical approach to circumvent the indicator function is to use a proxy to the loss called a surrogate loss function. Therefore, the objective function that maximizes the AUC can be written as follows:

$$\widehat{\mathrm{R}}_{\mathrm{AUC}}(w,S) = \frac{1}{n^+n^-}\sum_{i=1}^{n^+}\sum_{j=1}^{n^-}\boldsymbol{\ell}(w^Tx_i^+ - w^Tx_j^-), \tag{2.3}$$

where the loss function $\boldsymbol{\ell}(z) = max(0, 1-z)^p$. This loss function is known as a pairwise hinge loss function (L1-loss) when the exponent $p = 1$, and a pairwise quadratic hinge loss (L2-loss) when the exponent $p = 2$. Other loss functions (e.g., Huber loss) can also be used as a pairwise loss function. Both hinge and quadratic loss functions upper bound the indicator function. The study by [38] argues that the L1-loss is inconsistent with the AUC compared to the quadratic hinge loss. This means that the minimization of the pairwise hinge loss function might not lead to the minimization of the real loss function. The regularized objective function that maximizes the AUC is defined as

$$\min_w \frac{\lambda}{2}||w||^2 + \frac{1}{n^+n^-}\sum_{i=1}^{n^+}\sum_{j=1}^{n^-}\boldsymbol{\ell}(w^Tx_i^+ - w^Tx_j^-), \tag{2.4}$$

where $\lambda > 0$ is a regularization hyper-parameter that can be tuned via the cross-validation method. The norm in the first term $||w||$ is L2 norm regularization. L1 norm regularization can also be used to yield a sparse model. The kernelized variation of objective 2.4 with L1-loss can be obtained using the kernel trick [39], then solving the following constrained objective function [40]

$$\begin{aligned} &\min_{w,\,\xi} \frac{1}{2}||w||^2 + C\sum_{i,j\,\in\pi}\xi_{i,j} \\ &\text{s.t. } w^T\big(\phi(x_i^+) - \phi(x_j^-)\big) \;\geq\; 1 - \xi_{i,j}, \\ &\quad \xi_{i,j} \geq 0,\ \forall(i,j) \in \pi, \end{aligned} \tag{2.5}$$

where $C > 0$ is the regularization hyper-parameter and ϕ is a linear or nonlinear function that maps instances to higher dimensional space (feature space). π represents the set of correct pairs

$\pi \equiv \{(i,j) \mid y_i > y_j\}$, where $y \in \{-1, 1\}$. For L2-loss, the slack variable is turned into quadratic form $\xi_{i,j}^2$. This constrained objective function can be solved using any SVM solver. However, the large number of constraints makes the solution inefficient. The work by [41] reformulates the constrained objective as 1-slack structural SVM that adopts a cutting plane optimization method to solve the following problem:

$$\begin{aligned} &\min_{w,\xi} \frac{1}{2}||w||^2 + C\xi \\ &\text{s.t. } w^T \sum_{(i,j)\in\pi} c_{i,j}\phi_{i,j} \geq \sum_{(i,j)\in\pi} c_{i,j} - \xi, \\ &c_{i,j} \in \{0,1\}, \forall (i,j) \in \pi, \end{aligned} \tag{2.6}$$

where $\phi_{i,j} \equiv \phi(x_i) - \phi(x_j), \forall (i,j) \in \pi$. This problem is solved by considering a subset of the dual variables to solve in each iteration and adding the most violated constraint into a working set. In practice, the resulting solution has been shown to be sparse.

The kernelized pairwise SVM can also be formulated as an unconstrained objective function [42]

$$\min_{\beta} \frac{1}{2} \beta^T K \beta + C \sum_{A_i K\beta < 1} (1 - A_i K\beta)^2, \tag{2.7}$$

where A is a matrix of size $n^+n^- \times (n^+ + n^-)$, n^+n^- is the number of pairs, $(n^+ + n^-)$ is the sum of positive and negative instances (size of the training set), and $K = \{k(x_i, x_j)\}_{i,j=1}^{n^+ + n^-}$ is the kernel matrix. The matrix A seems very large, but it is also very sparse, which makes its computation and storage inexpensive. Each row has only two non-zero columns that indicate a correct pair. For example, row v has only two non-zero columns $A_{vi} = 1$ and $A_{vj} = -1$, which indicates this correct pair $(x_i - x_j)$. The work [42] uses non-linear conjugate gradient and truncated Newton techniques to optimize this objective function. The work [42] also proposes a faster algorithm to deal with the pairwise loss function. Both of the proposed algorithms in [42] scale linearly with the number of instances. However, the kernelized variations of the proposed algorithms in [42] become infeasible when dealing with large-scale and nonlinearly distributed data.

2.3 Nyström Low-Rank Approximation

The Nyström approximation [1, 43, 2, 44] is a popular approach to approximate the feature maps of linear and nonlinear kernels, and hence scaling up the kernel learning algorithms. The Nyström method relies on the input data points in approximating the feature maps. It was originally proposed to provide a numerical approximation for the following integral equations,

$$\int p(y)k(x,y)\phi_i(y)dy = \lambda_i\phi_i(x), \tag{2.8}$$

where $p(\cdot)$ is the probability density function of the vector y, k denotes a positive semidefinite kernel function, $\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$ are the eigenvalues, and $\phi_1, \phi_2, \cdots$ are the eigenfunctions.

Given a set of data points $X = \{x_1, x_2, \cdots, x_n\}$ drawn from the probability $p(\cdot)$, a kernel function K, and a set of landmark data points $L = \{l_1, l_2, \cdots, l_m\}$, which are sampled from X, with $m \ll n$, then for any x in X, the Nyström method approximates the integral equation by the following empirical average:

$$\frac{1}{m}\sum_{j=1}^{m} k(x, l_j)\phi_i(l_j) \simeq \lambda_i\phi_i(x). \tag{2.9}$$

Replacing x with any l_j, the equation 2.9 can be solved indirectly via eigenvalue decomposition: $KU = U\Lambda$, where $K_{ij} = k(x_i, x_j)$ for $i, j = 1, 2, \cdots, m$, $U \in \mathbb{R}^{l\times l}$ is the eigenvectors, $\Lambda \in \mathbb{R}^{l\times l}$ is the diagonal matrix of the eigenvalues. Therefore, the eigenfunctions and eigenvalues in 2.8 can be approximated as follows [1]:

$$\phi_i(x_j) \simeq \sqrt{m}U_{ji} \quad , \quad \lambda_i \simeq q^{-1}\lambda_i. \tag{2.10}$$

When considering equation 2.9, and for non-landmark point x, the j-th eigenfunction at x can be approximated as follows [44]:

$$\phi_i(x) \simeq \frac{1}{m\lambda_j} \sum_{i=1}^{m} k(x, l_i)\phi_j(l_i). \tag{2.11}$$

This means that the eigenvectors of the landmark points can be used to approximate the eigenvector for any given vector x.

This approximation method can be applied to a full kernel matrix by using the approximated eigenvectors and the eigenvalues of a set of landmark points to approximate the eigen-system of the full kernel matrix. The full kernel matrix K can be reconstructed as follows:

$$K \approx (E\, U_r\, \Sigma_r^{\frac{1}{2}})(E\, U_r\, \Sigma_r^{\frac{1}{2}})^T$$

$$K \approx E\, W^{-1}\, E^T,$$

where $W_{ij} = \kappa(l_i, l_j)$ is a kernel matrix computed on landmark points and W^{-1} is its pseudo-inverse. The matrix $E_{ij} = \kappa(x_i, l_j)$ is a kernel matrix representing the intersection between the input space and the landmark points. To derive W^{-1}, the matrix W is factorized using singular value decomposition or eigenvalue decomposition as follows: $W = U\Sigma^{-1}U^T$, where the columns of the matrix U hold the orthonormal eigenvectors while the diagonal matrix Σ holds the eigenvalues of W. Assume the eigenvalues is ordered in descending order, then $W^{-1} = \Sigma_{i=1}^{r}\sigma_i^{-\frac{1}{2}}U^iU^{i^T}$, where $r \leq$ rank(W), σ_i denotes the i-th diagonal element, and U^i is the i-th column of U [43]. The Nyström approximation can be utilized to transform the kernel machines into linear machines by embedding the input space nonlinearly in a finite-dimensional feature space. The nonlinear embedding for an instance x is defined as follows,

$$\psi(x) = U_r\, \Sigma_r^{-\frac{1}{2}}\phi^T(x),$$

where $\phi(x) = [\kappa(x, l_1), \ldots, \kappa(x, l_m)]$, the diagonal matrix Σ_r holds the top r eigenvalues, and U_r is the corresponding eigenvectors. The rank-r, $r \leq v$, is the best rank-r approximation of W. The choice of the landmark points has a crucial influence on the quality of the Nyström approximation. The optimal landmark points are hard to choose because of the combinatorial nature of the problem. Therefore, different strategies have been proposed to find a sound landmark points. The work in [45] uses uniform sampling without replacement to select the landmark points. Multiple studies [46, 2, 47, 48] propose non-uniform sampling approaches to select the landmarks points.

2.4 Random Fourier Features

Random Fourier features [4] is another approach to scale up kernel learning methods. Specifically, it is used in approximating positive-definite shift-invariant kernels. A shift-invariant kernel is any kernel $K(x, y)$ that can be written as $K(x, y) = k(z)$, where $z = (x - y)$. This is the property of the well-known kernels such as Gaussian kernel and Laplacian kernel. Unlike Nyström method, the random Fourier is data-independent, meaning it approximates the kernel function in isolation from the input data points.

The random Fourier embedding is constructed based on the classical Bochner's theorem [49], which states that a function $k : \mathbb{R}^d \to \mathbb{C}$ is positive definite if and only if it is the Fourier transform of a finite non-negative measure on $\mathbb{R}^d$. The Fourier transform of a positive definite function $k(z)$ can be written as the following integral equation,

$$p(\theta) = \frac{1}{2\pi} \int e^{-i\theta^T z} k(z) d(z),$$

where $p(\theta)$ can be expressed as a probability distribution [49] and i denotes the imaginary unit. The inverse Fourier features of $p(\theta)$ can be written in the form,

$$k(z) = \frac{1}{2\pi} \int p(\theta) e^{-i\theta^T z} d\theta.$$

According to Bochner's theorem [49], the kernel function $k(\cdot)$ can be reformulated as an expectation with a random variable θ [4, 50]:

$$k(x-y) = \frac{1}{2\pi} \int p(\theta) e^{-i\theta^T(x-y)} d\theta$$

$$k(x-y) = 2\ E_{\theta\sim p(\theta)}\ [e^{-i\theta^T(x)}\ e^{-i\theta^T(y)^*}], \tag{2.12}$$

where x^* denotes the complex conjugate of x. As the probability distribution $p(\theta)$ and the kernel are real-valued, we ignore the imaginary part of 2.12 and use cosines to replace the complex exponentials [4]. Therefore, the expectation 2.12 can be rewritten as follows,

$$k(x-y) = 2\ E_{\substack{\theta\sim p(\theta)\\ b\sim U(0,2\pi)}}\ [cos(\theta^T x + b)\ cos(\theta^T y + b)],$$

where $U(0, 2\pi)$ is the uniform distribution on $[0, 2\pi]$. For a given embedding of size D dimension, we sample D random Fourier components $\{\theta_i, b_i\}_{i=1}^D$ independently from the distribution $p(\theta)$. Therefore, the expectation above can be approximated using Monte Carlo sampling

$$k(x-y) \approx \sqrt{2/D} \sum_{j=1}^{D} cos(\theta_j^T x + b_j)\ cos(\theta_j^T y + b_j). \tag{2.13}$$

Therefore, we can approximate the original feature maps by defining the approximate mapping $\psi(x)$ for the input vector x as follows [4],

$$\psi(x) = \sqrt{2/D}[cos(\theta_1^T x + b_1)\ , \cdots, cos(\theta_D^T x + b_D)].$$

Notice that the inner products of the approximate feature maps approximate the inner products of the original ones in the reproducing kernel Hilbert space (RKHS). Therefore, a shift-invariant kernel function can be approximated as the inner products of two nonlinear approximate mapping,

$$k(x, y) \approx \psi(x)^T \psi(y).$$

For Gaussian kernel $k(x, y) = exp(-\frac{||x-y||_2^2}{2\sigma^2})$ the embedding is obtained by sampling each random Fourier component θ_i from the distribution $p(\theta) = \mathcal{N}(0, \sigma^{-2} I)$. The embedding for Laplacian kernel $k(x,y) = exp(-\frac{||x-y||_1}{\sigma})$ is obtained by sampling θ_i from the distribution $p(\theta) = \sigma \prod_d \frac{1}{\pi(1+\sigma^2\theta_d^2)}$.

The embedding based on the Fourier transform $p(\theta)$ can be expressed by another form [4]. In this variant, the expectation is defined as below,

$$k(x-y) = 2\ E_{\theta \sim p(\theta)}\ [e^{i\theta^T(x)}\ e^{-i\theta^T(y)}]$$

$$k(x-y) = 2\ E_{\theta \sim p(\theta)}\ [cos(\theta^T x)\ cos(\theta^T y) + sin(\theta^T x)\ sin(\theta^T y)]$$

$$k(x-y) = 2\ E_{\theta \sim p(\theta)}\ [[sin(\theta^T x), cos(\theta^T y)] \cdot [sin(\theta^T x), cos(\theta^T y)]].$$

Therefore, the representation for an embedding of size D dimension is obtained as follows,

$$\psi(x) = \sqrt{2/D}[sin(\theta_1^T x), cos(\theta_1^T x)\ , \cdots , sin(\theta_D^T x), cos(\theta_D^T x)].$$

Chapter 3

Scalable Batch Nonlinear AUC Maximization

3.1 Introduction

Kernelized AUC classifiers can model a complex nonlinear structure of datasets. As it is the case with the kernel methods, the kernelized AUC maximization methods use a positive semi-definite kernel function K that implicitly models a nonlinear mapping ϕ such that $K(x, y) = \langle \phi(x), \phi(y) \rangle$. This rich representation enables the kernelized AUC classifiers to learn hard nonlinear decision boundary and thus to achieve a powerful classification accuracy.

However, the kernelized AUC maximization methods inherit the curse of kernelization that hinders their scalability for training large-scale datasets. The kernel methods entail building a kernel function of size $n \times n$, where n is the number of instances in the original input space. Therefore, the complexity of training a kernelized AUC algorithm is quadratic in the number of instances $O(n^2)$, while the complexity of testing the classifier is linear in the number of support vectors, which also grows linearly with the number of training instances.

On the other hand, a linear AUC classifier can scale up for large-scale datasets compared to kernelized AUC machines. This because the complexity of training a linear AUC classifier is linear in the dimensionality of the training dataset $O(nd)$. However, the linear classifiers fail to model the complex nonlinear structure underlying most real-world datasets.

Kernel approximation methods, such as Nyström low-rank approximation [46] and random Fourier features [4], are practical solutions to speed up kernel methods. The Nyström method is a data-dependent, while random Fourier method is a data-independent method. Such methods approximate the feature maps explicitly, and hence a linear classifier can be exploited to learn on the approximate feature space. Though kernel approximation methods have been widely used to scale up standard kernel SVM, it still has not been adopted for kernel AUC maximization.

In this work, we present two scalable batch nonlinear AUC classifiers that adopt the kernel approximation methods. The first algorithms model nonlinearity by approximating the kernel matrix via the k-means Nyström approximation [2]. A batch linear AUC maximization classifier is then applied to the approximate feature space. The second algorithm approximates the kernel function using random Fourier features [4]. For the batch AUC classifier, we solve the pairwise squared hinge loss function using the truncated Newton solver [42].

3.2 Related work

The multivariate nature of the AUC loss function makes its optimization using vanilla support vector machine (SVM) infeasible. The work in [51] proposes to use a subset of the positive and negative instances based on their proximity to optimize the AUC metric using a quadratic programming solver. However, this simple approximation can yield a reduction in the generalization ability of the classifier due to neglecting some instances. The structural SVM formulation is adopted by [52, 41] to solve the AUC metric and other nonlinear measures. This algorithm optimizes the AUC loss function in the dual formulation using the cutting-plane method. Although using a sophisticated optimization problem, this method shows slow convergence [53].

Further, most ranking algorithms can be used to solve the AUC maximization problem. The large-scale kernel RankSVM is proposed in [54] to address the high complexity of learning nonlinear kernel ranking machines. Several linear RankSVM methods are presented in [42, 53, 55]. These methods implement different approaches to address the complexity of computing the pairwise loss function per iteration. To further reduce the complexity of optimizing the pairwise loss function for AUC, the work [56] approximates the real AUC using a polynomial approximation function and then uses gradient descent to optimize this approximated AUC.

However, the kernelized SVM algorithms designed to maximize the AUC metric require large memory and computation complexity, which grows quadratically with the number of instances. Therefore, they are infeasible for large-scale applications. Meanwhile, the linear SVM algorithms for AUC maximization are more efficient but cannot model the complex nonlinear structures underlying most real-world datasets.

A recent study [57] explores the Nyström approximation to speed up the training of the nonlinear kernel ranking function. This work does not address the AUC maximization problem. Another recent method [58] attempts to speed up the training of nonlinear AUC classifiers by learning a sparse model constructed incrementally based on chosen criteria [5]. However, the sparsity can deteriorate the generalization ability of the classifier.

3.3 Nonlinear AUC Maximization Methods

To address the scalability problem of kernelized AUC machines, we build our nonlinear models by learning linear AUC machines on a nonlinear space. The vantage of the linear classifier is that its complexity is linear to the dimension of the input space. The nonlinear representation is introduced by approximating the kernel representation using data-dependent or data-independent approaches. The AUC optimization problem can be solved for w in the embedded space as follows,

$$\min_{w} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} max(0, 1 - w^T(\psi(x_i^+) - \psi(x_j^-)))^2, \tag{3.1}$$

where $\psi(\cdot)$ is a nonlinear mapping. The minimization of (3.1) can be solved using truncated Newton methods [42] as shown in Algorithm 1. The matrix A in Algorithm 1 is a sparse matrix of size $r \times n$, where r is the number of pairs. It holds all possible pairs in which each row of A has only two nonzero values. That is, if $(i, j) \mid y_i > y_j$, the matrix A has a k-th row such that $A_{ki} = 1, A_{kj} = -1$. However, the complexity of this Newton batch learning is dependent on the number of pairs. Chapelle and Keerthi [42] also proposed the PSVM+ algorithm, which avoids the direct computation of pairs by reformulating the pairwise loss function as follows [42],

$$L = \sum_{k=1}^{n} \alpha_k \hat{y}_k^2 - \beta_k \hat{y}_k,$$

where $\alpha_i = |B_i|$, $i \in A$; $\alpha_j = |A_j|$, $j \in B$; $\beta_i = \sum_{j \in B_i} \hat{y}_j$, $i \in A$; $\beta_j = \sum_{i \in A_j} \hat{y}_i$, $j \in B$, the set $A = \{i : x_i^+\}$, the set $B = \{i : x_i^-\}$, and $\hat{y}$ is defined as,

$$\hat{y} = \begin{cases} w^T x_i - \frac{1}{2} & \text{if } i \in A \\ w^T x_i + \frac{1}{2} & \text{if } i \in B. \end{cases}$$

The gradient of the loss function can be computed as follows [42]:

$$g_L := \frac{\partial L}{\partial w} = X^T \frac{\partial L}{\partial y}; \text{ and } \frac{\partial L}{\partial y_k} = 2(\alpha_k \hat{y_k} - \beta_k), \forall k = 1, \cdots, n.$$

The Hessian vector multiplication is defined as follows [42],

$$H_L s = 2X^T z, \text{ where } z_k = (\alpha_k \delta_k - \gamma_k), \ \forall k = 1, \cdots, n,$$

where $\gamma_i = \sum_{j \in B_i} x_j s$, $i \in A$, $\gamma_j = \sum_{i \in A_j} x_i s$, $j \in B$, and s is a given vector. The time complexity of computing g_L is $O(n \; log \; n + nd)$, while the time complexity of each Hessian vector multiplication is $O(nd + n + n)$ [53].

3.3.1 Nyström AUC Maximization

To define the approximate feature maps in equation 3.1, we use Nyström low-rank kernel approximation [2]. In the embedding steps, we construct the nonlinear mapping (embedding) based on a given kernel function and landmark points. The landmark points are computed by the k-means clustering algorithm applied to the input space. Once the landmark points are obtained, the matrix W and its decomposition are computed. The original input space is then mapped nonlinearly to a

```
Algorithm 1: Batch Nonlinear AUC Maximization
    Input: embedded data $\tilde{X}$
    Output: the ranking model $w$
    initial vector $w \leftarrow 0$
    while stopping criterion is not satisfied do
        $D = max(0, 1 - A(w^T \tilde{X}))$
        Compute gradient $g = w - (CD^T A\tilde{X})^T$
        Compute a search direction $s_t$ by applying conjugate gradient to solve
        $\nabla^2 F(w_k)s = -\nabla F(w_k)$
        Update $w_{k+1} = w_k + s_k$
    end while
```

finite-dimensional feature space in which the nonlinear problem can be solved using linear AUC machines.

The complexity of the k-means algorithm is linear $\mathcal{O}(nd)$, while the complexity of singular value decomposition or eigenvalue decomposition is $\mathcal{O}(v^3)$. Therefore, the complexity of the k-means Nyström approximation is linear in the input space. The steps of constructing the embedding using Nytröm method are illustrated in Algorithm 2 and the mathematical explanation of Nyström approximation is illustrated in Chapter 2.

```
Algorithm 2: Nyström AUC Maximization
    Embedding Steps:
        Compute the centroid points $\{u_l\}_{l=1}^m$
        Form the matrix $W$: $W_{ij} = \kappa(u_i, u_j)$
        Compute the eigenvalue decomposition: $W = U\Sigma U^T$
        Form the matrix $E$: $E_i = \phi(x_i) = [\kappa(x_i, u_1), \ldots, \kappa(x_i, u_m)]$
        Construct the feature space: $\varphi(X) = U_r \Sigma_r^{-\frac{1}{2}} E^T$
    Training:
        Learn the batch model described in Algorithm 1
    Prediction:
        Map a test point $x$: $\varphi(x) = U_r \Sigma_r^{-\frac{1}{2}} \phi^T(x)$
        Score value: $w^T \varphi(x)$
```

3.3.2 Random Fourier AUC Maximization

We use random Fourier features [4] to define the approximate feature maps in equation 3.1. Given a shift-invariant kernel function k and the dimension m of the approximate mapping, we compute its Fourier transform p and sample m components from the distribution p. The steps of constructing the feature maps using random Fourier features are shown in Algorithm 3. As described in Chapter 2, Rahimi and Recht [4] proposed two approaches to approximate the kernel function using Fourier features. Experimentally, we did not notice a signification difference in accuracy between the two approaches. In our method, we implement the following transformation,

$$\psi(x) = \sqrt{2/m}[cos(\theta_1^T x + b_1) \ , \cdots , cos(\theta_m^T x + b_m)]. \tag{3.2}$$

```
Algorithm 3: Random Fourier AUC Maximization
    Embedding Steps:
        Compute the Fourier transform of a given kernel function $k$
        Sample the corresponding random Fourier components $\theta_1 \cdots , \theta_m$
        Sample $b_1, \cdots , b_v$ from $[0, 2\pi]$ uniformly at random
        Construct the feature space as in equation 3.2
    Training:
        Learn the batch model described in Algorithm 1
    Prediction:
        Map a test point as in equation 3.2
        Score value: $w^T \varphi(x)$
```

3.4 Experiments

We evaluate the proposed method on several benchmark datasets and compare it with the kernel AUC algorithm and other state-of-the-art online AUC maximization algorithms. The experiments are implemented in MATLAB, while the learning algorithms are written in C++ language via MEX

Table 3.1: Description of the data sets

Data	#training	#test	#feat	ratio
spambase	3,680	921	57	1.53
usps	7,291	2,007	256	1.40
magic04	15,216	3,804	10	1.84
protein	17,766	6,621	357	2.11
ijcnn1	49,990	91,701	22	9.44
connect-4	54,045	13,512	126	3.06
acoustic	78,823	19,705	50	3.31
skin	196,045	49,012	3	3.83
cod-rna	331,152	157,413	8	2.0
covtype	464,809	116,203	54	10.65

files. The experiments were performed on a computer equipped with an Intel 4GHz processor with 32G RAM.

3.4.1 Benchmark Datasets

The datasets we use in our experiments can be downloaded from LibSVM website[1] or UCI[2]. The datasets (i.e., spambase, magic04, connect-4, skin, and covtype) that are not split into training and test sets are divided into 80%-20% for training and testing. The multi-class datasets are converted into class-imbalanced binary data by grouping the instances into two sets, where each set has roughly the same number of class labels. To speed up the experiments that include the kernelized AUC algorithm, we train all the compared methods on 80k instances, randomly selected from the training set. The features of each dataset are standardized to have zero mean and unit variance. The characteristics of the datasets along with their imbalance ratios are shown in Table 3.1.

3.4.2 Compared Methods and Model Selection

We compare the proposed method with kernel RankSVM and linear RankSVM, which can be used to solve the AUC maximization problem. The random Fourier method that approximates the

[1]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[2]http://archive.ics.uci.edu/ml/index.php

kernel function is also involved in the experiments where the resulting classifier is solved by linear RankSVM.

1. **RBF-RankSVM:** This is the nonlinear kernel RankSVM [54]. We use Gaussian kernel $K(x, y) = exp(-\gamma||x-y||^2)$ to model the nonlinearity of the data. The best width of the kernel γ is chosen by 3-fold cross validation on the training set via searching in $\{2^{-6}, \ldots, 2^{-1}\}$. The regularization hyper-parameter C is also tuned by 3-fold cross validation by searching in the grid $\{2^{-5}, \ldots, 2^{5}\}$. The searching grids are selected based on [54]. We also train the RBF-RankSVM on $1/5$ subsamples, selected randomly.

2. **Linear RankSVM (PRSVM+):** This is the linear RankSVM that optimizes the squared hinge loss function using truncated Newton [42]. The best regularization hyper-parameter C is chosen from the grid $\{2^{-15}, \ldots, 2^{10}\}$ via 3-fold cross validation.

3. **FBAM:** This is the proposed batch AUC maximization trained on the approximate feature space constructed by random Fourier features [4]. We use PRSVM+ to solve the AUC maximization problem on the projected space. The hyperparameters C and γ are selected via 3-fold cross validation by searching on the grids $\{2^{-15}, \ldots, 2^{10}\}$ and $\{1, 10, 100\}$, respectively.

4. **NBAM:** This is the proposed batch AUC maximization algorithm trained on the embedded space. We solve it using the PRSVM+ algorithm [42]. The hyper-parameter C is tuned similarly to the linear RankSVM.

For our algorithm that involves the k-means Nyström approximation, we compute 1600 landmark points using the k-means clustering algorithm, which is implemented in C++ language. We select a Gaussian kernel function to be used with the k-means Nyström approximation. The bandwidth of the Gaussian function is set to be the average square distance between the first 80k instances and the mean, which is computed over these instances. For a fair comparison, we also set the number of random Fourier features to 1600.

3.4.3 Results and Discussion

The comparison of batch AUC maximization methods in terms of AUC classification accuracy on the test set is shown in Table 3.2, while Table 3.3 compares these batch methods in terms of training time. For connect-4 dataset, the results of RBF-RankSVM are not reported because the training runs over five days.

We observe that the proposed NBAM outperforms the competing batch methods in terms of AUC classification accuracy. The AUC performance of RBF-RankSVM might be improved for some datasets if the best hyperparameters are selected on a more restricted grid of values. Nevertheless, the training of NBAM is several orders of magnitude faster than RBF-RankSVM. The fast training of NBAM is demonstrated on the large datasets.

The proposed NBAM shows a robust AUC performance compared to FBAM on most datasets. This can be attributed to the robust capability of the k-means Nyström method in approximating complex nonlinear structures. It also indicates that a better generalization can be attained by capitalizing on the data to construct the feature maps, which is the main characteristic of the Nyström approximation, while the random Fourier features are oblivious to the data.

We also observe that the AUC performance of both RBF-RankSVM and its variant applied to random subsamples outperform the linear RankSVM, except for the protein dataset. However, RBF-RankSVM methods require longer training time, especially for large datasets. We see that the linear RankSVM performs better than the kernel AUC machines on the protein dataset. This implies that the protein dataset is linearly separable. However, the AUC performance of the proposed method NBAM is even better than linear RankSVM on this dataset.

The optimization of PRSVM+ to maximize the AUC metric still requires $O(n\hat{d} + 2n + \hat{d})$ operations to compute each of the gradient and the Hessian-vector product in each iteration, where $\hat{d}$ is the dimension of the embedded space. This makes the training of PRSVM+ expensive for massive datasets embedded using a large number of landmark points. A large set of landmark points is desirable to improve the approximation of the feature maps; hence boosting the generalization ability of the involved classifier. In the next section, we attempt to address the scalability of AUC

Table 3.2: Comparison of AUC performance $(\%)$ for different batch classifiers

Data	RBF-RankSVM	RBF-RankSVM(subsample)	Linear RankSVM	FBAM	**NBAM**
spambase	98.00	96.02	97.47	97.75	98.04
usps	99.08	98.54	90.27	97.42	99.24
magic04	92.18	91.34	84.47	92.83	93.06
protein	80.97	77.60	83.30	58.43	84.33
ijcnn1	99.68	99.35	91.56	98.86	99.57
connect-4	-	91.32	88.20	91.10	94.09
acoustic	93.60	93.02	87.38	91.82	94.14
skin	99.92	99.92	94.81	100	99.98
cod-rna	99.07	99.07	98.85	99.12	99.12
covtype	93.94	94.05	87.75	95.99	96.03

Table 3.3: Comparison of training time (in seconds) for different batch classifiers

Data	RBF-RankSVM	RBF-RankSVM(subsample)	Linear RankSVM	**FBAM**	**NBAM**
spambase	3.08	0.10	0.13	3.59	7.71
usps	492.30	0.83	1.42	6.77	27.68
magic04	518.04	3.71	0.08	21.51	25.46
protein	2614.7	4.81	4.47	14.20	73.81
ijcnn1	15,434	282	0.57	80.17	88.87
connect-4	-	12,701	3.42	62.60	164.48
acoustic	134,030	5,610	1.88	92.74	151.78
skin	2037.30	78.20	0.20	73.18	23.71
cod-rna	5,715	255.4	0.44	83.01	113.66
covtype	133,270	11,670	2.54	273.67	220.90

maximization by developing online (one-pass) AUC maximization algorithms that can scale up to massively large datasets.

3.5 Conclusion

In this chapter, we have proposed two scalable kernelized AUC maximization methods. We use both Nyström and random Fourier features methods to speed up the kernel AUC maximization. The proposed methods can scale well compared to standard kernel AUC maximization while achieving AUC classification accuracy on par with the kernel AUC maximization method. The proposed Nyström AUC algorithm has shown a robust AUC classification performance compared to the Fourier AUC algorithm.

However, these two proposed methods are operating in a batch setting, meaning their training complexities depend on the number of training instances for each iteration in the learning algorithms. Therefore, these methods are suitable for mid-size datasets. Designing a scalable AUC maximization algorithm that can scale well with large-scale datasets is important future work.

Chapter 4

Second-Order Online AUC Maximization

4.1 Introduction

Online learning is an appealing learning paradigm for large-scale datasets. The merit of online learning is the ability to learn from a sequence or large batches of data points by carrying out a single pass over them. Unlike batch algorithms, online algorithms can update the model for any new single instance or a bunch of instances without a need to retrain the model from scratch. This property makes online algorithms desirable for large-scale applications.

Specifically, online learning is carried out by making a single pass over the training data. In each iteration, the algorithm receives a new instance and is required to predict its class label. Then the true class label is revealed, and the learner suffers a loss, which can be used to measure the quality of the learner. The loss is also involved in deciding whether to update the model or not.

Among many online learning algorithms, confidence-weighted has shown to be very effective in improving the classification performance [59]. Confidence-weighted (CW) learning [60, 61, 59] takes the advantage of the underlying structure between features by modeling the classifier as a Gaussian distribution parameterized by a mean vector and covariance matrix [61].

The confidence-weighted model captures the notion of confidence for each weight coordinate via the covariance matrix. A large diagonal value corresponding to the i-th feature in the covariance matrix results in less confidence in its weight (i.e., its mean). Therefore, an aggressive update is performed on the less confident weight coordinates. This adaptive approach is analogous to the adaptive subgradient method [62] that involves the geometric structure of the data seen so far in regularizing the weights of sparse features (i.e., less occurring features) as they are deemed more informative than dense features.

The confidence-weighted algorithm [61] has also been improved by introducing the adaptive regularization (AROW) that deals with inseparable data [63]. The soft confidence-weighted (SCW) algorithm improves upon AROW by maintaining an adaptive margin [59].

In this work, we present scalable and robust online AUC maximization algorithms. We develop a linear online soft confidence-weighted bipartite ranking algorithm that maximizes the AUC metric via optimizing a pairwise loss function. The complexity of the pairwise loss function is mitigated in our algorithm by employing a finite buffer that is updated using one of the stream oblivious policies (i.e., reservoir sampling and first-in-first-out). We also develop a diagonal variation of our confidence-weighted bipartite ranking algorithm to deal with high-dimensional data by maintaining only the diagonal elements of the covariance matrix instead of the full covariance matrix.

4.2 Related work

Multiple online learning algorithms are developed to optimize the AUC objective function. The work [64] optimizes the AUC indirectly by learning a weighted univariate loss function. Although this method has linear time and space complexities, it neglects the pairwise structure of the AUC metric. Therefore, it is prone to a suboptimal convergence to the local surrogate minimum. To optimize the surrogate AUC pairwise loss function in an online setting, the work in [65] proposes to use buffers of fixed size for storing positive and negative instances. The buffers are updated using the reservoir sampling technique to allow them to store an accurate representation of all received training instances.

The work by [66] proposes a one-pass AUC maximization algorithm (OPAUC) that optimizes the pairwise least square loss function. This algorithm is capable of eliminating the need for buffers by storing the first-order (mean) and second-order (covariance) of each received instance. The OPAUC algorithm shows improvement in the convergence rate over the first-order method [65] due to the incorporation of second-order information. To further enhance the convergence of OPAUC, the work [67] makes use of the geometric information [62] of received instances in updating the weight vector (i.e., mean). That is, sparse features will be penalized higher than the dense ones.

The work by [68] formulates the AUC maximization problem as a convex-concave saddle point problem. The proposed algorithm in [68] solves a pairwise squared hinge loss function without the need to access the buffered instances or the second-order information. Therefore, it shows linear space and time complexities per iteration with respect to the number of features.

A recent study [69] suggests optimizing the real AUC metric instead of its surrogate to achieve fast convergence. The authors of [69] attempt to optimize the real AUC loss function using a nonparametric learning algorithm. However, learning the nonparametric algorithm on high dimensional datasets is not reliable.

4.3 Confidence-Weighted Bipartite Ranking

We consider a linear online bipartite ranking function that learns from a sequence of imbalanced dataset. Let $\mathcal{S} = \{x_i^+ \cup x_j^- \ \in \mathbb{R}^d | i = \{1, \dots, n^+\}, j = \{1, \dots, n^-\}\}$ denotes the input space of dimension d generated from unknown distribution $\mathcal{D}$, where x_i^+ is the i-th positive instance and x_j^- is the j-th negative instance. The n^+ and n^- denote the number of positive and negative instances received thus far. The linear bipartite ranking function $f : \mathbb{R}^d \to \mathbb{R}$ is a real valued function that maximizes the AUC metric by minimizing the following loss function:

$$\mathcal{L}(f; \mathcal{S}) = \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} I(f(x_i^+) \leq f(x_j^-)),$$

where $f(x) = w^T x$ and $I(\cdot)$ is an indicator function that outputs 1 if the condition is held, and 0 otherwise. It is common to replace the indicator function with a convex surrogate function,

$$\mathcal{L}(f; \mathcal{S}) = \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} \ell(f(x_i^+) - f(x_j^-)), \tag{4.1}$$

where $\ell(\cdot)$ is a surrogate loss function such as hinge loss $\ell(z) = max(0, 1 - z)$.

It is easy to see that the complexity of optimizing (4.1) will grow quadratically with respect to the number of training instances. Following the approach suggested by [65] to deal with the complexity of the pairwise loss function, we reformulate the pairwise loss function (4.1) as a sum of two losses for a pair of instances,

$$\sum_{t=1}^{T} I_{(y_t=+1)} g_t^+(w) + I_{(y_t=-1)} g_t^-(w), \tag{4.2}$$

where $T = n^+ + n^-$, and $g_t(w)$ is defined as follows

$$g_t^+(w) = \sum_{t'=1}^{t-1} I_{(y_{t'}=-1)} \ell(f(x_t) - f(x_{t'})), \tag{4.3}$$

$$g_t^-(w) = \sum_{t'=1}^{t-1} I_{(y_{t'}=+1)} \ell(f(x_{t'}) - f(x_t)). \tag{4.4}$$

Instead of maintaining all the received instances to compute the gradients $\nabla g_t(w)$, we store random instances from each class in the corresponding buffer. Therefore, two buffers B_+ and B_- with predefined capacity are maintained for positive and negative classes, respectively. The buffers are updated using a stream oblivious policy. The current stored instances in a buffer are used to update the classifier as in equation (4.2) whenever a new instance from the opposite class label is received.

The framework of the online confidence-weighted bipartite ranking is shown in Algorithm 4. The two main components of this framework are UpdateBuffer and UpdateRanker, which are explained below.

Algorithm 4: Framework for Confidence-Weighted Bipartite Ranking (CBR)

```
Input:
    • the penalty parameter $C$
    • the capacity of the buffers $M_+$ and $M_-$
    • $\eta$ parameter
    • $a_i = 1$ for $i \in 1, \dots, d$
Initialize: $\mu_1 = \{0, \dots, 0\}^d$, $B_+ = B_- = \emptyset$, $M_+^1 = M_-^1 = 0$
            $\Sigma_1 = diag(a)$ or $G_1 = a$
for $t = 1, \dots, T$ do
    Receive a training instance $(x_t, y_t)$
    if $y_t = +1$ then
        $B_-^{t+1} = B_-^t$, $M_+^{t+1} = M_+^t + 1$, $M_-^{t+1} = M_-^t$
        $C_t = C$
        $B_+^{t+1}$ = UpdateBuffer($x_t, B_+^t, M_+$,$M_+^{t+1}$)
        $[\mu_{t+1}, \Sigma_{t+1}]$ = UpdateRanker($\mu_t, \Sigma_t, x_t, y_t, C_t, B_-^{t+1}, \eta$) or
        $[\mu_{t+1}, G_{t+1}]$ = UpdateRanker($\mu_t, G_t, x_t, y_t, C_t, B_-^{t+1}, \eta$) (CBR-diag)
    else
        $B_+^{t+1} = B_+^t$,$M_-^{t+1} = M_-^t + 1$, $M_+^{t+1} = M_+^t$
        $C_t = C$
        $B_-^{t+1}$ = UpdateBuffer($x_t, B_-^t, M_-$,$M_-^{t+1}$)
        $[\mu_{t+1}, \Sigma_{t+1}]$ = UpdateRanker($\mu_t, \Sigma_t, x_t, y_t, C_t, B_+^{t+1}, \eta$) or
        $[\mu_{t+1}, G_{t+1}]$ = UpdateRanker($\mu_t, G_t, x_t, y_t, C_t, B_+^{t+1}, \eta$) (CBR-diag)
    end if
end for
```

Update Buffer

One effective approach to deal with pairwise learning algorithms is to maintain a buffer with a fixed capacity. This raises the problem of updating the buffer to store the most informative instances. In our online Bipartite ranking framework, we investigate the following two stream oblivious policies to update the buffer:

Reservoir Sampling (**RS**): Reservoir Sampling is a common oblivious policy to deal with streaming data [70]. In this approach, the new instance (x_t, y_t) is added to the corresponding buffer if its capacity is not reached, $|B_{y_t}^t| < M_{y_t}$. If the buffer is at capacity, it will be updated with

Algorithm 5: Reservoir Sampling Approach

```
Input: $x_t$, $B^t$, $M$, $M_{t+1}$
Output: updated buffer $B^{t+1}$
if $|B^t| < M$ then
    $B^{t+1} = B^t \cup \{x_t\}$
else
    Sample $Z$ from a Bernoulli distribution with $Pr(Z = 1) = M/M_{t+1}$
    if $Z = 1$ then
        Randomly delete an instance from $B^t$
        $B^{t+1} = B^t \cup \{x_t\}$
    end if
end if
Return $B^{t+1}$
```

probability $\frac{M_{y_t}}{M_{yt}^{t+1}}$ by randomly replacing one instance in $B_{y_t}^t$ with x_t. Algorithm 5 shows the steps of the Reservoir sampling approach for updating the buffers.

First-In-First-Out **(FIFO)**: This simple strategy replaces the oldest instance with the new instance if the corresponding buffer reaches its capacity. Otherwise, the new instance is simply added to the buffer.

Update Ranker

Inspired by the robust performance of second-order learning algorithms, we apply the soft confidence-weighted learning approach [59] to updated the bipartite ranking function. Therefore, our confidence-weighted bipartite ranking model (CBR) is formulated as a ranker with a Gaussian distribution parameterized by mean vector $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. The mean vector μ represents the model of the bipartite ranking function, while the covariance matrix captures the confidence in the model. The ranker is more confident about the model value μ_p as its diagonal value $\Sigma_{p,p}$ is smaller. The model distribution is updated once the new instance is received while being close to the old model distribution. This optimization problem is performed by minimizing the Kullback-Leibler divergence between the new and the old distributions of the model. The online confidence-weighted bipartite ranking (CBR) is formulated as follows:

$$(\mu_{t+1}, \Sigma_{t+1}) = \underset{\mu,\Sigma}{\operatorname{argmin}} D_{KL}(\mathcal{N}(\mu,\Sigma)||\mathcal{N}(\mu_t,\Sigma_t)) \\ + C\ell^{\phi}(\mathcal{N}(\mu,\Sigma);(z,y_t)), \tag{4.5}$$

where $z = (x_t - x)$, C is the the penalty hyperparamter, $\phi = \Phi^{-1}(\eta)$, and Φ is the normal cumulative distribution function. The loss function $\ell^{\phi}(\cdot)$ is defined as:

$$\ell^{\phi}(\mathcal{N}(\mu,\Sigma);(z,y_t)) = max(0, \phi\sqrt{z^T\Sigma z} - y_t\mu\cdot z).$$

The solution of 4.5 is given by the following proposition.

Proposition 1. *The optimization problem 4.5 has a closed-form solution as follows:*

$$\mu_{t+1} = \mu_t + \alpha_t y_t \Sigma_t z,$$

$$\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t z^T z \Sigma_t.$$

The coefficients α and β are defined as follows:

$\alpha_t = min\{C, max\{0, \frac{1}{\upsilon_t\zeta}(-m_t\psi + \sqrt{m_t^2\frac{\phi^4}{4} + \upsilon_t\phi^2\zeta})\}\}$,

$\beta_t = \frac{\alpha_t\phi}{\sqrt{u_t}+\upsilon_t\alpha_t\phi}$. *where* $u_t = \frac{1}{4}(-\alpha_t\upsilon_t\phi + \sqrt{\alpha_t^2\upsilon_t^2\phi^2 + 4\upsilon_t})^2$,
$\upsilon_t = z^T\Sigma_t z$, $m_t = y_t(\mu_t \cdot z)$, $\phi = \Phi^{-1}(\eta)$, $\psi = 1 + \frac{\phi^2}{2}$, $\zeta = 1 + \phi^2$, *and*
$z = x_t - x$.

The proposition 1 is analogous to the one derived in [59].

4.4 Diagonal Confidence-Weighted Bipartite Ranking

Though modeling the full covariance matrix lends the CW algorithms a powerful capability in learning [71, 72, 59], it raises potential concerns with high-dimensional data. The covariance matrix grows quadratically with respect to the data dimension. This makes the CBR algorithm impractical with high-dimensional data due to high computational and memory requirements.

We remedy this deficiency by a diagonalization technique [71, 62]. Therefore, we present a diagonal confidence-weighted bipartite ranking (CBR-diag) that models the ranker as a mean vector $\mu \in \mathbb{R}^d$ and diagonal matrix $\hat{\Sigma} \in \mathbb{R}^{d\times d}$. Let G denotes $diag(\hat{\Sigma})$, and the optimization problem of CBR-diag is formulated as follows:

$$\begin{aligned}(\mu_{t+1}, G_{t+1}) = \underset{\mu,G}{\text{argmin}} D_{KL}(\mathcal{N}(\mu, G)||\mathcal{N}(\mu_t, G_t)) \\ +C\ell^{\phi}(\mathcal{N}(\mu, G); (z, y_t)).\end{aligned} \tag{4.6}$$

Proposition 2. *The optimization problem (4.6) has a closed-form solution as follows:*

$$\mu_{t+1} = \mu_t + \frac{\alpha_t y_t z}{G_t},$$

$$G_{t+1} = G_t + \beta_t z^2.$$

The coefficients α and β are defined as follows

$\alpha_t = min\{C, max\{0, \frac{1}{\upsilon_t\zeta}(-m_t\psi + \sqrt{m_t^2\frac{\phi^4}{4} + \upsilon_t\phi^2\zeta})\}\},$

$\beta_t = \frac{\alpha_t\phi}{\sqrt{u_t}+\upsilon_t\alpha_t\phi},$

where $u_t = \frac{1}{4}(-\alpha_t\upsilon_t\phi + \sqrt{\alpha_t^2\upsilon_t^2\phi^2 + 4\upsilon_t})^2$, $\upsilon_t = \sum_{i=1}^d \frac{z_i^2}{G_i+C}$, $m_t = y_t(\mu_t \cdot z)$,

$\phi = \Phi^{-1}(\eta)$, $\psi = 1 + \frac{\phi^2}{2}$, $\zeta = 1 + \phi^2$, *and* $z = x_t - x$.

The propositions 1 and 2 can be proved similarly to the proof in [59]. The steps of updating the online confidence-weighted bipartite ranking with full covariance matrix or with the diagonal elements are summarized in Algorithm 6.

```
Algorithm 6: Update Ranker
Input:
    • $\mu_t$ : current mean vector
    • $\Sigma_t$ or $G_t$ : current covariance matrix or diagonal elements
    • $(x_t, y_t)$ : a training instance
    • $B$ : the buffer storing instances from the opposite class label
    • $C_t$ : weighting parameter
    • $\eta$ : the predefined probability
Output: updated ranker:
    • $\mu_{t+1}$
    • $\Sigma_{t+1}$ or $G_{t+1}$
Initialize: $\mu^1 = \mu_t$, ($\Sigma^1 = \Sigma_t$ or $G^1 = G_t$), $i = 1$
for $x \in B$ do
    Update the ranker $(\mu^i, \Sigma^i)$ with $z = x_t - x$ and $y_t$ by
    $(\mu^{i+1}, \Sigma^{i+1}) = \underset{\mu,\Sigma}{\text{argmin}} D_{KL}(\mathcal{N}(\mu, \Sigma)||\mathcal{N}(\mu^i, \Sigma^i)) + C\ell^{\phi}(\mathcal{N}(\mu, \Sigma); (z, y_t))$
    or
    Update the ranker $(\mu^i, G^i)$ with $z = x_t - x$ and $y_t$ by
    $(\mu^{i+1}, G^{i+1}) = \underset{\mu,G}{\text{argmin}} D_{KL}(\mathcal{N}(\mu, G)||\mathcal{N}(\mu^i, G^i)) + C\ell^{\phi}(\mathcal{N}(\mu, G); (z, y_t))$
    $i = i + 1$
end for
Return $\mu_{t+1} = \mu^{|B|+1}$
       $\Sigma_{t+1} = \Sigma^{|B|+1}$ or $G_{t+1} = G^{|B|+1}$
```

4.5 Experiments

In this section, we conduct extensive experiments on several real world datasets in order to demonstrate the effectiveness of the proposed algorithms. We also compare the performance of our methods with existing online learning algorithms in terms of AUC and classification accuracy at the optimal operating point of the ROC curve (OPTROC). The running time comparison is also presented. The experiments are implemented in MATLAB and performed on a computer equipped with an Intel 4GHz processor with 32G RAM.

4.5.1 Real World Datasets

We conduct extensive experiments on various benchmark and high-dimensional datasets. All datasets can be downloaded from LibSVM[3] and the machine learning repository UCI[4] except the Reuters[5] dataset that is used in [73]. If the data are provided as training and test sets, we combine them together in one set. For cod-rna data, only the training and validation sets are grouped together. For rcv1 and news20, we only use their training sets in our experiments. The multi-class datasets are transformed randomly into class-imbalanced binary datasets. For glass, vehicle, and svmguide4 datasets, we transform the classification task to distinguish between class label one from other classes. For segment dataset, we transform it into an imbalanced binary classification task by partitioning its classes equally into two groups. For covtype dataset, the classification task is transformed to distinguish the class label seven from the rest of classes. For news20 and Reuters, we distinguish between classes whose labels are less than five and the rest of classes. For sector dataset, we distinguish between classes whose labels are less than or equal ten and the rest of classes. For rcv1 dataset, we transform the classification task to separate labels less than four from the rest. We randomly choose 8k instances if the data exceeds this size. For high-dimensional datasets, we experiment only on 2k random instances from each dataset because of the high dimen-

[3]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/

[4]https://archive.ics.uci.edu/ml/

[5]http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html

sionality. Tables 4.1 and 4.2 show the characteristics of the benchmark and the high-dimensional datasets, respectively.

Table 4.1: Benchmark data sets

Data	#inst	#feat
glass	214	10
heart	270	13
ionosphere	351	34
svmguide4	612	10
australian	690	14
diabetes	768	8
vehicle	846	18
german	1,000	24
svmguide3	1,284	21
segment	2,310	19
spambase	4,601	57
magic04	19,020	11
acoustic	78,823	50
cod-rna	331,152	8
covtype	581,012	54

Table 4.2: High-dimensional data sets

Data	#inst	#feat
farm-ads	4,143	54,877
Reuters	8,293	18,933
sector	9,619	55,197
rcv1	15,564	47,236
news20	15,937	62,061
real-sim	72,309	20,958

4.5.2 Compared Methods and Model Selection

1. **Online Uni-Exp** [64]: An online pointwise ranking algorithm that optimizes the weighted univariate exponential loss. The learning rate is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

2. **OPAUC** [66]: An online learning algorithm that optimizes the AUC in one-pass through square loss function. The learning rate is tuned by 3-fold cross validation by searching in $2^{[-10:10]}$, and the regularization hyperparameter is set to a small value 0.0001.

3. **OPAUCr** [66]: A variation of OPAUC that approximates the covariance matrices using low-rank matrices. The model selection step is carried out similarly to OPAUC, while the value of rank τ is set to 50 as suggested in [66].

4. **OAM$_{\text{seq}}$** [65]: The online AUC maximization (OAM) is the state-of-the-art first-order learning method. We implement the algorithm with the Reservoir Sampling as a buffer updating scheme. The size of the positive and negative buffers is fixed at 50. The penalty hyperparameter C is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

5. **AdaOAM** [67]: This is a second-order AUC maximization method that adapts the classifier to the importance of features. The smooth hyperparameter δ is set to 0.5, and the regularization hyperparameter is set to 0.0001. The learning rate is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

6. **CBR$_{\text{RS}}$** and **CBR$_{\text{FIFO}}$**: The proposed confidence-weighted bipartite ranking algorithms with the Reservoir Sampling and First-In-First-Out buffer updating policies, respectively. The size of the positive and negative buffers is fixed at 50. The hyperparameter η is set to 0.7, and the penalty hyperparameter C is tuned by 3-fold cross validation by searching in $2^{[-10:10]}$.

7. **CBR-diag$_{\text{FIFO}}$**: The proposed diagonal variation of confidence-weighted bipartite ranking that uses the First-In-First-Out policy to update the buffer. The buffers are set to 50, and the hyperparameters are tuned similarly to CBR$_{\text{FIFO}}$.

Table 4.3: AUC performance on the benchmark data sets

Data	CBR$_{\text{RS}}$	CBR$_{\text{FIFO}}$	Online Uni-Exp	OPAUC	OAM$_{\text{seq}}$	AdaOAM
glass	**0.825** $\pm$ 0.043	0.823 $\pm$ 0.046	0.714 $\pm$ 0.075	0.798 $\pm$ 0.061	0.805 $\pm$ 0.047	0.794 $\pm$ 0.061
ionosphere	0.950 $\pm$ 0.027	**0.951** $\pm$ 0.028	0.913 $\pm$ 0.018	0.943 $\pm$ 0.026	0.946 $\pm$ 0.025	0.943 $\pm$ 0.029
german	**0.782** $\pm$ 0.024	0.780 $\pm$ 0.019	0.702 $\pm$ 0.032	0.736 $\pm$ 0.034	0.731 $\pm$ 0.028	0.770 $\pm$ 0.024
svmguide4	0.969 $\pm$ 0.013	**0.974** $\pm$ 0.013	0.609 $\pm$ 0.096	0.733 $\pm$ 0.056	0.771 $\pm$ 0.063	0.761 $\pm$ 0.053
svmguide3	0.755 $\pm$ 0.022	**0.764** $\pm$ 0.036	0.701 $\pm$ 0.025	0.737 $\pm$ 0.029	0.705 $\pm$ 0.033	0.738 $\pm$ 0.033
cod-rna	0.983 $\pm$ 0.000	**0.984** $\pm$ 0.000	0.928 $\pm$ 0.000	0.927 $\pm$ 0.001	0.951 $\pm$ 0.025	0.927 $\pm$ 0.000
spambase	0.941 $\pm$ 0.006	**0.942** $\pm$ 0.006	0.866 $\pm$ 0.016	0.849 $\pm$ 0.020	0.897 $\pm$ 0.043	0.862 $\pm$ 0.011
covtype	0.816 $\pm$ 0.003	**0.835** $\pm$ 0.001	0.705 $\pm$ 0.033	0.711 $\pm$ 0.041	0.737 $\pm$ 0.023	0.770 $\pm$ 0.010
magic04	0.799 $\pm$ 0.006	**0.801** $\pm$ 0.006	0.759 $\pm$ 0.006	0.748 $\pm$ 0.033	0.757 $\pm$ 0.015	0.773 $\pm$ 0.006
heart	0.908 $\pm$ 0.019	**0.909** $\pm$ 0.021	0.733 $\pm$ 0.039	0.788 $\pm$ 0.054	0.806 $\pm$ 0.059	0.799 $\pm$ 0.079
australian	0.883 $\pm$ 0.028	**0.889** $\pm$ 0.019	0.710 $\pm$ 0.130	0.735 $\pm$ 0.138	0.765 $\pm$ 0.107	0.801 $\pm$ 0.037
diabetes	0.700 $\pm$ 0.021	**0.707** $\pm$ 0.033	0.633 $\pm$ 0.036	0.667 $\pm$ 0.041	0.648 $\pm$ 0.040	0.675 $\pm$ 0.034
acoustic	0. 879 $\pm$ 0.006	**0.892** $\pm$ 0.003	0.876 $\pm$ 0.003	0.878 $\pm$ 0.003	0.863 $\pm$ 0.011	0.882 $\pm$ 0.003
vehicle	**0.846** $\pm$ 0.031	**0.846** $\pm$ 0.034	0.711 $\pm$ 0.053	0.764 $\pm$ 0.073	0.761 $\pm$ 0.078	0.792 $\pm$ 0.049
segment	0.900 $\pm$ 0.013	**0.903** $\pm$ 0.008	0.689 $\pm$ 0.061	0.828 $\pm$ 0.024	0.812 $\pm$ 0.035	0.855 $\pm$ 0.008

For a fair comparison, the datasets are scaled similarly in all experiments. We randomly divide each dataset into 5 folds, where 4 folds are used for training and one fold is used as a test set. The results on the benchmark and the high-dimensional datasets are averaged over 10 and 5 runs, respectively. A random permutation is performed on the datasets with each run. All experiments are conducted with Matlab 15 on a workstation computer with 8x2.6G CPU and 32 GB memory.

4.5.3 Results on Benchmark Datasets

The comparison in terms of AUC is shown in Table 4.3, while the comparison in terms of classification accuracy at OPTROC is shown in Table 4.4. The running time (in milliseconds) comparison is illustrated in Figure 4.1.

The results show the robust performance of the proposed methods CBR$_{RS}$ and CBR$_{FIFO}$ in terms of AUC and classification accuracy compared to other first and second-order online learning algorithms. We can observe that the improvement of the second-order methods such as OPAUC and AdaOAM over first-order method OAM$_{\text{seq}}$ is not reliable, while our CBR algorithms often outperform the OAM$_{\text{seq}}$. Also, the proposed methods are faster than OAM$_{\text{seq}}$, while they incur more running time compared to AdaOAM except with spambase, covtype, and acoustic datasets. The pointwise method online Uni-Exp maintains fastest running time, but at the expense of the AUC and classification accuracy. We also notice that the performance of CBR$_{\text{FIFO}}$ is often slightly better than CBR$_{\text{RS}}$ in terms of AUC, classification accuracy, and running time.

Table 4.4: Comparison of classification accuracy at OPTROC on the benchmark data sets

Data	CBR$_{RS}$	CBR$_{FIFO}$	Online Uni-Exp	OPAUC	OAM$_{seq}$	AdaOAM
glass	**0.813** ± 0.044	0.811 ± 0.049	0.732 ± 0.060	0.795 ± 0.046	0.788 ± 0.040	0.783 ± 0.047
ionosphere	**0.946** ± 0.028	**0.946** ± 0.022	0.902 ± 0.028	0.936 ± 0.018	0.943 ± 0.017	0.938 ± 0.018
german	0.780 ± 0.022	**0.787** ± 0.019	0.741 ± 0.027	0.754 ± 0.022	0.751 ± 0.028	0.770 ± 0.030
svmguide4	0.951 ± 0.014	**0.956** ± 0.012	0.829 ± 0.021	0.843 ± 0.024	0.839 ± 0.022	0.848 ± 0.020
svmguide3	0.784 ± 0.015	**0.793** ± 0.016	0.784 ± 0.019	0.777 ± 0.024	0.780 ± 0.020	0.777 ± 0.024
cod-rna	0.948 ± 0.002	**0.949** ± 0.000	0.887 ± 0.001	0.887 ± 0.001	0.910 ± 0.019	0.887 ± 0.001
spambase	**0.899** ± 0.009	0.898 ± 0.009	0.818 ± 0.019	0.795 ± 0.022	0.849 ± 0.053	0.809 ± 0.014
covtype	0.746 ± 0.005	**0.766** ± 0.003	0.672 ± 0.018	0.674 ± 0.021	0.685 ± 0.016	0.709 ± 0.008
magic04	0.769 ± 0.011	**0.771** ± 0.006	0.734 ± 0.007	0.731 ± 0.015	0.736 ± 0.013	0.752 ± 0.008
heart	**0.883** ± 0.032	0.875 ± 0.026	0.716 ± 0.021	0.753 ± 0.038	0.777 ± 0.043	0.772 ± 0.053
australian	0.841 ± 0.023	**0.842** ± 0.022	0.711 ± 0.056	0.725 ± 0.070	0.742 ± 0.064	0.768 ± 0.036
diabetes	**0.714** ± 0.029	0.705 ± 0.032	0.683 ± 0.037	0.692 ± 0.040	0.694 ± 0.044	0.689 ± 0.040
acoustic	0. 844 ± 0.005	**0.850** ± 0.003	0.840 ± 0.005	0.839 ± 0.002	0.832 ± 0.005	0.841 ± 0.003
vehicle	**0.816** ± 0.018	0.814 ± 0.018	0.764 ± 0.027	0.797 ± 0.014	0.790 ± 0.029	0.805 ± 0.021
segment	**0.838** ± 0.015	0.836 ± 0.008	0.691 ± 0.031	0.768 ± 0.027	0.755 ± 0.024	0.796 ± 0.014

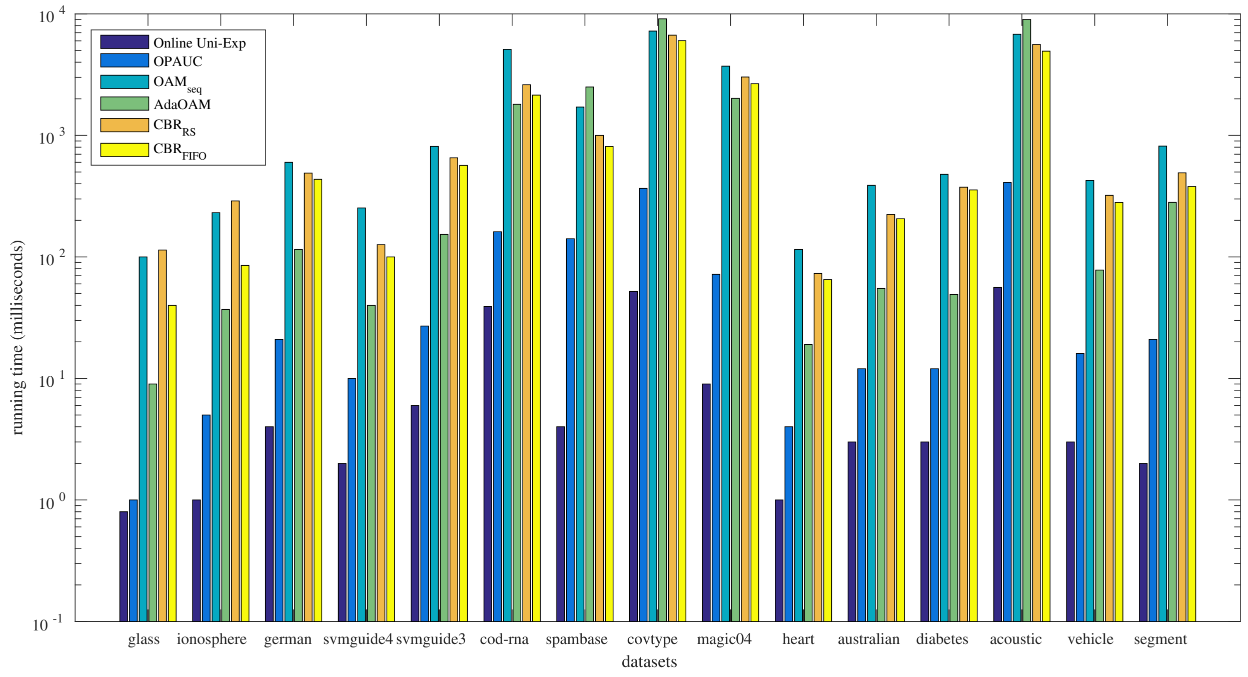


Figure 4.1: Running time (in milliseconds) of CBR and the other online learning algorithms on the benchmark datasets. The *y*-axis is displayed in log- scale.

Table 4.5: AUC performance on the high-dimensional data sets

Data	CBR-diag$_{\text{FIFO}}$	Online Uni-Exp	OPAUCr	OAM$_{\text{seq}}$
farm-ads	**0.961** ± 0.004	0.942 ± 0.006	0.951 ± 0.004	0.952 ± 0.005
rcv1	**0.950** ± 0.007	0.927 ± 0.015	0.914 ± 0.016	0.945 ± 0.008
sector	**0.927** ± 0.009	0.846 ± 0.019	0.908 ± 0.013	0.857 ± 0.008
real-sim	**0.982** ± 0.001	0.969 ± 0.003	0.975 ± 0.002	0.977 ± 0.001
news20	**0.956** ± 0.003	0.939 ± 0.005	0.942 ± 0.006	0.944 ± 0.005
Reuters	**0.993** ± 0.001	0.985 ± 0.003	0.988 ± 0.002	0.989 ± 0.003

4.5.4 Results on High-Dimensional Datasets

We study the performance of the proposed CBR-diag$_{FIFO}$ and compare it with online Uni-Exp, OPAUCr, and OAM$_{seq}$ that avoid constructing the full covariance matrix. Table 4.5 compares our method and the other online algorithms in terms of AUC, while Table 4.6 shows the classification accuracy at OPTROC. Figure 4.2 displays the running time (in milliseconds) comparison.

The results show that the proposed method CBR-diag$_{FIFO}$ yields a better performance on both measures. We observe that the CBR-diag$_{FIFO}$ presents a competitive running time compared to its counterpart OAM$_{seq}$ as shown in Figure 4.2. We can also see that the CBR-diag$_{FIFO}$ takes more running time compared to the OPAUCr. However, the CBR-diag$_{FIFO}$ achieves better AUC and classification accuracy compared to the OPAUCr. The online Uni-Exp algorithm requires the least running time, but it presents lower AUC and classification accuracy compared to our method.

Table 4.6: Comparison of classification accuracy at OPTROC on the high-dimensional data sets

Data	CBR-diag$_{\text{FIFO}}$	Online Uni-Exp	OPAUCr	OAM$_{\text{seq}}$
farm-ads	**0.897** ± 0.007	0.872 ± 0.012	0.885 ± 0.008	0.882 ± 0.007
rcv1	**0.971** ± 0.001	0.967 ± 0.002	0.966 ± 0.003	0.970 ± 0.001
sector	**0.850** ± 0.012	0.772 ± 0.011	0.831 ± 0.015	0.776 ± 0.008
real-sim	**0.939** ± 0.003	0.913 ± 0.005	0.926 ± 0.002	0.929 ± 0.001
news20	**0.918** ± 0.005	0.895 ± 0.005	0.902 ± 0.009	0.907 ± 0.006
Reuters	**0.971** ± 0.004	0.953 ± 0.006	0.961 ± 0.006	0.961 ± 0.006

4.6 Conclusion

In this work, we have developed a linear online soft confidence-weighted bipartite ranking algorithm that maximizes the AUC metric via optimizing a pairwise loss function. The complexity of the pairwise loss function is mitigated in our algorithm by employing a finite buffer that is updated using one of the stream oblivious policies. We have also developed a diagonal variation for the proposed confidence-weighted bipartite ranking algorithm to deal with high-dimensional data by maintaining only the diagonal elements of the covariance matrix instead of the full covariance matrix.

The experimental results on several benchmark and high-dimensional datasets show that our algorithms yield a robust performance. The results also show that the proposed algorithms outperform the first and second-order AUC maximization methods on most of the datasets. In the future, we plan to use the proposed method with a kernel approximation technique to handle complex nonlinear decision boundaries.

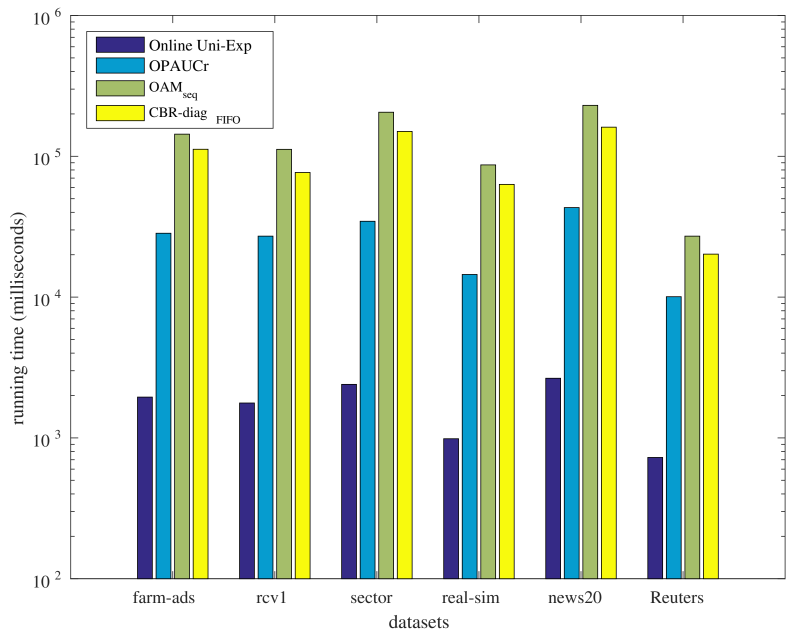


Figure 4.2: Running time (in milliseconds) of CBR-diag$_{FIFO}$ algorithm and the other online learning algorithms on the high-dimensional datasets. The *y*-axis is displayed in log-scale.

Chapter 5

Kernelized Second-Order Online AUC Maximization

5.1 Introduction

Despite its scalability, online linear learning algorithms fail to model linearly inseparable datasets, which is not uncommon in real-world applications. The lack of modeling nonlinearity motivates online kernel learning approaches [74, 75, 76]. In the kernel-based online learning, the learner maintains a set of support vectors in memory to build the kernel of the predictive model. Conventionally, any misclassified instance is considered a support vector, and therefore, is added to the set of support vectors. However, the number of support vectors quickly becomes too large, which in turns reduces the efficiency of online learning. To address the unbounded growth of support vectors, fixed budget size has been introduced to keep the most representative support vectors using different updating techniques whenever the budget is excesses [77, 78, 79].

For maximizing the AUC objective function on a nonlinear space, the budget kernel learning is adopted by [80]. However, the computation of this method inhibits its scalability when applied to large-scale datasets. A sparse kernel AUC maximization algorithm is developed in [58] to speed up the kernel learning and to overcome the low generalization capability of kernel online learning. However, sparse algorithms are prone to the under-fitting problem due to the sparsity of the model, especially for large datasets.

Embedding the input space into a finite approximate feature space is another approach to approximate online kernel learning. Specifically, the kernel matrix can be approximated by Nyström low-rank approximation, while the random Fourier method is used to approximate the kernel function. Then a linear model can be applied to the approximate feature space to solve a classification or regression tasks [11].

These embedding approaches are adopted by [81] for maximizing the AUC objective function. However, no attempt has been made to learn an online second-order linear classifier on the embedded space. While second-order learning algorithms, such as confidence-weighted classifier [82], are able to handle linearly inseparable data, they fail to model complex nonlinear functions.

In this work, we propose an online nonlinear second-order classifier to optimize the AUC objective function on a space embedded by random Fourier features. In particular, we extend our online linear confidence-weighted AUC classifier to learn a nonlinear classifier, where nonlinearity is induced via approximate feature mappings.

5.2 Related Work

Online Kernel Learning. Online kernel learning is proposed to address the problem of online linear learning with modeling nonlinear decision boundary. It is common in online kernel learning to bound the number of support vectors using a fixed budget size in order to reduce the computation of constructing the kernel. Multiple budget update strategies are proposed such as removal [77, 78], projection [79], and merging [10]. In order to reduce the training and test time of budget kernel learning, kernel functional approximation approaches (i.e., Nyström and random Fourier features) are explored in [11]. A dual space [83] uses random Fourier features to store the information of the removed SV from the budget, and hence enhancing the accuracy using only a small number of random features. Recently, the work by [84] improves the performance of online Fourier features by optimizing the kernel parameters.

Online Kernel AUC Maximization. The work authors of [85] adopt the budget online kernel learning technique in their online algorithm for AUC maximization. This framework [85] uses a fixed-size buffer for each class label. These buffers are utilized to deal with the pairwise loss function and to maintain positive and negative support vectors. However, the computation of the kernel undermines this method, especially for extremely large datasets. In [81], nonlinear AUC maximization methods are devised using Nyström approximation and random Fourier features. In these methods [81], first-order online learning is utilized to optimize the AUC objective function on the embedded feature space.

5.3 Kernelized Confidence-Weighted AUC Maximization

We consider a problem of learning a nonlinear function $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of imbalanced labeled data points. Let (x_t, y_t) denotes the labeled data point received at the t-th trial, where $x_t \in \mathbb{R}^d$ is the training instance of d-dimension and $y_t \in \{1, 1\}$ is the associated class label. The predictive model of the kernel AUC maximization $f(x)$ for a new incoming instance x is identical to the one of online kernel learning, and it can be defined by:

$$f(x) = \sum_{i=1}^{m} \alpha_i \langle \varphi(x_i), \varphi(x) \rangle = \sum_{i=1}^{m} \alpha_i \kappa(x_i, x),$$

where m is the number of support vector, $\varphi(\cdot)$ is a mapping from the input space into some Hilbert space, α_i indicates the coefficient of the i-th support vector, and $\kappa(x_i, x)$ is the kernel function over a pair of instances. In practice, implementing the kernel function is what we need to compute because it implied the computation of the inner products in the embedded space.

However, it is more practical to approximate the exact feature map and then solve the problem in the primal space constructed by the approximate mappings. Let $\psi(x)$ denotes an approximate mapping that can replace the original feature map $\varphi(x)$, and the predictive model of the kernel AUC maximization can be reformulated as follows,

$$f(x) = \sum_{i=1}^{m} \alpha_i \kappa(x_i, x) \approx \sum_{i=1}^{m} \alpha_i \psi(x_i)^T \psi(x) = w^T \psi(x),$$

where $w \equiv \sum_{i=1}^{m} \alpha_i \psi(x_i)$ is the model learned in the approximate feature space obtained via $\psi(x)$.

Prior to describing the approximate mapping that we use in our model, we briefly review the AUC maximization problem. Given a training dataset $\mathcal{S} = \{x_i^+ \cup x_j^- \in \mathbb{R}^d | i = \{1, \ldots, n^+\}, j = \{1, \ldots, n^-\}\}$, where x_i^+ is the i-th positive instance and x_j^- is the j-th negative instance. The n^+ and n^- denote the number of positive and negative instances received thus far. The AUC loss function in the transformed space can be defined by:

$$\mathcal{L}(f;\mathcal{S}) = \sum_{i=1}^{n^+}\sum_{j=1}^{n^-} \frac{I_{(w^T\psi(x_i^+) > w^T\psi(x_j^-))} + \frac{1}{2}I_{(w^T\psi(x_i^+) = w^T\psi(x_j^-))}}{n^+n^-},$$

where $I(\cdot)$ is the indicator function that outputs 1 if the condition is held, and 0 otherwise. Due to its discrete nature and hence it is difficult to optimize, a convex surrogate function is used to replace the indicator function. The AUC measure can then be defined as follows,

$$\mathcal{L}(f;\mathcal{S}) = \sum_{i=1}^{n^+}\sum_{j=1}^{n^-} \frac{\ell(w^T(\psi(x_i^+) - \psi(x_j^-)))}{2n^+n^-},$$

where $\ell(\cdot)$ is a convex surrogate pairwise loss function (e.g., pairwise hinge loss function). The pairwise loss function entails storing all the received instances from the opposite class label to compute the loss function, which is impractical. We handle this problem using a fixed buffer size for each class label to store a sample from the received instances of each class label.

Our proposed method optimizes the AUC loss function using a second-order classifier. In particular, we use a confidence-weighted classifier that maintains a model as Gaussian distribution parameterized by mean vector $\mu \in \mathbb{R}^m$ and covariance matrix $\Sigma \in \mathbb{R}^{m\times m}$, where m is the dimension of the embedded space. The mean vector represents the weight model, while the covariance matrix scales the weight's parameters adaptively.

The model distribution is updated based on minimizing the Kullback-Leibler divergence between the new and the old model distributions [61, 71, 59]. The confidence-weighted kernel AUC maximization is formulated as follows:

$$\begin{aligned}(\mu_{t+1}, \Sigma_{t+1}) = \underset{\mu,\Sigma}{\text{argmin}} & D_{KL}(\mathcal{N}(\mu,\Sigma)||\mathcal{N}(\mu_t,\Sigma_t)) \\ & + C\ell^{\phi}(\mathcal{N}(\mu,\Sigma);(\psi(x_t) - \psi(x), y_t),\end{aligned} \tag{5.1}$$

where C is the the penalty hyperparamter, $\phi = \Phi^{-1}(\eta)$, and Φ is the normal cumulative distribution function. The loss function $\ell^{\phi}(\cdot)$ is defined as:

$$\begin{aligned}\ell^{\phi}(\mathcal{N}(\mu,\Sigma);(\psi(x_t)-\psi(x),y_t) = max(0,\phi\sqrt{(\psi(x_t)-\psi(x))^T\Sigma(\psi(x_t)-\psi(x))}\\ -y_t\ \mu\ (\psi(x_t)-\psi(x))).\end{aligned}$$

The solution of 5.1 is given by the following proposition.

Proposition 3. *The optimization problem 5.1 has a closed-form solution as follows:*

$$\mu_{t+1} = \mu_t + \frac{\alpha_t}{2} y_t \Sigma_t z,$$

$$\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t z^T z \Sigma_t,$$

where $z = \psi(x_t) - \psi(x)$ and the coefficients α and β are defined as follows:

$\alpha_t = min\{C, max\{0, \frac{1}{\upsilon_t\zeta}(-q_t\delta + \sqrt{q_t^2\frac{\phi^4}{4} + \upsilon_t\phi^2\zeta})\}\}$,

$\beta_t = \frac{\alpha_t\phi}{\sqrt{u_t}+\upsilon_t\alpha_t\phi}$. *where* $u_t = \frac{1}{4}(-\alpha_t\upsilon_t\phi + \sqrt{\alpha_t^2\upsilon_t^2\phi^2 + 4\upsilon_t})^2$,
$\upsilon_t = z^T\Sigma_t z$, $q_t = y_t(\mu_t \cdot z)$, $\phi = \Phi^{-1}(\eta)$, $\delta = 1 + \frac{\phi^2}{2}$, $\zeta = 1 + \phi^2$,

The proposition 3 is analogous to the one derived in [59].

5.3.1 Fourier Confidence-Weighted AUC Maximization

We use random Fourier features to approximate a shift-invariant kernel in an online fashion. The approximated kernel mapping is plugged into the confidence-weighted classifier designed to maximize the AUC measure [82]. Based on Bochner's theorem [49], the Fourier transform $p(\theta)$ of a shift-invariant kernel is a proper probability distribution. Therefore, a shift-invariant kernel can be expressed as below,

$$k(x-y) = \frac{1}{2\pi} \int p(\theta) e^{-i\theta^T(x-y)} d\theta.$$

$$= 2\ E_{\theta \sim p(\theta)}\ [e^{i\theta^T(x-y)}]$$

$$= 2\ E_{\theta \sim p(\theta)}\ [cos(\theta^T x)\ cos(\theta^T y) + sin(\theta^T x)\ sin(\theta^T y)]$$

$$= 2\ E_{\theta \sim p(\theta)}\ [[sin(\theta^T x), cos(\theta^T y)] \cdot [sin(\theta^T x), cos(\theta^T y)]].$$

The embedding of a data point x is defined as the following concatenation of cosine and sine functions $\psi(x) = [cos(\theta^T x), sin(\theta^T x)]$. For an embedding of size m dimension, we sample $\{\theta_i\}_{i=1}^m$ Fourier components, and the Fourier mapping function is then obtained as follows,

$$\psi(x) = \sqrt{2/m}[sin(\theta_1^T x), cos(\theta_1^T x)\ , \cdots, sin(\theta_m^T x), cos(\theta_m^T x)]. \tag{5.2}$$

In this work, we use the shift-invariant Gaussian kernel $k(x,y) = exp(-\frac{||x-y||_2^2}{2\sigma^2})$. Therefore, the corresponding random Fourier component θ are sampled independently from the distribution $p(\theta) = \mathcal{N}(0, \sigma^{-2}I)$.

The framework of the confidence-weighted kernel AUC maximization is illustrated in Algorithm 8. The algorithm has three main steps. The first step is to embed the received instance into a finite-dimensional feature space. The embedding is carried out by applying the Fourier mapping function 5.2 to the received instances.

The second step is to update the corresponding buffer of the received instance's class label. We use two oblivious strategies to update the buffers when they are full. The first policy is Reservoir sampling (RS), which imitates the uniform random sampling from the training data. Algorithm 7 shows the steps of the Reservoir sampling approach. The second policy is First-In-First-Out

Algorithm 7: Reservoir Sampling Approach

```
Input: $x_t$, $B^t$, $M$, $M_{t+1}$
Output: updated buffer $B^{t+1}$
if $|B^t| < M$ then
    $B^{t+1} = B^t \cup \{x_t\}$
else
    Sample $Z$ from a Bernoulli distribution with $Pr(Z = 1) = M/M_{t+1}$
    if $Z = 1$ then
        Randomly delete an instance from $B^t$
        $B^{t+1} = B^t \cup \{x_t\}$
    end if
end if
Return $B^{t+1}$
```

(FIFO), which allows any received instance to be included in the buffer by letting the new instance replaces the first added instance.

The third step updates the kernelized AUC classifier in the approximate features space by solving the kernelized AUC confidence-weighted objective function described in equation 5.1. The update steps of the classifier are outlined in Algorithm 9.

Algorithm 8: Framework for Kernelized Confidence-Weighted AUC Maximization

```
Input:
  • the penalty parameter $C$ and the probability value $\eta$
  • the maximum buffer size $M_+$ and $M_-$
  • the Gaussian kernel function $k(\cdot,\cdot)$ with the kernel width $\sigma$.
  • the number of random Fourier components $m$
Initialize: $a_i = 1$ for $i \in 1,\dots,m$, $\mu_1 = \{0,\dots,0\}^m$, $\Sigma_1 = diag(a)$, $B_+ = B_- = \emptyset$
$M_+^1 = M_+^1 = 0$
Generate i.i.d. random Fourier components: $\theta_1,\cdots,\theta_m$ sampled from the distribution
$p(\theta) = \mathcal{N}(0,\sigma^{-2}I)$.
for $t = 1,\dots,T$ do
   Receive a training instance $(x_t, y_t)$
   Transform the received instance $x_t$
   $\psi(x) = (sin(\theta_1^T x), cos(\theta_1^T x), \cdots, sin(\theta_m^T x), cos(\theta_m^T x))^T$.
   if $y_t = +1$ then
      $B_-^{t+1} = B_-^t$, $M_+^{t+1} = M_+^t + 1$, $M_-^{t+1} = M_-^t$
      $C_t = C$
      $B_+^{t+1}$ = UpdateBuffer($\psi(x_t), B_+^t, M_+, M_+^{t+1}$)
      $[\mu_{t+1}, \Sigma_{t+1}]$ = UpdateClassifier($\mu_t, \Sigma_t, \psi(x_t), y_t, C_t, B_-^{t+1}, \eta$)
   else
      $B_+^{t+1} = B_+^t$, $M_-^{t+1} = M_-^t + 1$, $M_+^{t+1} = M_+^t$
      $C_t = C$
      $B_-^{t+1}$ = UpdateBuffer($\psi(x_t), B_-^t, M_-, M_-^{t+1}$)
      $[\mu_{t+1}, \Sigma_{t+1}]$ = UpdateClassifier($\mu_t, \Sigma_t, \psi(x_t), y_t, C_t, B_+^{t+1}, \eta$)
   end if
end for
```

5.4 Experiments

In this section, we evaluate the proposed second-order algorithms on several benchmark datasets and compare their performance in terms of AUC accuracy and running time to the existing linear and kernelized AUC maximization methods. The aim of including the linear AUC maximization methods in the experiments is to show the advantage of the kernelized methods over linear methods. We also study the performance of the proposed second-order methods on a different number of random features and compare its AUC performance with their first-order counterpart method.

Algorithm 9: Update Classifier

```
Input:
    • $\mu_t$ : current mean vector
    • $\Sigma_t$ : current covariance matrix
    • $(\psi(x_t), y_t)$ : a transformed labeled instance
    • $B$ : the buffer storing instances from the opposite class label
    • $C_t$ : weighting parameter
    • $\eta$ : the predefined probability
Output: updated classifier:
    • $\mu_{t+1}$
    • $\Sigma_{t+1}$
Initialize: $\mu^1 = \mu_t$, $\Sigma^1 = \Sigma_t$, $i = 1$
for $\psi(x) \in B$ do
    Update the classifier $(\mu^i, \Sigma^i)$ with $z = \psi(x_t) - \psi(x)$ and $y_t$ by
    $(\mu^{i+1}, \Sigma^{i+1}) = \underset{\mu,\Sigma}{\text{argmin}} D_{KL}(\mathcal{N}(\mu,\Sigma)||\mathcal{N}(\mu^i,\Sigma^i)) + C\ell^{\phi}(\mathcal{N}(\mu,\Sigma);(z,y_t))$
    $i = i + 1$
end for
Return $\mu_{t+1} = \mu^{|B|+1}$
       $\Sigma_{t+1} = \Sigma^{|B|+1}$
```

5.4.1 Benchmark Datasets

We conduct the experiments on 12 benchmark datasets. We download the datasets from LibSVM[6] and UCI[7]. The multi-class datasets (i.e., glass, vehicle, segment, pendigits, acoustic, and covtype) are transformed randomly into class-imbalanced binary datasets. For glass, vehicle, and acoustic datasets, we transform their classification tasks to distinguish class label one from the rest of the classes. For segment dataset, we transform its task into imbalanced binary classification by grouping the labels less than or equal three into one class and the rest of the labels into another class. The classification task of pendigits is transformed to distinguish classes less than or equal label one from the rest of the classes. For covtype dataset, an imbalanced binary classification task is generated to distinguish class label 7 from the rest of the classes. Tables 5.1 shows the characteristics of the datasets. The ratio for each dataset is calculated by dividing the majority class label by

[6]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

[7]https://archive.ics.uci.edu/ml/

the minority one. Each dataset is standardized by removing the mean of each feature and dividing the features by their corresponding standard deviation.

Table 5.1: Benchmark data sets

Data	#inst	#feat	ratio
glass	214	9	2.05
vehicle	846	18	2.99
svmguide3	1,284	22	2.81
segment	2,310	19	1.33
spambase	4,601	57	1.53
magic04	19,020	10	1.84
pendigits	10,992	16	1.04
cod-rna	331,152	8	2.0
acoustic	78,823	50	3.30
ijcnn1	49,990	22	9.3
kddcup08	102,294	117	163.19
covtype	581,012	54	27.32

5.4.2 Compared Methods and Model Selection

1. **OAM$_{\textbf{seq}}$** and **OAM$_{\textbf{gra}}$** [65]: The online AUC maximization (OAM) is the state-of-the-art first-order learning method. These algorithms are implemented with the Reservoir Sampling as a buffer updating scheme. The size of the positive and negative buffers is fixed at 100. The penalty hyperparameter C is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

2. **OPAUC** [66]: An online learning algorithm that optimizes the AUC in one-pass through square loss function. The learning rate is tuned by 3-fold cross validation by searching in $2^{[-10:10]}$, and the regularization hyperparameter is set to a small value 0.0001.

3. **AdaOAM** [67]: This is a second-order AUC maximization method that adapts the classifier to the importance of features. The smooth hyperparameter δ is set to 0.5, and the regulariza-

tion hyperparameter is set to 0.0001. The learning rate is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

4. **CBR$_{\mathbf{RS}}$** and **CBR$_{\mathbf{FIFO}}$** [82]: These are the confidence-weighted AUC maximization methods with the Reservoir Sampling (CBR$_{\text{RS}}$) and First-In-First-Out (CBR$_{\text{FIFO}}$) buffer updating policies. The size of the positive and negative buffers is fixed at 50. The hyperparameter η is set to 0.7, and the penalty hyperparameter C is tuned by 3-fold cross-validation by searching in $2^{[-10:10]}$.

5. **FOAM [81]**: The kernelized first-order AUC maximization with random Fourier features. This method implements the Reservoir sampling to update the buffer. The buffer size is set to 100. The learning rate is tuned by 3-fold cross validation on the training set by searching in $2^{[-10:10]}$.

6. **FCBR$_{\mathbf{RS}}$** and **FCBR$_{\mathbf{FIFO}}$**: The proposed second-order AUC maximization methods with random Fourier features. FCBR$_{\text{RS}}$ uses the Reservoir sampling and FCBR$_{\text{FIFO}}$ implements the First-In-Firt-Out to update the buffers. The size of the positive and negative buffers is fixed at 50. The hyperparameter η is set to 0.7, and the learning rate C is tuned by 3-fold cross validation by searching in $2^{[-10:10]}$.

For the kernelized AUC maximization algorithms (i.e., FOAM, FCBR$_{\text{RS}}$, and FCBR$_{\text{FIFO}}$), we set the number of random Fourier components to 150. Consequently, the dimension of the approximate feature space will be 300. In all experiments, we use Random Fourier to approximate the Gaussian kernel and set its width to 100. Choosing the best width via cross-validation might also improve the results. The reported results are the average of 3 runs. In each run, we also averaged the results over 5-fold cross-validation.

Table 5.2: AUC classification accuracy and running time for different online AUC classifiers. The running time is in seconds.

Algorithm	glass		vehicle		svmguide3	
	AUC	time	AUC	time	AUC	time
OAM$_{\text{seq}}$	0.8063 ± 0.0189	0.0001	0.7532 ± 0.0274	0.0016	0.7592 ± 0.0096	0.0032
OAM$_{\text{gra}}$	0.7960 ± 0.0193	0.0001	0.7531 ± 0.0232	0.0013	0.7639 ± 0.0068	0.0024
OPAUC	0.8145 ± 0.0278	0.0003	0.7640 ± 0.0080	0.0021	0.7734 ± 0.0117	0.0038
AdaOAM	0.8098 ± 0.0299	0.0003	0.7683 ± 0.0199	0.0021	0.7903 ± 0.0100	0.0038
CBR$_{\text{RS}}$	0.8088 ± 0.0380	0.0032	0.8533 ± 0.0098	0.0294	0.8081 ± 0.0048	0.0563
CBR$_{\text{FIFO}}$	0.8120 ± 0.0283	0.0032	0.8558 ± 0.0098	0.0268	0.8153 ± 0.0098	0.0526
FOAM	0.8164 ± 0.0018	0.0325	0.7678 ± 0.0142	0.3010	0.7763 ± 0.0344	0.5152
FCBR$_{\text{RS}}$	0.8228 ± 0.0222	0.5050	0.8961 ± 0.0067	2.2611	0.8275 ± 0.0063	4.1687
FCBR$_{\text{FIFO}}$	0.8201 ± 0.0269	0.5415	0.9015 ± 0.0001	2.2933	0.8346 ± 0.0162	4.2970

Algorithm	segment		spambase		magic04	
	AUC	time	AUC	time	AUC	time
OAM$_{\text{seq}}$	0.8947 ± 0.0024	0.0045	0.9542 ± 0.0021	0.0181	0.6993 ± 0.0190	0.0267
OAM$_{\text{gra}}$	0.8958 ± 0.0068	0.0038	0.9521 ± 0.0020	0.0174	0.7193 ± 0.0136	0.0227
OPAUC	0.9084 ± 0.0038	0.0060	0.9511 ± 0.0017	0.0635	0.8383 ± 0.0028	0.0275
AdaOAM	0.9097 ± 0.0035	0.0060	0.9536 ± 0.0009	0.0637	0.8384 ± 0.0026	0.0270
CBR$_{\text{RS}}$	0.9176 ± 0.0032	0.0813	0.9669 ± 0.0001	0.5634	0.8357 ± 0.0023	0.4792
CBR$_{\text{FIFO}}$	0.9191 ± 0.0038	0.0740	0.9669 ± 0.0014	0.5113	0.8406 ± 0.0014	0.4323
FOAM	0.9102 ± 0.0033	1.0225	0.9640 ± 0.0013	2.8374	0.8358 ± 0.0049	8.0778
FCBR$_{\text{RS}}$	0.9885 ± 0.0035	5.3164	0.9685 ± 0.0011	11.667	0.9016 ± 0.0014	56.792
FCBR$_{\text{FIFO}}$	0.9894 ± 0.0026	5.1665	0.9724 ± 0.0006	11.585	0.9201 ± 0.0013	55.475

Algorithm	pendigits		cod-rna		acoustic	
	AUC	time	AUC	time	AUC	time
OAM$_{\text{seq}}$	0.7968 ± 0.0152	0.0175	0.9670 ± 0.0066	0.2750	0.7809 ± 0.0139	0.2947
OAM$_{\text{gra}}$	0.7752 ± 0.0131	0.0156	0.9709 ± 0.0095	0.2677	0.7644 ± 0.0029	0.2826
OPAUC	0.9030 ± 0.0028	0.0214	0.9856 ± 0.0003	0.3746	0.8627 ± 0.0010	0.8504
AdaOAM	0.9038 ± 0.0023	0.0210	0.9862 ± 0.0000	0.3666	0.8738 ± 0.0006	0.8579
CBR$_{\text{RS}}$	0.9004 ± 0.0014	0.2810	0.9864 ± 0.0001	5.1410	0.8765 ± 0.0010	10.131
CBR$_{\text{FIFO}}$	0.9056 ± 0.0016	0.2550	0.9869 ± 0.0000	4.7122	0.8920 ± 0.0005	9.1061
FOAM	0.9286 ± 0.0078	4.8981	0.9871 ± 0.0007	138.55	0.8767 ± 0.0003	46.847
FCBR$_{\text{RS}}$	0.9983 ± 0.0000	24.493	0.9903 ± 0.0000	744.89	0.8855 ± 0.0003	269.05
FCBR$_{\text{FIFO}}$	0.9988 ± 0.0001	23.620	0.9917 ± 0.0000	719.57	0.9049 ± 0.0005	257.14

Algorithm	ijcnn1		kddcup08		covtype	
	AUC	time	AUC	time	AUC	time
OAM$_{\text{seq}}$	0.8906 ± 0.0018	0.1027	0.8874 ± 0.0057	0.7741	0.9637 ± 0.0026	2.2459
OAM$_{\text{gra}}$	0.8831 ± 0.0047	0.0901	0.8988 ± 0.0065	0.7841	0.9623 ± 0.0012	2.2127
OPAUC	0.9301 ± 0.0012	0.1463	0.5835 ± 0.0393	5.5734	0.5099 ± 0.0184	7.3215
AdaOAM	0.9322 ± 0.0011	0.1493	0.9250 ± 0.0007	5.6170	0.9769 ± 0.0002	7.4047
CBR$_{\text{RS}}$	0.9249 ± 0.0005	1.9243	0.9074 ± 0.0017	44.626	0.9762 ± 0.0005	73.212
CBR$_{\text{FIFO}}$	0.9350 ± 0.0009	1.7720	0.9245 ± 0.0028	41.214	0.9806 ± 0.0001	57.798
FOAM	0.9290 ± 0.0016	23.4473	0.9157 ± 0.0023	87.712	0.9752 ± 0.0007	370.31
FCBR$_{\text{RS}}$	0.9758 ± 0.0004	114.85	0.9029 ± 0.0028	296.41	0.9831 ± 0.0008	1394.7
FCBR$_{\text{FIFO}}$	0.9855 ± 0.0004	116.10	0.9272 ± 0.0033	303.39	0.9901 ± 0.0002	1365.5

5.5 Results and Discussion

The results in terms of AUC classification accuracy and running time are shown in Table 5.2. We report the results of non-kernelized AUC maximization methods to show the advantage of using approximate kernel techniques.

We observe that the AUC performance of our proposed nonlinear AUC maximization method surpasses the linear methods. This includes its linear counterparts CBR_{RS} and CBR_{FIFO}. We also noticed that the proposed FCBR_{RS} and $\text{FCBR}_{\text{FIFO}}$ require more running time compared to FOAM because of the matrix update step. However, the AUC classification performance of $\text{FCBR}_{\text{FIFO}}$ consistently surpasses the AUC performance of FOAM. FCBR_{RS} also demonstrates a competitive AUC performance compared to FOAM except on kddcup08 dataset.

We see that the linear methods CBR_{RS} and CBR_{FIFO} perform well on most datasets compared to OPAUC and AdaOAM, which optimize a squared AUC loss function. In addition, CBR_{RS} and CBR_{FIFO} achieve higher AUC classification compared to FOAM on nine datasets. The methods CBR_{RS} and CBR_{FIFO} also require less running time compared to FOAM. However, they are still not able to handle complex nonlinear structure underlying some datasets.

In addition, the method $\text{FCBR}_{\text{FIFO}}$ consistently outperforms FCBR_{RS}. This indicates that FIFO policy is better than Reservoir sampling in updating the buffer. However, they do not differ significantly in terms of running time.

Study on the Number of Random Features

We study the performance of the proposed methods FCBR_{RS} and $\text{FCBR}_{\text{FIFO}}$ on a different number of random features and compare their AUC classification accuracies to the ones obtained by FOAM method. The results are shown in Figure 5.1. We vary the number of Fourier features from 30 to 250. The y-axis shows the averaged AUC classification accuracy over 5 folds cross-validation.

We can observe that the classification accuracy of $\text{FCBR}_{\text{FIFO}}$ is better than FCBR_{RS} and FOAM. Further, The curves of our both methods $\text{FCBR}_{\text{FIFO}}$ and FCBR_{RS} show less fluctuation with varying the number of random features compared to FOAM. This would facilitate the grid search for finding the best number of features.

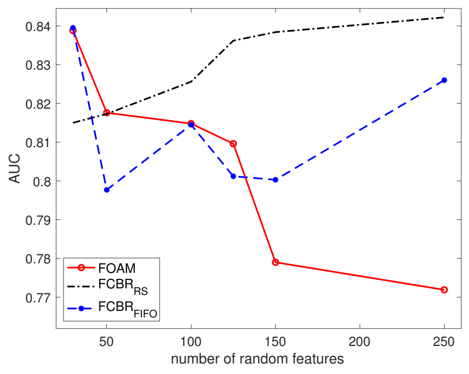
We also note that on some datasets the FOAM is susceptible to overfitting when increasing the number of random features. However, our methods demonstrate that they are less sensitive to this phenomenon. We attribute this virtue of our algorithms to confidence-weighted learning.

5.6 Conclusion

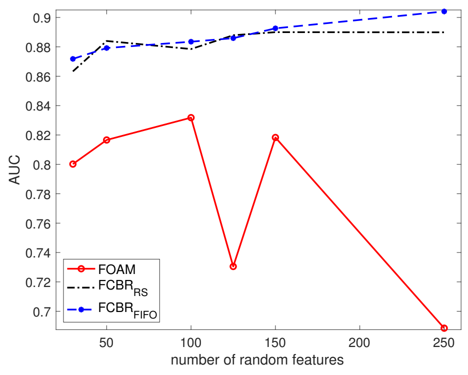
In this work, we have proposed a scalable online nonlinear AUC maximization algorithm. We approximate the function of the Gaussian kernel using random Fourier features and implement our confidence-weighted AUC classifier on the embedded space. We experiment with both First-In-First-Out and Reservoir sampling to deal with the multivariate nature of the AUC loss function.

The results on several benchmark datasets show that our method outperforms linear and nonlinear AUC maximization methods. The advantage of our method is learning a second-order AUC maximization classifier on the approximate space instead of using a first-order classifier, which is implemented by FOAM.

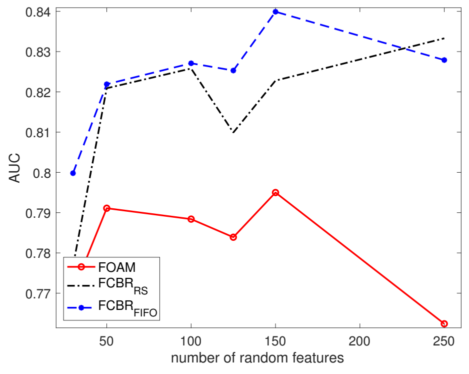
In further research, we will study the usage of a data-dependent method (i.e., Nyström low-rank approximation) with a second-order classifier to enhance the AUC classification accuracy. Another future work can investigate to improve the efficiency of our method by approximating its covariance matrix.



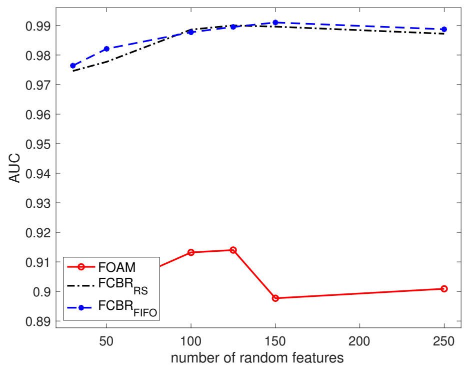
(a) glass



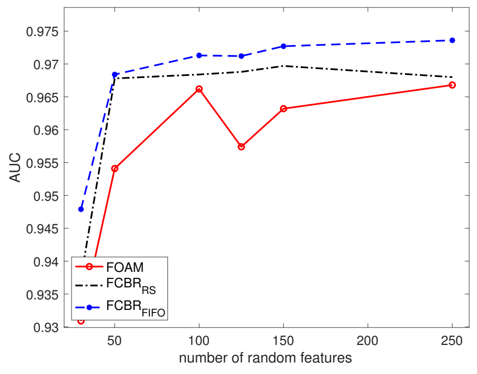
(b) vehicle



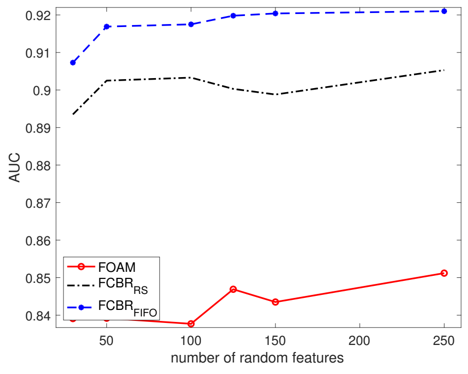
(c) svmguide3



(d) segment

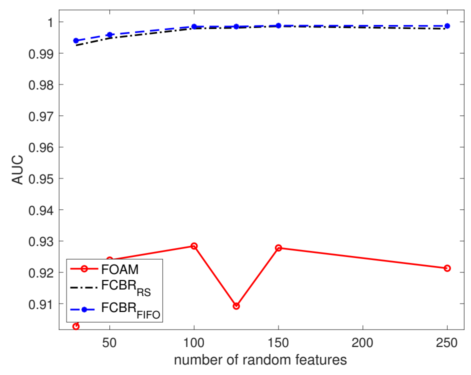


(e) spambase

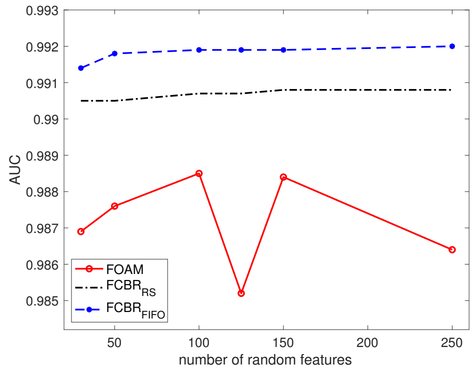


(f) magic04

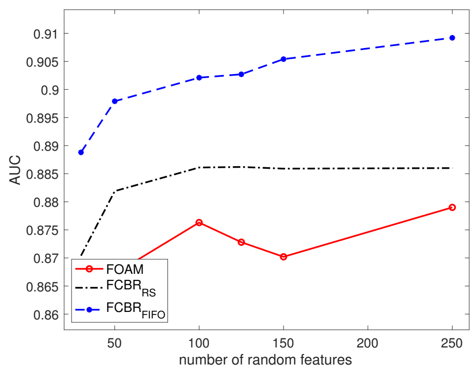
Figure 5.1: Study on the classification accuracy of random Fourier AUC maximization methods with a different number of random features



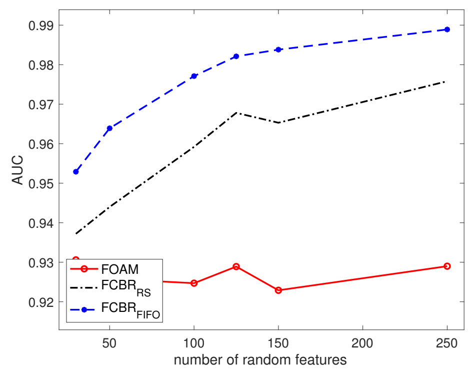
(g) pendigits



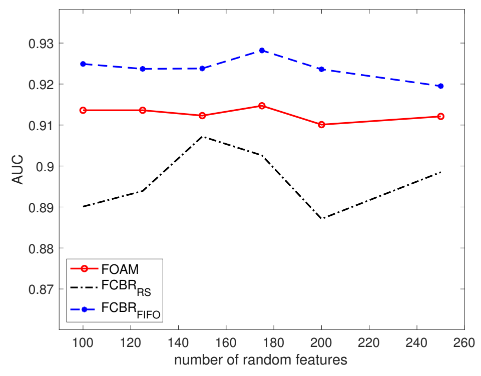
(h) cod-rna



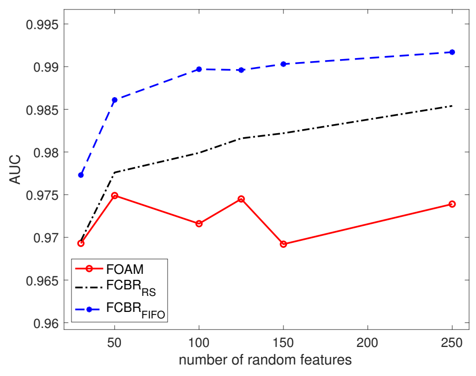
(i) acoustic



(j) ijcnn1



(k) kddcup08



(l) covtype

Figure 5.1: Study on the classification accuracy of random Fourier AUC maximization methods with a different number of random features

Chapter 6

Accelerated Stochastic AUC Maximization

6.1 Introduction

The optimization of the AUC objective function is typically solved using Batch methods (e.g., inexact Hessian using conjugate gradient [42]). These batch methods yield a high AUC classification accuracy because they can reach to the optimal minimum. However, these batch methods have two limitations. First, they are not able to deal with large-scale datasets because of their high computational time. Second, they cannot handle sequential data, due to the deterministic nature of their optimization approaches.

On the other hand, online learning methods for AUC maximization [65, 66] tackle the scalability of batch AUC maximization methods by running one-pass over the dataset. Such methods lack optimal convergence because they do not benefit from second-order information. Hence, recent online AUC maximization methods consider online second-order optimization techniques to improve the convergence rate [67, 82]. However, forming the exact covariance matrix reduce the scalability of these methods when learning on high dimensional data.

A challenging problem here lies in developing an AUC algorithm that converges to the optimal minimum with respect to the surrogate loss function while its rate of convergence is fast. One popular optimization technique is stochastic learning [86]. The appealing property of stochastic gradient is that its convergence rate is not dependent on the size of the sample set.

However, stochastic learning suffers from slow convergence (sublinear rate of convergence), due to the noisy gradient estimates. A plethora of methods have been developed to address the noise reduction problem for univariate loss functions, but only a few studies have considered reducing the noise in the presence of a pairwise loss function.

We propose a fast convergence first-order stochastic AUC maximization algorithm. The proposed algorithm improves the sublinear convergence of the first-order stochastic algorithm by employing variance reduction methods. Specifically, we suggest a scheduled averaging and combine it with a scheduled regularization to accelerate the convergence rate of the stochastic AUC classifier. To deal with nonlinearly distributed datasets, we optimize our linear classifier on a finite-dimensional approximate feature space constructed via the k-means Nyström low-rank approximation and the random Fourier features.

6.2 Related work

The proposed accelerated stochastic AUC maximization method is mostly related to noise reduction, adaptive stochastic gradient methods, and Fast convergence stochastic online AUC maximization topics. In what follows, we briefly review the main approaches in these two areas.

Noise reduction

One of the main approaches to remedy the slow convergence of stochastic gradient is known as noise reduction. The noise can be reduced by learning a basic stochastic algorithm in a mini-batch setting where the batch size increases periodically [87]. Other stochastic algorithms are developed based on the idea of utilizing the previously estimated directions (i.e., gradients) in improving the quality of the new one. This approach includes SAG [88], SVRG [89], and SAGA [90].

Iterate averaging is another intuitive technique improving the final noisy model by averaging not the gradients but the resulting estimators of each iteration [91]. This auxiliary averaged estimator is not involved during the optimization process and only used as a final model. Scheduling the regularization to improve the convergence of the standard stochastic gradient descent is proposed in [92].

Adaptive stochastic gradient

Another approach to improve the sublinear rate of convergence of the standard stochastic gradient is based on estimating adaptive learning rates for the coefficients of the model in each update step. Examples of such adaptive approach including Momentum [93], Nesterov [94], RMSprop [95], AdaGrad [62], and Adam [96]. These methods have been successfully applied to accelerate the convergence of stochastic gradient solving non-convex optimization problems (e.g., Neural network).

Fast convergence stochastic online AUC maximization

The work by [97] suggests using sampling with replacement to reduce the variance and hence improving the convergence rate of the vanilla stochastic gradient. Our online AUC maximization algorithm [82] improves the convergence rate by exploiting the second-order information in a confidence-weighted style. However, the second-order information compensates the scalability of the algorithm when solving high dimensional data or when embedding the input space into high finite-dimensional approximate feature space.

Another work [68] views the problem of scaling up the AUC objective function lies on how to handle its pairwise loss function. Therefore, the authors of [68] suggest to formulate the AUC objective function as a convex-concave saddle point problem (SOLAM). This formulation enables them to solve a univariate loss function for AUC maximization. Recently, an adaptive multi-stage algorithm [98] and a proximal variant of SOLAM [99] are proposed to improve the convergence rate of SOLAM.

6.3 Accelerated Stochastic AUC Maximization Algorithm

Given a training a dataset $S = \{x_i, y_i\}_{i=1}^{n}$ drawn from unknown distribution $\mathcal{D}$, where the input space has n instances with d dimensional features $x \in \mathbb{R}^d$ and $y = \{-1, 1\}$. The regularized empirical risk maximizing the AUC measure can be formulated as the minimization of the following cost function:

$$\begin{aligned}\min_{w} R(w) &= \frac{\lambda}{2}||w||^2 + \frac{1}{n^+n^-}\sum_{i=1}^{n^+}\sum_{j=1}^{n^-}\ell(w^T(x_i^+ - x_j^-)) \\ &= \frac{1}{n^+n^-}\sum_{i=1}^{n^+}\sum_{j=1}^{n^-}\Big(\frac{\lambda}{2}||w||^2 + \ell(w^T(x_i^+ - x_j^-))\Big),\end{aligned} \tag{6.1}$$

where the hyper-parameter $\lambda > 0$ controls the effect of the regularization term and $\ell(\cdot)$ is a surrogate loss function. The surrogate loss function can be twice differential, which has been proven to be consistent with AUC measure [66, 38]. However, we opt to implement a hinge loss function, which is nondifferentiable, because using first-order information is more efficient when working on the large-scale regime.

While the hinge loss is nondifferentiable, it is still possible to apply the gradient descent by using the subgradient instead of the gradient. The subgradient of the pairwise hinge loss function at w can be defined as follows,

$$\ell'(w^T(x^+ - x^-)) = \begin{cases} 0 & \text{if } (w^T(x^+ - x^-)) \geq 1 \\ -1 & \text{if } (w^T(x^+ - x^-)) < 1 \end{cases}$$

In a standard stochastic learning sitting [92], we sample two opposite instances uniformly at random and update the weight vector in each iteration as follows,

$$w_{t+1} = w_t - \frac{1}{(t+t_0)} M g_t(w_t) \quad \text{where} \quad g_t(w_t) = \lambda w_t - \ell'(w_t^T(x_t^+ - x_t^-))(x^+ - x^-),$$

where the rescaling matrix M is defined as $M = \lambda^{-1} I$ when updating the weight vector using only the first-order information. Theoretically, the norm of the optimal weight vector w^* is upper bounded by B [100] $\{||w^*|| \leq B\}$. And because any update step might push the weight vector outside the bound B, the positive constant t_0 is utilized to circumvent the projection step by preventing too large steps in the first few iterations [92], where the best t_0 can be set experimentally.

However, the low complexity of the precedent standard stochastic gradient descent is associated with its slow convergence rate, shown to be sublinear [87], and its low accuracy. Speeding up the converges while maintaining the low complexity of the standard stochastic gradient is our main aim in this study.

In this work, we optimize the pairwise hinge loss function using stochastic gradient descent accelerated by scheduling both the regularization update [92] and averaging techniques [91, 101].

Regulating the regularization step allows the model to discover larger hypothesis classes most of the time during the optimization process, while the averaging step reduces the variance of the weight vector that stems from its stochastic nature. The advantage of scheduling the regularization and averaging steps is to reduce the per-iteration complexity, while effectively accelerating the rate of convergence.

Algorithm 10 describes the accelerated stochastic AUC maximization method. The algorithm randomly selects a positive and negative instance and updates the model in each iteration as follows,

$$w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)} \ell'(w^T(x_t^+ - x_t^-))(x^+ - x^-),$$

where $\ell'(z)$ is the gradient of the hinge loss function, w_t is the solution after t iterations, and $(\lambda(t+t_0))^{-1}$ is the learning rate, which decreases in each iteration. The hyper-parameter λ can be tuned on a validation set. We regulate the regularization update and averaging steps to be performed each $rskip$ and $askip$ iterations respectively as follows,

$$w_{t+1} = w_{t+1} - rskip(t+t_0)^{-1} w_{t+1}$$

$$\tilde{w}_{q+1} = \frac{q\tilde{w}_q + w_{t+1}}{q \, + \, 1},$$

where $\tilde{w}$ is the averaged solution after q iterations with respect to the $askip$. The presented first-order stochastic AUC maximization has linear complexity per iteration with respect to the number of dimensions. Therefore, it is a feasible algorithm to train large-scale datasets. It also can be seen as an averaging variant of the SVMSGD2 method proposed in [92].

6.4 Extension to Approximate Kernel

In many machine learning applications, the data are distributed nonlinearly, which results in a highly nonlinear decision boundary. In these cases, a linear classifier would fail to learn the best

Algorithm 10: Accelerated Stochastic AUC Maximization

```
Input: training dataset $X$, $\lambda$, $t_0$,$T$, $rskip$, $askip$
Set $rcount = rskip$, $acount = askip$, $q = 0$
Initialize $w_1 \leftarrow 0$ and $\tilde{w}_0 \leftarrow 0$
for $t = 1, \ldots, T$ do
    Randomly pick a pair $i_t \in 1, \ldots, n^+, j_t \in 1, \ldots, n^-$
    $x_t = x_{i_t} - x_{j_t}$
    $w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)} \ell'(w_t^T x_t) x_t$
    $rcount = rcount - 1$
    if $rcount \leq 0$ then
        $w_{t+1} = w_{t+1} - rskip(t + t_0)^{-1} w_{t+1}$
        $rcount = rskip$
    end if
    $acount = acount - 1$
    if $acount \leq 0$ then
        $\tilde{w}_{q+1} = \frac{q\tilde{w}_q + w_{t+1}}{q+1}$
        $q = q + 1$
        $acount = askip$
    end if
end for
set $w = \tilde{w}_q$
return $w$
```

model. To address this problem, we can embed the input space into a finite-dimensional space where a complex nonlinear decision boundary can be recognized using a linear classifier. In this work, we investigate both random Fourier features [4] and improved Nyström method [2] to embed the input space into highly but finite dimensional space.

Given an approximate feature map $\varphi(x)$ defined either by random Fourier features or Nyström methods, the objective function of our method can be written in the following form:

$$\min_w \frac{1}{2}||w||^2 + C \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} max(0, 1 - w^T(\varphi(x_i^+) - \varphi(x_j^-))). \quad (6.2)$$

6.5 Experiments for Linear AUC Maximization Methods

In this section, we evaluate the performance of our proposed method on several benchmark datasets. We compare our stochastic AUC maximization method with the state-of-the-art online and stochastic AUC maximization algorithms. We also study the convergence rate of our stochastic AUC maximization algorithm. The experiments are implemented in MATLAB, while the learning algorithms are written in C++ language via MEX files. The experiments were performed on a computer equipped with an Intel 4GHz processor with 32G RAM.

6.5.1 Benchmark Datasets

We use twelve datasets described in Table 6.1. We also experiment on high dimensional datasets described in Table 6.2. The datasets can be downloaded from LibSVM website[8] or UCI[9]. If any of the data is not given as training and test sets, we partition it into 80% for training and 20% for testing. The multi-class data are transformed into imbalanced binary data by grouping roughly half of the classes into a label and the rest of classes into a different label. For sector dataset, we transform the classification task to distinguish between classes whose labels are less than or equal ten from the rest of classes. We also generate sector_v2 that differentiates between classes whose labels less than or equal five and the rest of the classes. For news20 dataset, we use only its training data due to its high dimensionality, and we transform it into imbalanced dataset by grouping the topics of labels less than or equal five into a class and the rest of topics into another class. The version news_v2 distinguishes the topics whose labels greater than 15 from the rest of the topics. We standardize the features of each dataset to have zero mean and unit variance.

[8]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[9]http://archive.ics.uci.edu/ml/index.php

Table 6.1: Description of the benchmark data sets

Data	#training	#test	#feat	ratio
w8a	49,749	14,951	300	32.4
ijcnn1	49,990	91,701	22	9.44
cifar10	50,000	10,000	3072	2.33
connect-4	54,045	13,512	126	3.06
mnist	60,000	10,000	784	2.32
acoustic	78,823	19,705	50	3.31
skin	196,045	49,012	3	3.83
webspam	280,000	70,000	254	1.53
cod-rna	331,152	157,413	8	2.0
epsilon	400,000	100,000	2000	1.00
covtype	464,809	116,203	54	10.65
susy	4,500,000	500,000	18	1.18

Table 6.2: Description of the high dimensional data sets

Data	#training	#test	#feat	ratio
farm-ads	3,314	829	54877	1.14
sector	6,412	3,207	55197	9.13
sector_v2	6,412	3,207	55197	20.0
news20	12,748	3,187	62061	3.01
news20_v2	12,748	3,187	62061	2.98
real-sim	57,847	14,462	20958	2.25

6.5.2 Compared Methods and Model Selection

1. **OAM$_{\text{seq}}$ and OAM$_{\text{gra}}$** The sequential and gradient variants of online AUC maximization [65]. The hyperparameters are chosen as suggested by [65] via 3-fold cross validation. The number of positive and negative buffers is set to 100.

2. **SOLAM:** This is the stochastic online AUC maximization [68]. The hyperparameters of the algorithm (i.e., the learning rate and the bound on the weight vector) are selected via 3-fold cross validation by searching in the grids $\{1 : 9 : 100\}$ and $\{10^{-1}, \ldots, 10^{5}\}$, respectively. The number of iterations is set to 15.

3. **ASAM:** This is our accelerated stochastic AUC maximization algorithm. The hyper-parameter λ is chosen from the grid $\{10^{-10}, \ldots, 10^{-7}\}$ via 3-fold cross validation. For the experiment

with high dimensional data, we tune λ using 3-fold cross validation by searching in the grid $\{1 : 9 : 100\}$.

4. **BAM:** This is the batch AUC maximization algorithm [42]. This algorithm optimizes the squared hinge loss function using truncated Newton. The best regularization hyper-parameter C is chosen from the grid $\{2^{-15}, \ldots, 2^{10}\}$ via 3-fold cross validation.

6.5.3 Results for Linear AUC Algorithms on Benchmark Datasets

The comparison in terms of AUC classification accuracy and training time is shown in Table 6.3. The reported AUC results are the average of 5 runs, except for the results of BAM, which are based on a single run.

We observe that the AUC performance of our proposed method ASAM is on a par with the batch method BAM on most datasets while its training time is shorter than BAM. We also observe that ASAM achieves better AUC classification accuracy than BAM on cifar10, acoustic, and skin datasets, but its AUC is lower than BAM on connect-4 dataset.

We can see that our method ASAM outperforms SOLAM, OAM_{seq}, and OAM_{grd} in terms of AUC classification accuracy. The training time of ASAM is marginally longer than the other online and stochastic methods on most datasets. The method SOLAM shows significantly better AUC performance compared to OAM_{seq} and OAM_{grd}. The efficacy of SOLAM can be ascribed to optimizing a squared loss function and performing multiple passes over the training data.

6.5.4 Results for Linear AUC Algorithms on High Dimensional Datasets

Table 6.4 shows the AUC classification accuracy and training time of the compared algorithms. The reported AUC results are the average of two runs. We can observe that our algorithm ASAM performs better than OAM algorithms on all the datasets. We can also see that the training time of our method ASAM is shorter than those of all methods. We attribute the robust and efficient performance of our ASAM to its neat and straightforward acceleration techniques. Note that scheduling the averaging step avoids many vectors summation steps while retaining the effectiveness of the standard iterate averaging method.

Table 6.3: AUC classification accuracy and training time (in seconds) for linear AUC maximization algorithms on the benchmark data

Algorithm	w8a		ijcnn1		cifar10	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{seq}$	94.379 ± 0.637	1.233	85.034 ± 2.049	0.1182	63.981 ± 4.590	13.248
OAM$_{gra}$	95.339 ± 0.586	1.2274	86.398 ± 3.443	0.1101	64.058 ± 2.875	13.288
SOLAM	91.201 ± 4.222	0.9610	90.494 ± 0.070	0.0722	54.658 ± 6.595	14.664
ASAM	97.742 ± 0.019	1.2813	91.312 ± 0.209	0.2170	76.170 ± 0.098	12.876
BAM	97.876	10.338	91.557	0.5717	74.819	541.16

Algorithm	connect-4		mnist		acoustic	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{seq}$	78.703 ± 1.823	0.6188	92.218 ± 0.921	3.9409	76.303 ± 5.619	0.3696
OAM$_{gra}$	79.491 ± 1.312	0.6046	92.389 ± 0.313	3.9272	76.204 ± 6.561	0.3536
SOLAM	87.450 ± 0.113	0.4295	94.799 ± 0.047	3.0077	86.806 ± 0.130	0.2544
ASAM	87.755 ± 0.060	1.1328	95.909 ± 0.038	4.2642	88.452 ± 0.081	1.0420
BAM	88.197	3.3520	96.053	28.487	87.378	1.8684

Algorithm	skin		webspam		cod-rna	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{seq}$	89.823 ± 6.210	0.1282	94.328 ± 1.473	6.4626	96.710 ± 1.001	0.3482
OAM$_{gra}$	92.899 ± 4.779	0.1138	94.882 ± 0.643	6.2794	96.945 ± 1.330	0.3311
SOLAM	94.698 ± 0.003	0.1293	96.218 ± 0.013	4.5160	98.734 ± 0.007	0.2438
ASAM	95.006 ± 0.189	0.2417	97.230 ± 0.057	9.3542	98.862 ± 0.020	1.7804
BAM	94.794	0.4737	97.289	16.430	98.861	1.9487

Algorithm	epsilon		covtype		susy	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{seq}$	87.511 ± 0.383	67.993	80.931 ± 1.491	2.3906	74.294 ± 3.727	9.0414
OAM$_{gra}$	87.806 ± 0.382	67.284	77.883 ± 4.085	2.2848	72.740 ± 7.592	8.6058
SOLAM	95.957 ± 0.012	53.699	86.876 ± 0.049	1.6590	83.470 ± 0.003	5.6492
ASAM	95.907 ± 0.014	69.410	87.520 ± 0.037	6.7974	85.718 ± 0.086	41.608
BAM	95.967	800.1	87.858	14.916	85.807	64.418

Table 6.4: AUC classification accuracy and training time (in seconds) for linear AUC maximization algorithms on the high dimensional data

Algorithm	farm-ads		sector		sector_v2	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM_{seq}	89.412 ± 0.055	19.396	98.103 ± 0.378	40.164	97.840 ± 0.851	34.507
OAM_{gra}	90.071 ± 0.503	19.515	98.266 ± 0.260	39.332	97.589 ± 0.774	34.126
SOLAM	92.422 ± 0.555	19.375	96.313 ± 0.021	36.863	97.805 ± 0.041	33.252
ASAM	95.701 ± 0.013	12.647	98.927 ± 0.010	24.791	98.326 ± 0.009	28.114

Algorithm	news20		news20_v2		real-sim	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM_{seq}	97.907 ± 0.021	93.996	97.701 ± 0.060	81.000	94.789 ± 0.092	124.14
OAM_{gra}	97.926 ± 0.120	90.831	97.681 ± 0.019	81.846	93.932 ± 0.429	124.20
SOLAM	97.967 ± 0.007	65.868	97.727 ± 0.054	63.530	99.053 ± 0.009	124.23
ASAM	98.715 ± 0.007	51.949	98.682 ± 0.011	61.441	99.612 ± 0.005	111.39

6.5.5 Study on the Convergence Rate

We investigate the convergence of ASAM and its counterpart SOLAM with respect to the number of epochs. We also include SAM algorithm that minimizes the pairwise hinge loss function using SVMSGD2 [92]. The algorithm SAM is analogous to the proposed algorithm ASAM but without the averaging technique. Figure 6.1 depicts the AUC performance of these stochastic methods upon varying the number of epochs. We vary the number of epochs according to the grid $\{1, 2, 3, 4, 5, 10, 30, 60, 100\}$, and run the stochastic algorithms using the same setup described in the preceding subsection. In all subfigures, the x-axis represents the number of epochs, while the y-axis is the AUC classification accuracy averaged over 3 runs on the test set.

We can observe that our ASAM algorithm achieves better AUC classification accuracy from the first epoch compared to SOLAM. We can also see that the AUC performance of ASAM outperforms its non-averaging variant SAM on all the datasets. Furthermore, the AUC performance of ASAM does not fluctuate considerably with varying the number of epochs on all datasets. This stable performance indicates the effectiveness of incorporating the scheduled averaging technique with the scheduled regularization update. It also implies that choosing the best number of epochs would be easy.

We can also see that increasing the number of iterations does not significantly improve the AUC performance of SOLAM on most datasets. The reason might be that SOLAM reaches a saddle point in the first few epochs and gets stuck there.

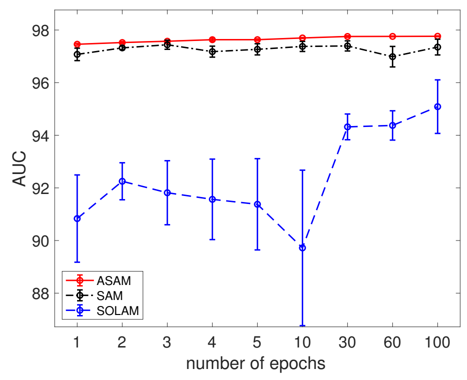
6.6 Experiments for kernelized AUC Maximization Methods

In this section, we evaluate the kernelized variant of our accelerated stochastic AUC maximization algorithm on several datasets. We use the k-means Nyström method to approximate the feature maps. We compare our algorithm with the batch AUC method and the state-of-the-art online and stochastic algorithms. Further, we experimentally study the convergence rate of our algorithm under the high dimensional approximate space.

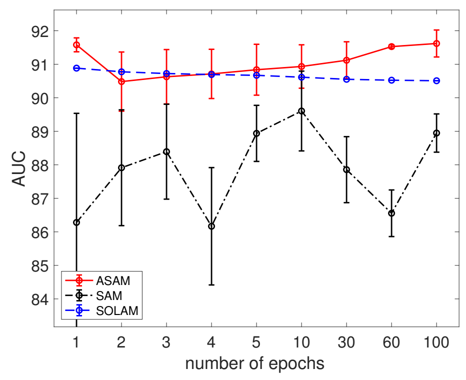
6.6.1 Compared Methods and Model Selection

In this experiment, we use the same methods we implemented in the precedent section, but we apply them on the approximate feature space constructed via k-means Nyström low-rank approximation. We denote these methods as NOAM_{seq}, NOAM_{gra}, NSOLAM, NASAM, and NBAM. Further, we use the same protocol of the precedent section for tuning the hyperparameters of each algorithm. We also include the results of our scalable kernelized batch algorithm NBAM for reference.

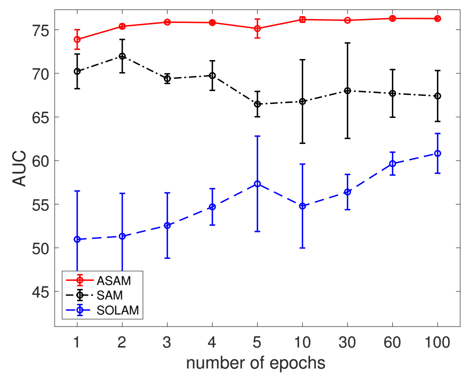
The k-means Nyström approximation is implemented separately for each algorithm. We compute landmark points using the k-means clustering algorithm, which is implemented in C++ language. We set the number of landmark points to be 1600 for all datasets, except for cifar10, epsilon, and susy, which have landmark points set to 3000, 2800, and 400, respectively. We select a Gaussian kernel function to be used with the k-means Nyström approximation. The bandwidth of the Gaussian function is set to be the average square distance between the first 80k instances and the mean, which is computed over these instances.



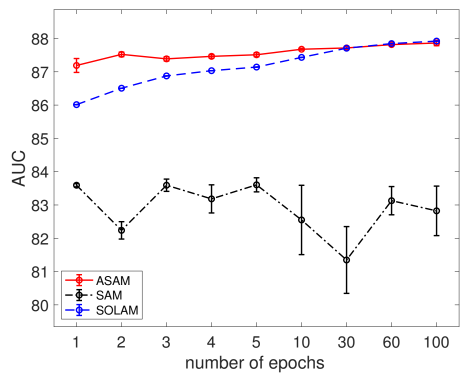
(a) w8a



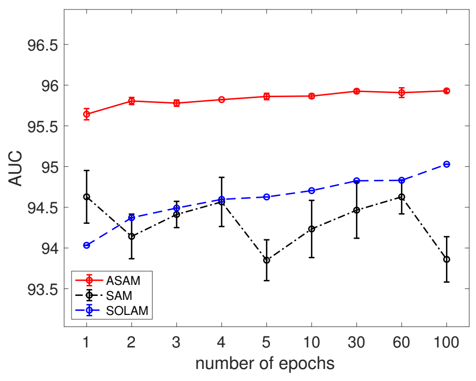
(b) ijcnn1



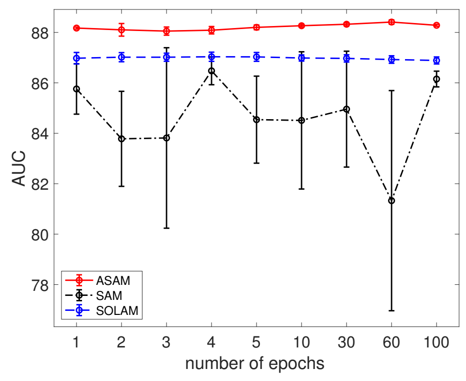
(c) cifar10



(d) connect-4

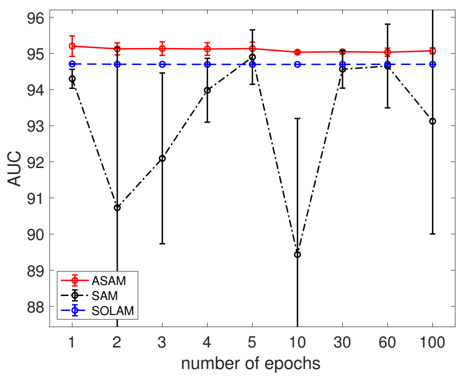


(e) mnist

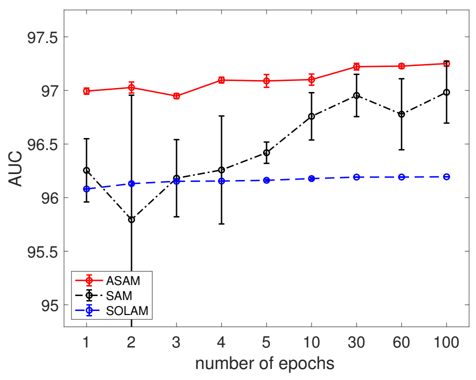


(f) acoustic

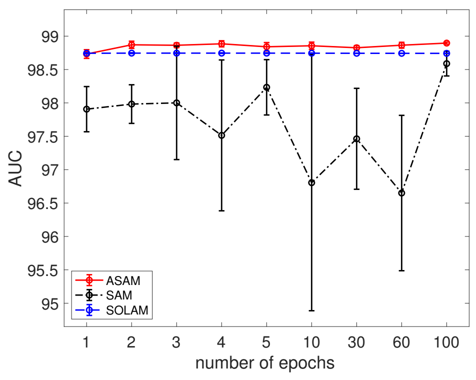
Figure 6.1: AUC classification accuracy of stochastic linear AUC maximization algorithms with respect to the number of epochs. We randomly pick a positive and negative instance in each iteration for ASAM and SAM, where n iterations correspond to one epoch.



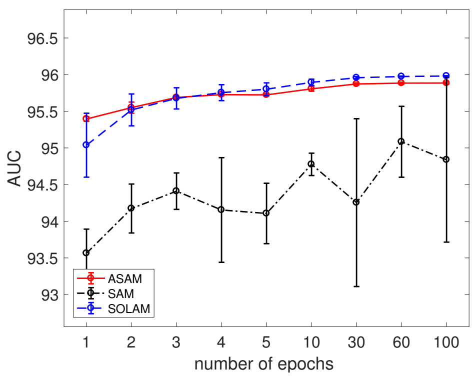
(g) skin



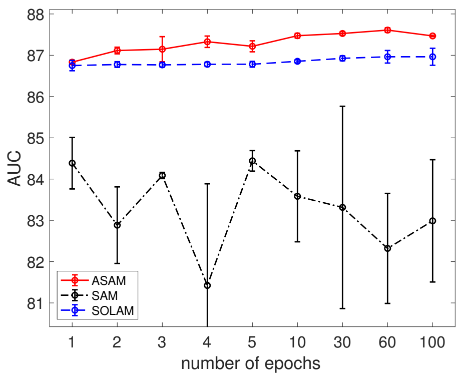
(h) webspam



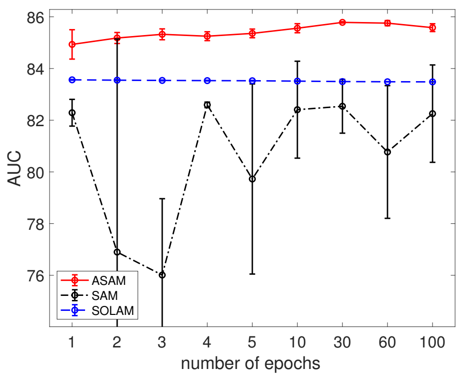
(i) cod-rna



(j) epsilon



(k) covtype



(l) susy

Figure 6.1: AUC classification accuracy of stochastic linear AUC maximization algorithms with respect to the number of epochs. We randomly pick a positive and negative instance in each iteration for ASAM and SAM, where n iterations correspond to one epoch.

6.6.2 Results and Discussion

Table 6.5 shows the comparison of our methods with the other stochastic and online AUC maximization algorithms. The training time in Table 6.5 reports the time cost of learning including the embedding steps. The reported AUC results are averaged over 3 runs.

We can observe that utilizing the kernel approximation improves the AUC classification accuracy of all the algorithms. The proposed NASAM achieves a competitive AUC performance compared to the proposed NBAM. However, NASAM requires shorter training time than NBAM to yield an acceptable AUC accuracy. For example, on the covtype dataset, the AUC performance of NASAM is on par with NBAM, while it only requires 49.17 seconds for training compared to more than 18 minutes required by NBAM. In contrast to the online methods, the proposed NASAM can converge to the optimal solution obtained by the batch method NBAM. We again attribute the robust performance of NASAM to the effectiveness of scheduling both the regularization update and averaging.

We observe that the training time of our proposed NASAM closely matches the training time of those stochastic and online methods. Also, we see that NSOLAM performs better than NOAM methods in terms of AUC classification accuracy. The robust performance of NSOLAM over NOAM implies the advantage of optimizing the pairwise squared hinge loss function, performed by NSOLAM, over the pairwise hinge loss function, carried out by NOAM methods.

6.6.3 Study on the Convergence Rate

In this experiment, we investigate the convergence rate of our method NASAM with respect to the number of epochs. We compare it with NSAM and NSOLAM and follow the same protocol implemented in the previous section for studying the rate of convergence of the stochastic linear methods. For our method NASAM and NSAM, each epoch means n iteration wherein each iteration we randomly pick a positive and negative instance. Figure 6.2 shows the AUC classification accuracy with varying the number epochs. The reported AUC on the test set is averaged over 3 runs.

Table 6.5: AUC classification accuracy and training time (in seconds) for NASAM and the other kernelized AUC maximization algorithms. The reported training time includes the embedding steps.

Algorithm	w8a		ijcnn1		connect-4	
	AUC	Training time	AUC	Training time	AUC	Training time
NOAM$_{\text{seq}}$	96.648 ± 0.387	174.054	99.053 ± 0.165	22.610	85.934 ± 0.217	85.264
NOAM$_{\text{gra}}$	96.351 ± 0.888	173.75	98.764 ± 0.166	22.589	86.004 ± 0.569	84.780
NSOLAM	95.458 ± 0.774	173.90	98.729 ± 0.156	21.262	89.855 ± 0.042	83.029
NASAM	98.547 ± 0.037	173.24	99.692 ± 0.011	22.589	93.602 ± 0.079	85.334
NBAM	98.579	216.95	99.505	96.807	94.069	159.768

Algorithm	mnist		acoustic		skin	
	AUC	Training time	AUC	Training time	AUC	Training time
NOAM$_{\text{seq}}$	98.341 ± 0.206	566.54	88.806 ± 0.667	57.562	99.979 ± 0.002	24.824
NOAM$_{\text{gra}}$	98.078 ± 0.294	565.89	89.743 ± 0.999	56.966	99.960 ± 0.035	24.995
NSOLAM	98.094 ± 0.016	553.33	91.751 ± 0.029	54.223	99.950 ± 0.006	23.029
NASAM	99.375 ± 0.013	566.21	93.804 ± 0.083	56.957	99.982 ± 0.000	26.288
NBAM	99.424	650.79	94.188	176.375	99.978	62.280

Algorithm	webspam		cod-rna		covtype	
	AUC	Training time	AUC	Training time	AUC	Training time
NOAM$_{\text{seq}}$	98.895 ± 0.084	344.72	98.067 ± 0.325	93.295	89.750 ± 0.722	353.78
NOAM$_{\text{gra}}$	98.895 ± 0.325	344.01	98.180 ± 0.135	93.022	89.637 ± 0.577	352.23
NSOLAM	99.065 ± 0.004	339.72	99.104 ± 0.002	83.339	91.842 ± 0.071	335.90
NASAM	99.720 ± 0.005	347.69	99.181 ± 0.007	91.866	96.033 ± 0.088	346.02
NBAM	99.726	598.17	99.184	467.80	96.765	2564.9

The results show that NASAM is able to achieve a better AUC classification accuracy from the first run, while the other algorithms require multiple epochs. We can also see the stability of NASAM over the different number of epochs compared to the other algorithms.

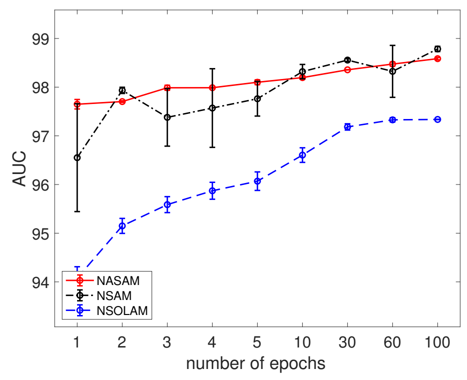
6.7 Experiments of NASAM vs. FASAM

In this experiment, we investigate the performance of our accelerated stochastic AUC maximization algorithm (ASAM) with approximate feature space constructed via the k-means Nyström low-rank matrix approximation and random Fourier features. The former is a data-dependent approximation, while the later is data-independent as it approximates the kernel function. We name our accelerated stochastic AUC maximization algorithm applied to random features as FASAM, while NASAM refers to our accelerated algorithm applied to Nyström approximation.

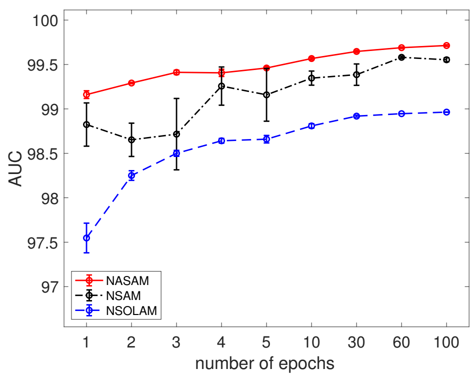
We use these approximation methods to approximate a Gaussian kernel. The bandwidth of the Gaussian kernel for Nyström approximation is set to be the average square distance between the first 80k instances and the mean, which is computed over these instances. For Fourier features, the bandwidth of the kernel is set to $\gamma = 2/\sigma$, where σ is tuned by 3-fold cross validation via the grid $\{10, 20, 50, 100\}$.

Table 6.6 shows the AUC performance and training time of FASAM NASAM on seven benchmark datasets. The number of features is set to 1600 for both Nyström and Fourier features methods. We can observe that the NASAM outperforms FASAM in terms of AUC classification accuracy. This indicates that data-dependent approach is more robust than data-independent in approximating the kernel. However, this is obtained at the expense of the computational time. The computational time of k-means Nyström approximation can be reduced by restricting the computation of the landmark points on a subset of the dataset.

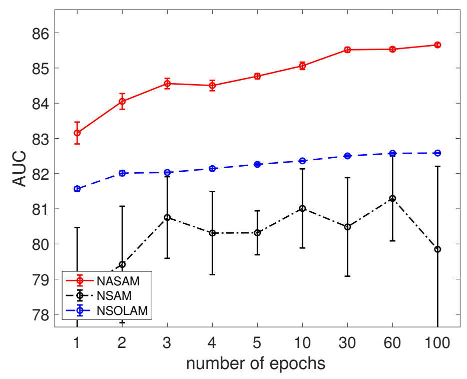
We study the AUC performance of NASAM and FASAM with a different number of features. The results are shown in Figure 6.3. We can observe that increasing the number of features improves the AUC performance. This indicates that transforming the input space into a feature space



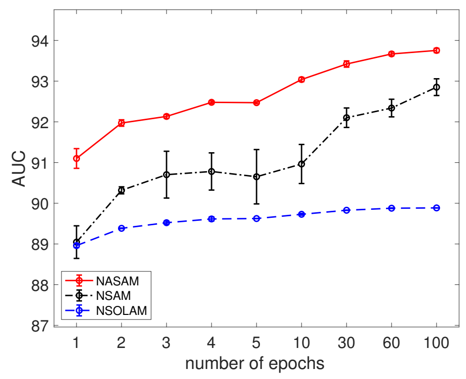
(a) w8a



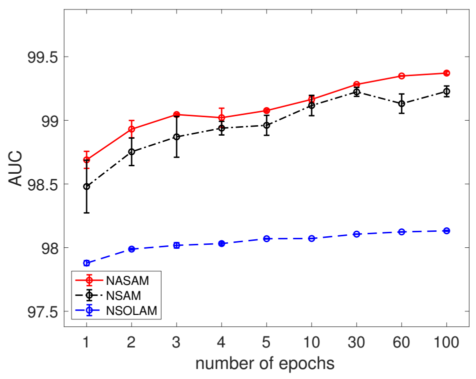
(b) ijcnn1



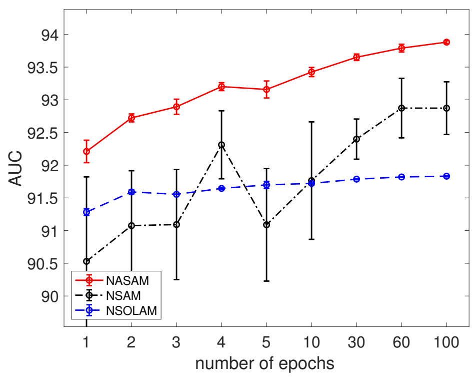
(c) cifar10



(d) connect-4

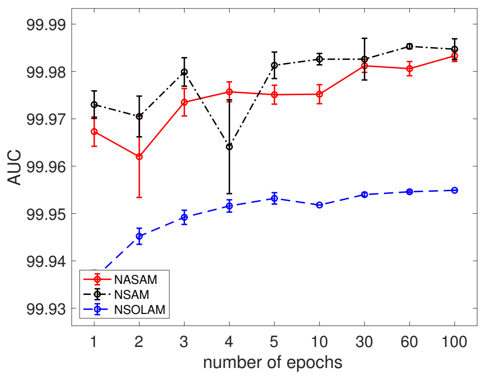


(e) mnist

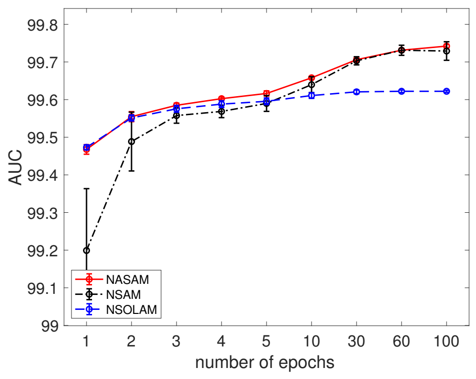


(f) acoustic

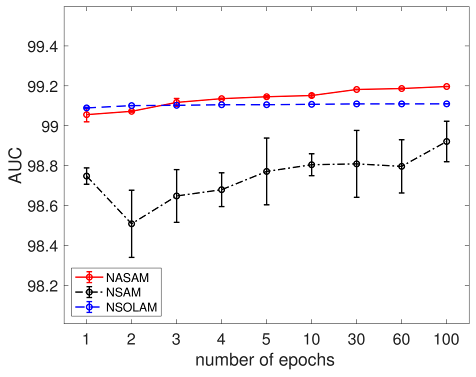
Figure 6.2: AUC classification accuracy of the kernelized stochastic AUC algorithms with respect to the number of epochs. We use Nyström approximation to approximate the kernel matrix. We randomly pick a positive and negative instance in each iteration for NASAM and NSAM, where n iterations correspond to one epoch.



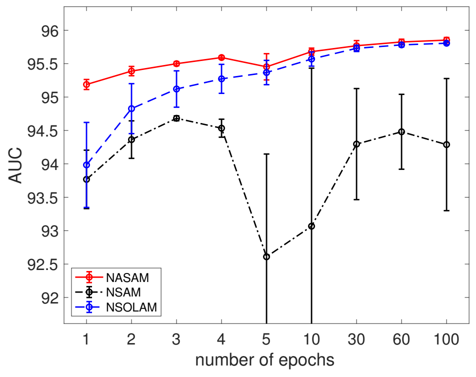
(g) skin



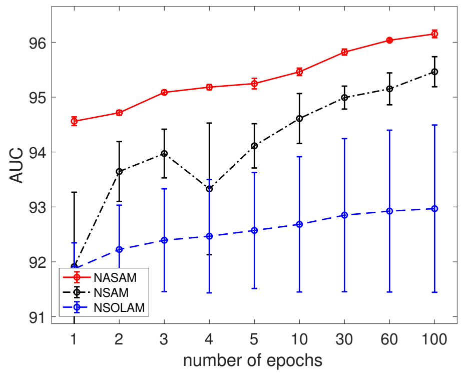
(h) webspam



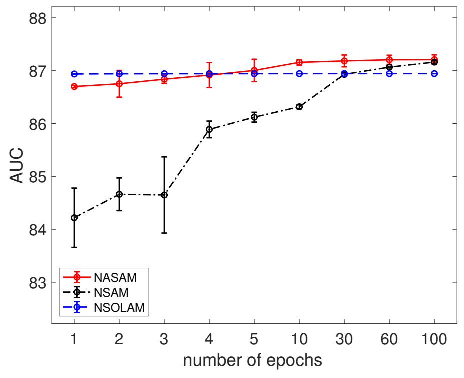
(i) cod-rna



(j) epsilon



(k) covtype



(l) susy

Figure 6.2: AUC classification accuracy of the kernelized stochastic AUC algorithms with respect to the number of epochs. We use Nyström approximation to approximate the kernel matrix. We randomly pick a positive and negative instance in each iteration for NASAM and NSAM, where n iterations correspond to one epoch.

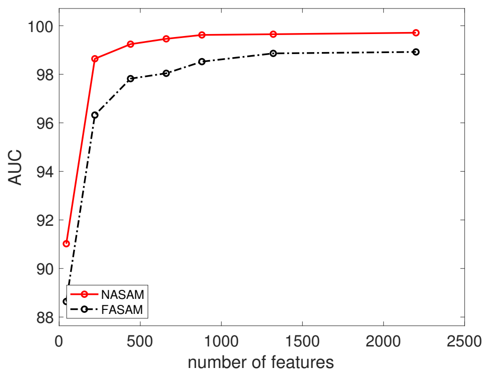
boosts the AUC performance. We can also see that the AUC performance of NASAM surpasses that of FASAM. This can be attributed to the superiority of the data-dependent approximation.

Table 6.6: Comparison of NASAM vs. FASAM in terms of AUC performance (%) and training time (in seconds)

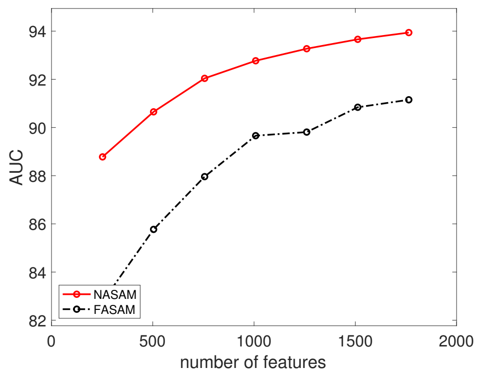
Data	Metric	**FASAM**	**NASAM**
ijcnn1	AUC	98.70 ± 0.0038	99.69 ± 0.0000
	Training time	6.77	21.80
connect-4	AUC	90.71 ± 0.0008	93.65 ± 0.0001
	Training time	8.10	85.06
acoustic	AUC	91.95 ± 0.0012	93.80 ± 0.0002
	Training time	11.33	55.99
skin	AUC	99.97 ± 0.0000	99.98 ± 0.0000
	Training time	27.31	26.46
cod-rna	AUC	99.13 ± 0.0000	99.18 ± 0.0000
	Training time	46.70	91.47
covtype	AUC	96.20 ± 0.0041	96.00 ± 0.0003
	Training time	66.73	342.2
webspam	AUC	98.90 ± 0.0001	99.71 ± 0.0000
	Training time	40.06	341.1

6.8 Conclusion

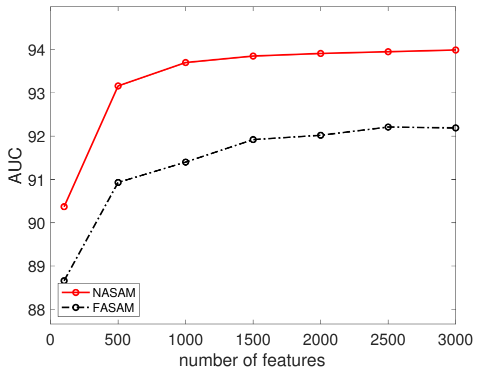
In this chapter, we have developed a fast convergence stochastic AUC maximization algorithm that solves a first-order pairwise objective function. The acceleration technique is based on scheduling the regularization and the averaging steps. Experimentally, we show that our accelerated stochastic algorithm can achieve a competitive AUC accuracy compared to the batch algorithm, which optimizes a second-order objective function. We also show that our algorithm is able to surpass the state-of-the-art stochastic and online AUC maximization methods with a marginal increase in the training time. Further, we extend our algorithm to handle nonlinear decision boundaries by approximating the feature maps via the k-means Nyström and random Fourier methods. For future work, we plan to improve the convergence rate of our accelerated stochastic AUC maximization by utilizing different acceleration techniques such as a proximal operator [102].



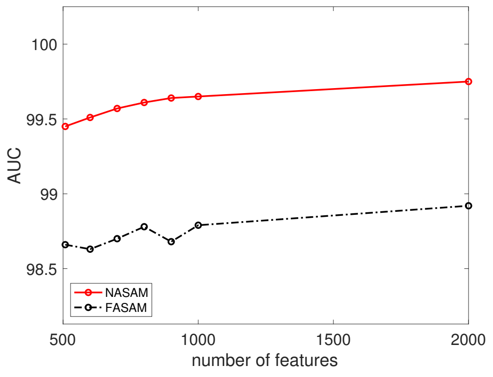
(a) ijcnn1



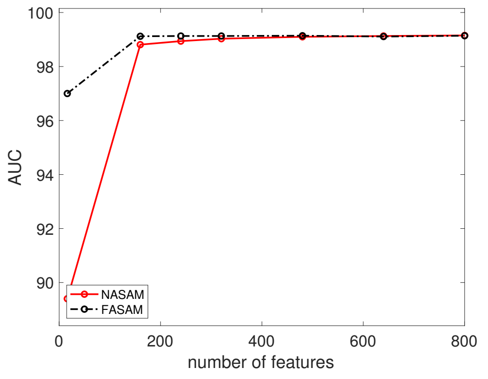
(b) connect-4



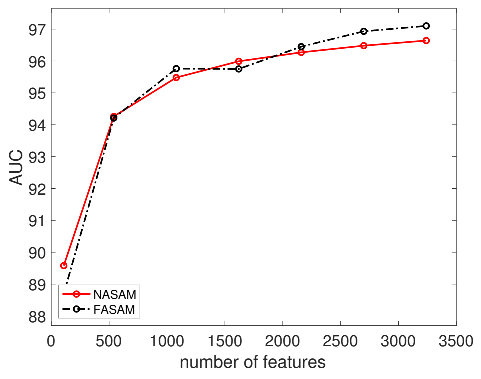
(c) acoustic



(d) webspam



(e) cod-rna



(f) covtype

Figure 6.3: AUC classification accuracy of NASAM and FASAM with a different number of features

Chapter 7

Proximal Stochastic AUC Maximization Algorithm

7.1 Introduction

Stochastic and online AUC maximization algorithms have been demonstrated successful optimization techniques for large-scale settings. In contrast to batch AUC maximization methods, the stochastic and online learning algorithms are capable of handling sizable data and dealing efficiently with data arrive in a sequential manner. However, the batch methods converge to the optimal solution, while the stochastic and online methods suffer from suboptimal convergence. The ability to reach the optimal solution makes batch methods achieve a higher accuracy compared to the stochastic and online algorithms. However, the per-iteration complexity of the batch methods is time and memory consuming.

The primary challenge here is to develop stochastic AUC algorithms that are able to achieve AUC classification accuracy obtained by the batch methods while maintaining low training complexity. The accelerated stochastic AUC maximization algorithm [103] shows a strong AUC performance at the same level as a batch method. However, with complex data, the accelerated stochastic AUC maximization algorithm requires a large number of iterations to achieve AUC classification accuracy comparable to the batch method.

In this work, we develop a proximal stochastic AUC maximization algorithm. The proposed proximal stochastic AUC maximization algorithm can be applied to a non-smooth regularization term. Our method uses the proximal mapping of the hinge loss function to improve the convergence rate of our stochastic AUC maximization algorithm. We experimentally verify the efficiency and the efficacy of the proposed proximal stochastic AUC maximization algorithm on several benchmark datasets.

7.2 Related Work

Several learning approaches have been developed to optimize the AUC objective function efficiently. First-order [65, 66] and second-order [82] online algorithms are devised to optimize a pairwise convex surrogate loss function by utilizing different buffering schemes to handle the pairwise property. Another second-order approach [66] circumvent the need to buffer some instances by maintaining a covariance matrix for each class, and it learns the model based on the first and second moments. Among these online methods, the second-order algorithm [82] based on the soft confidence-weighted learning shows a robust AUC classification accuracy. However, its learning complexity of $\mathcal{O}(Bd^2)$ makes it inefficient for high dimensional data, where B is the size of the buffer and d is the number of features.

Recently, the work [68] finds that optimizing a pairwise least square loss function is equivalent to min-max saddle point problem. Therefore, a stochastic online AUC maximization algorithm is devised [68] based on minimizing the primal and maximizing the dual variables in a stochastic manner. Building on the saddle point formulation for the AUC maximization, the work [99] proposes a proximal stochastic algorithm, while [98] proposes an adaptive multi-stage algorithm. These algorithms [99, 98] are shown to achieve a convergence rate of $\mathcal{O}(\frac{1}{t})$, which is faster than the convergence rate $\mathcal{O}(\frac{1}{\sqrt{t}})$ of the standard saddle point algorithm for AUC maximization, where t is the number of iteration or the number of instances.

We should point out that our proposed algorithm is similar to the proximal stochastic AUC maximization [99] in terms of utilizing the proximal operator. However, the proximal operator is used by [99] as a regularizer for the model updated based on the saddle point formulation, while we apply the proximal operator to the hinge loss function optimized by accelerated stochastic gradient descent. The proximal stochastic algorithm proposed in [104] is also similar to our proposed method. However, the proximal stochastic algorithm [104] is designed to maximize the accuracy, whereas our proximal stochastic algorithm minimizes a pairwise loss function for AUC maximization.

7.3 Proximal Algorithm

Given a sequence of training instances $(x_1, y_1), \ldots, (x_n, y_n)$ independently drawn from unknown distribution $\mathcal{D}$ on $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ represents an instance with d dimensional features and $y \in \{1, -1\}$ represents the label. Let $h(x) = (w^T x)$ denotes a linear classifier, then the AUC score is defined as:

$$AUC(w) = Pr(h(x^+) \geq h(x^-)) = \mathbb{E}[\mathbb{I}_{h(x^+) \geq h(x^-)}],$$

where $\mathbb{I}(\cdot)$ is an indicator function. In practice, the indicator function, which is discontinuous, is replaced by a convex surrogate loss function. In this work, we define the AUC loss function by the hinge loss $max(0, 1 - h(x^+ - x^-))$. The optimization problem for maximizing the AUC objective function is defined as:

$$\min_{w \in \mathbb{R}^d} F(w) \triangleq \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} f(w) + \lambda \psi(w), \tag{7.1}$$

where $f(w) = \ell(w^T(x_i^+ - x_j^-)) = max\{0, 1 - w^T(x_i^+ - x_j^-)\}$ is a convex differentiable function and $\psi(w)$ is a convex regularizer, which could be non-differentiable. The popular and scalable approach to solve such an optimization problem is stochastic gradient descent (SGD) [86], which enjoys a low per-iteration complexity. However, the convergence rate of the vanilla SGD is slower than that of the gradient method. The accelerated stochastic AUC maximization algorithm [103] improves the convergence rate by combining both the scheduled regularization and the scheduled iterate averaging techniques.

In some cases with complex datasets, this algorithm [103] turns out to be inefficient in terms of iteration complexity, meaning a large number of iterations is required to achieve an AUC performance comparable to the batch AUC method. How to make a first-order SGD to generalize as better as a batch method from the first few iterations is a challenging problem.

In this work we promote the convergence rate of the accelerated stochastic AUC maximization via using the proximal mapping of the hinge loss function. The minimization of the proximal variant of the objective function 7.1 using a stochastic algorithm comprises of drawing a random positive and negative instance at each iteration and compute the model as,

$$w_{t+1} = w_t - \frac{1}{(t+t_0)} \; M \; \text{prox}_{\lambda t}(w_t),$$

where the rescaling matrix M is defined as $M = \lambda^{-1} I$ when updating the weight vector using only the first-order information. The proposed algorithm is detailed in Algorithm 11. The main step in our algorithm is the use of the proximal mapping of the pairwise hinge loss function. The operator of the proximal mapping of $f(w_t)$ is defined as:

$$\text{prox}_{\lambda t}(w) = \underset{v \in \mathbb{R}^d}{\text{argmin}} \Big\{ \lambda f_t(v) + \frac{1}{2} ||v - w||^2 \Big\}. \tag{7.2}$$

The solution of the proximal operator 7.2 can be derived analytically using its optimality condition. The derivation steps are detailed in Appendix A. The proposed proximal algorithm applies the scheduled regularization and averaging steps [103] to the weights of the model to speed up the convergence. These two steps are regulated to be performed each $rskip$ and $askip$ iterations respectively as follows,

$$w_{t+1} = w_{t+1} - rskip(t+t_0)^{-1} w_{t+1}$$

$$\tilde{w}_{q+1} = \frac{q\tilde{w}_q + w_{t+1}}{q \; + \; 1},$$

where $\tilde{w}$ is the averaged solution after q iterations with respect to the $askip$.

To show the difference between the proximal stochastic and the standard stochastic gradient methods for AUC maximization, we can rewrite the update step of our proximal algorithm as:

$$w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)} \; g(w_{t+1}),$$

whereas the update step of the vanilla stochastic algorithm is written as:

$$w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)}\, g(w_t).$$

We can see that the proximal stochastic algorithm can evaluate the hinge loss function at w_{t+1} instead of w_t without making an actual iteration. Moreover, the proximal operator makes small update steps during training iterations. In contrast to the accelerated stochastic method [103], our proximal algorithm averages a set of weights that received a small changing during training. The averaging of such weights yields an improvement in accelerating the convergence rate.

Algorithm 11: Proximal Stochastic AUC Maximization

```
Input: training dataset $X$, $\gamma$,t$_0$,$T$, $rskip$, $askip$
Set $rcount = rskip$, $acount = askip$, $q = 0$
Initialize $w_1 = 0 \in \mathbf{R}^d$ and $\tilde{w}_0 = 0 \in \mathbf{R}^d$
for $t = 1, \ldots, T$ do
  Randomly pick a pair $i_t \in 1, \ldots, n^+, j_t \in 1, \ldots, n^-$
  $x_t = x_{i_t} - x_{j_t}$
  $\lambda_t = \frac{1}{\gamma(t+t_0)}$
  $w_{t+1} = \text{prox}_{\lambda_t f_t}(w_t)$
  $rcount = rcount - 1$
  if $rcount \leq 0$ then
    $w_{t+1} = w_{t+1} - rskip\ (t+t_0)^{-1}\ w_{t+1}$
    $rcount = rskip$
  end if
  $acount = acount - 1$
  if $acount \leq 0$ then
    $\tilde{w}_{q+1} = \frac{q\tilde{w}_q + w_{t+1}}{q+1}$
    $q = q + 1$
    $acount = askip$
  end if
end for
set $w = \tilde{w}_q$
return $w$
```

7.4 Experiments

In this section, we evaluate the performance of our proposed method on several benchmark datasets. We compare our proximal stochastic AUC maximization algorithm with the state-of-the-art online and stochastic AUC maximization algorithms. The experiments are implemented in MATLAB, while the learning algorithms are written in C++ language via MEX files. The experiments were performed on a computer equipped with an Intel 4GHz processor with 32G RAM.

7.4.1 Datasets

We use several datasets described in Table 7.1. We also experiments on high dimensional datasets described in Table 7.2. The datasets can be downloaded from LibSVM website[10] and UCI[11]. For datasets that are not split into training and test sets, we partition them into 80% for training and 20% for testing. The multi-class data are transformed into imbalanced binary data by grouping roughly half of the classes into a label and the rest of classes into a different label.

Table 7.1: Benchmark data sets

Data	#training	#test	#feat	ratio
spambase	3,680	921	57	1.53
letter	15,000	5,000	16	2.07
a9a	26,048	6,513	123	3.15
w8a	49,749	14,951	300	32.4
ijcnn1	49,990	91,701	22	9.44
cifar10	50,000	10,000	3072	2.33
connect-4	54,045	13,512	126	3.06
mnist	60,000	10,000	784	2.32
acoustic	78,823	19,705	50	3.31
aloi	86,400	21,600	128	46.61
webspam	280,000	70,000	254	1.53
cod-rna	331,152	157,413	8	2.0
epsilon	400,000	100,000	2000	1.00
covtype	464,809	116,203	54	11.26
susy	4,500,000	500,000	18	1.18

[10]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[11]http://archive.ics.uci.edu/ml/index.php

Table 7.2: High dimensional data sets

Data	#training	#test	#feat	ratio
farm-ads	3,314	829	54,877	1.14
sector	6,412	3,207	55,197	9.13
news20	12,748	3,187	62,061	3.01
real-sim	57,847	14,462	20,958	2.25

7.4.2 Compared Methods and Model Selection

1. **OAM$_{\text{seq}}$ and OAM$_{\text{gra}}$** [65]: The sequential and gradient variants of online AUC maximization. The hyperparameters are chosen as suggested by [65] via 3-fold cross validation. The number of positive and negative buffers is set to 100.

2. **CBR$_{\text{FIFO}}$** [82]: The confidence-weighted bipartite ranking algorithms with the First-In-First-Out buffer updating policy. The size of the positive and negative buffers is fixed at 50. The hyperparameter η is set to 0.7, and the penalty hyperparameter C is tuned by 3-fold cross validation by searching in $2^{[-10:10]}$. We use the diagonal variant when experimenting on the high dimensional datasets.

3. **SOLAM** [68]: This is the stochastic online AUC maximization. The hyperparameters of the algorithm (i.e., the learning rate and the bound on the weight vector) are selected via 3-fold cross validation by searching in the grids $\{1 : 9 : 100\}$ and $\{10^{-1}, \ldots, 10^{5}\}$, respectively. The number of iterations is set to 15.

4. **BAM** [42]: This is the batch AUC maximization algorithm. This algorithm optimizes the squared hinge loss function using truncated Newton. The best regularization hyperparameter C is chosen from the grid $\{2^{-15}, \ldots, 2^{10}\}$ via 3-fold cross validation.

5. **ASAM** [103]: This is the accelerated stochastic AUC maximization algorithm. The hyperparameter λ is chosen from the grid $\{10^{-10}, \ldots, 10^{-7}\}$ via 3-fold cross validation. For the

experiment with high dimensional data, we tune λ using 3-fold cross validation by searching in the grid $\{1 : 9 : 100\}$.

6. **PSAM**: This is the proposed proximal stochastic AUC maximization algorithm. The hyper-parameter λ is chosen from the grid $\{10^{-10}, \ldots, 10^{-7}\}$ via 3-fold cross validation. For the experiment with high dimensional data, we tune λ using 3-fold cross validation by searching in the grid $\{1 : 9 : 100\}$.

7.4.3 Results and Discussion

Results for Linear AUC Maximization Methods:

The comparison in terms of AUC performance and training time on the benchmark datasets is shown in Table 7.3, while the comparison on the high dimensional datasets are shown Table 7.4. The reported AUC is averaged over 3 runs for experiments on the high dimensional datasets; otherwise, the AUC results are averaged over 5 runs.

We observe that our algorithm PSAM outperforms the other online and stochastic methods in terms of AUC classification accuracy. Further, we see that the AUC performance of PSAM is comparable to the batch method, whereas the training of PSAM is faster than the batch method. PSAM is also able to achieve better AUC classification accuracy compared to its non-proximal counterpart ASAM, while its training time is on par with that of ASAM. $\mathrm{CBR}_{\mathrm{FIFO}}$ achieves a robust AUC performance on most datasets. However, its training is significantly slower than the other stochastic and online algorithms on most datasets, especially for datasets with a large number of features.

Results for Nonlinear AUC Maximization Methods:

We compare the performance of the nonlinear variant of our proximal method with the other nonlinear batch and stochastic AUC maximization algorithms on six datasets (i.e., acoustic, aloi, cod-rna, webspam, covtype, and susy). The results comparing the performance are shown in Table 7.5. The Gaussian kernel matrix is approximated using the k-means Nyström method. The bandwidth of the Gaussian function is set to be the average square distance between the first 80k instances and the mean, which is computed over these instances. We set the number of landmark points to be 1600 for acoustic, aloi, cod-rna, webspam, and covtype, while susy has landmark points set to 400. The results of the stochastic methods are averaged over 3 runs.

We can see that our method NPSAM achieves a robust performance compared to NSOLAM and NASAM, while its AUC performance is on par with that of the batch method NBAM. However, the training time of our method NPSAM is significantly shorter than NBAM. Among the stochastic algorithms, NSOLAM performs poorly compared to our methods. The robust performance and the fast training of our proximal algorithm make it appealing for large-scale applications.

7.4.4 Study on the Convergence Rate

We study the convergence of PSAM with respect to the number of epochs. We also compare it with the other stochastic AUC maximization methods ASAM and SOLAM. The AUC results of these stochastic methods upon varying the number of epochs are depicted in Figure 7.1. The results of the nonlinear variants are shown in 7.2. We vary the number of epochs according to the grid $\{1, 2, 3, 4, 5, 10, 30, 60\}$, and run the stochastic algorithms using the same setup described in the preceding subsection. One epoch means n number of iterations, where n is the number of instances. For PSAM and ASAM, we pick a positive and negative instance at random in each iteration. In all subfigures, the x-axis represents the number of epochs, while the y-axis is the AUC classification accuracy averaged over 3 runs on the test set.

We observe that PSAM and its nonlinear variant NPSAM are able to reach the optimal solution from the first epoch in most datasets. We attribute this superior performance of our algorithms to the formulation of the proximal operator with scheduling both the regularization and the averaging steps. We also note that increasing the number of epochs improve the AUC performance of PSAM and NPSAM on some datasets. Notice that the number of iterations in the first epoch is much smaller than the number of pairs. This suggests that studying different sampling strategies is a possible research direction to boost the rate of convergence.

Table 7.3: Comparison of AUC classification accuracy and training time (in seconds) for different AUC maximization algorithms

Algorithm	spambase		letter		a9a	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	96.236 $\pm$ 0.473	0.018	65.552 $\pm$ 1.839	0.036	81.565 $\pm$ 0.869	0.286
OAM$_{\text{gra}}$	95.995 $\pm$ 0.913	0.017	66.759 $\pm$ 1.057	0.029	81.456 $\pm$ 1.878	0.282
CBR$_{\text{FIFO}}$	97.573 $\pm$ 0.093	0.533	68.173 $\pm$ 0.122	0.446	89.900 $\pm$ 0.013	17.25
SOLAM	94.204 $\pm$ 0.143	0.017	68.514 $\pm$ 0.158	0.015	89.540 $\pm$ 0.047	0.225
ASAM	97.356 $\pm$ 0.100	0.011	68.209 $\pm$ 0.286	0.017	89.637 $\pm$ 0.104	0.287
PSAM	97.508 $\pm$ 0.143	0.015	69.043 $\pm$ 0.047	0.024	90.111 $\pm$ 0.011	0.367
BAM	97.72	0.110	68.51	0.223	90.43	1.7459

Algorithm	w8a		ijcnn1		cifar10	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	94.003 $\pm$ 1.104	1.225	87.498 $\pm$ 1.282	0.113	64.440 $\pm$ 2.303	13.31
OAM$_{\text{gra}}$	94.815 $\pm$ 1.334	1.214	86.617 $\pm$ 1.850	0.113	64.561 $\pm$ 2.154	13.37
CBR$_{\text{FIFO}}$	97.442 $\pm$ 0.455	5.149	91.591 $\pm$ 0.048	2.224	66.942 $\pm$ 1.060	146.2
SOLAM	94.537 $\pm$ 0.881	0.973	90.527 $\pm$ 0.087	0.071	57.193 $\pm$ 4.620	14.63
ASAM	97.695 $\pm$ 0.018	0.681	91.503 $\pm$ 0.197	0.116	76.391 $\pm$ 0.089	6.881
PSAM	97.747 $\pm$ 0.026	0.920	92.218 $\pm$ 0.024	0.188	76.056 $\pm$ 0.035	9.113
BAM	97.88	10.07	91.56	0.570	74.53	809.8

Algorithm	connect-4		mnist		acoustic	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	79.737 $\pm$ 0.179	0.618	92.176 $\pm$ 0.748	3.928	82.116 $\pm$ 2.264	0.368
OAM$_{\text{gra}}$	78.501 $\pm$ 1.504	0.602	92.097 $\pm$ 0.656	3.950	78.000 $\pm$ 8.433	0.354
CBR$_{\text{FIFO}}$	88.151 $\pm$ 0.029	37.75	95.753 $\pm$ 0.119	17.24	88.573 $\pm$ 0.076	11.251
SOLAM	87.491 $\pm$ 0.062	0.433	94.866 $\pm$ 0.046	3.024	87.083 $\pm$ 0.177	0.250
ASAM	87.771 $\pm$ 0.057	0.701	95.911 $\pm$ 0.047	2.164	88.393 $\pm$ 0.024	0.543
PSAM	88.200 $\pm$ 0.005	0.851	96.051 $\pm$ 0.018	2.768	88.557 $\pm$ 0.010	0.672
BAM	88.20	3.429	96.05	27.74	87.38	1.880

Algorithm	webspam		cod-rna		epsilon	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	95.432 $\pm$ 0.399	6.571	94.956 $\pm$ 3.567	0.343	88.201 $\pm$ 0.412	67.44
OAM$_{\text{gra}}$	95.331 $\pm$ 0.334	6.356	97.632 $\pm$ 0.121	0.332	87.375 $\pm$ 0.614	68.19
CBR$_{\text{FIFO}}$	97.234 $\pm$ 0.024	37.75	98.893 $\pm$ 0.003	5.913	95.591 $\pm$ 0.074	689.4
SOLAM	96.615 $\pm$ 0.025	4.536	98.770 $\pm$ 0.006	0.250	95.961 $\pm$ 0.006	53.50
ASAM	97.197 $\pm$ 0.026	4.870	98.900 $\pm$ 0.012	0.896	95.814 $\pm$ 0.031	37.46
PSAM	97.250 $\pm$ 0.004	6.289	98.687 $\pm$ 0.002	1.203	95.964 $\pm$ 0.001	50.35
BAM	97.37	17.56	98.86	2.046	95.97	837.8

Algorithm	covtype		susy		aloi	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	78.683 $\pm$ 1.952	2.378	71.957 $\pm$ 0.669	8.973	73.226 $\pm$ 1.521	1.088
OAM$_{\text{gra}}$	80.760 $\pm$ 1.613	2.281	69.810 $\pm$ 7.131	8.590	74.128 $\pm$ 2.219	1.007
CBR$_{\text{FIFO}}$	86.760 $\pm$ 0.895	72.60	85.953 $\pm$ 0.001	202.7	81.576 $\pm$ 0.243	80.14
SOLAM	86.425 $\pm$ 0.114	1.662	83.525 $\pm$ 0.015	5.723	73.846 $\pm$ 1.636	0.719
ASAM	86.851 $\pm$ 0.048	4.183	85.820 $\pm$ 0.060	21.06	80.311 $\pm$ 0.145	0.789
PSAM	87.059 $\pm$ 0.001	5.344	85.950 $\pm$ 0.001	28.44	80.993 $\pm$ 0.020	1.248
BAM	87.18	15.02	85.81	63.75	81.64	12.36

Table 7.4: Comparison of AUC classification accuracy and training time (in seconds) for different AUC maximization algorithms on the high dimensional datasets

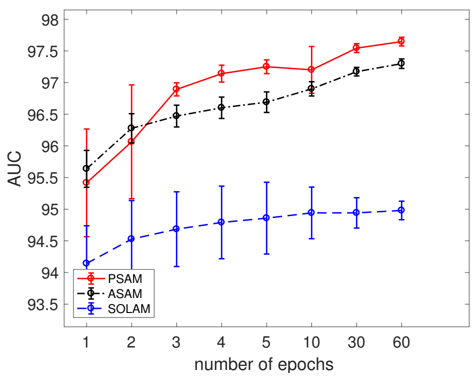
Algorithm	farm-ads		sector		news20	
	AUC	Training time	AUC	Training time	AUC	Training time
OAM$_{\text{seq}}$	89.260 ± 0.091	19.43	98.258 ± 0.240	39.108	97.610 ± 0.100	90.90
OAM$_{\text{gra}}$	90.080 ± 0.397	20.15	98.161 ± 0.206	37.94	97.647 ± 0.046	89.40
CBR$_{\text{FIFO}}$	93.959 ± 1.572	226.8	98.720 ± 0.245	541.6	98.619 ± 0.074	1576
SOLAM	91.738 ± 0.385	19.39	96.836 ± 0.186	29.39	97.605 ± 0.021	65.73
ASAM	95.690 ± 0.071	7.877	97.719 ± 0.039	20.537	97.904 ± 0.006	36.67
PSAM	95.839 ± 0.037	12.45	98.875 ± 0.013	25.03	98.182 ± 0.395	56.73

Algorithm	real-sim	
	AUC	Training time
OAM$_{\text{seq}}$	94.533 ± 0.081	123.0
OAM$_{\text{gra}}$	94.331 ± 0.355	123.3
CBR$_{\text{FIFO}}$	99.553 ± 0.011	1355
SOLAM	99.006 ± 0.017	123.0
ASAM	99.513 ± 0.003	80.38
PSAM	99.652 ± 0.008	97.47

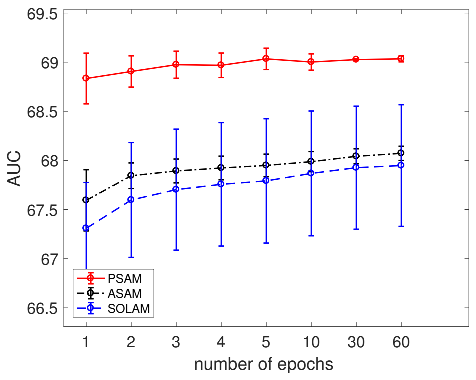
Table 7.5: Comparison of AUC classification accuracy and training time (in seconds) for the nonlinear variants of the batch and the stochastic AUC maximization algorithms. The training time excludes the embedding steps.

Algorithm	acoustic		aloi		cod-rna	
	AUC	Training time	AUC	Training time	AUC	Training time
NSOLAM	92.826 ± 0.279	8.25	92.450 ± 0.212	8.99	99.108 ± 0.000	34.93
NASAM	93.316 ± 0.063	5.61	98.992 ± 0.025	5.75	99.163 ± 0.007	23.65
NPSAM	94.073 ± 0.028	7.36	99.507 ± 0.002	8.22	99.195 ± 0.000	31.15
NBAM	94.173	142.4	99.742	171.8	99.182	419.5

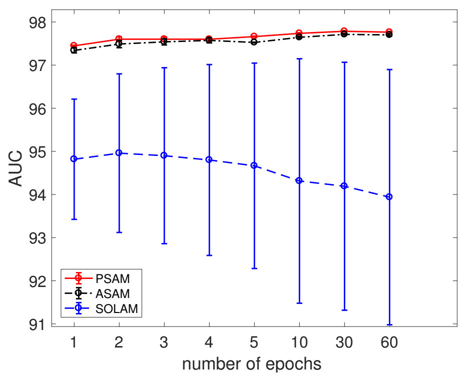
Algorithm	webspam		covtype		susy	
	AUC	Training time	AUC	Training time	AUC	Training time
NSOLAM	99.594 ± 0.001	29.34	94.324 ± 0.241	49.07	86.933 ± 0.000	303.44
NASAM	99.629 ± 0.014	19.45	95.201 ± 0.081	34.26	87.200 ± 0.024	603.08
NPSAM	99.759 ± 0.000	26.74	96.150 ± 0.021	45.40	87.301 ± 0.006	589.48
NBAM	99.740	295.65	96.650	2025.1	87.280	5512.3



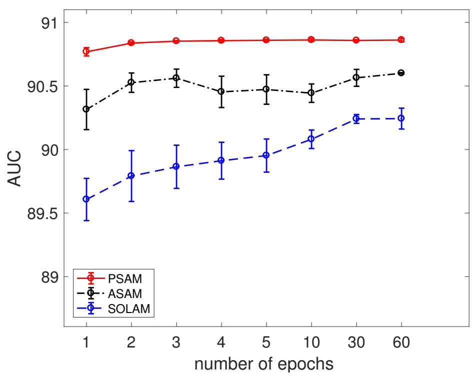
(a) spambase



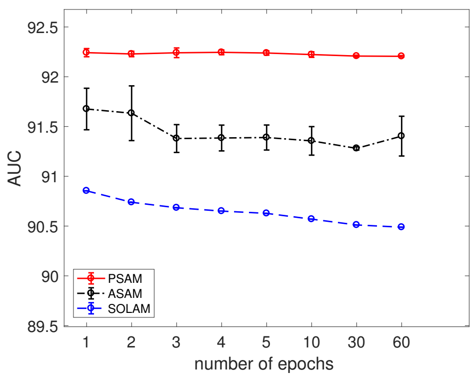
(b) letter



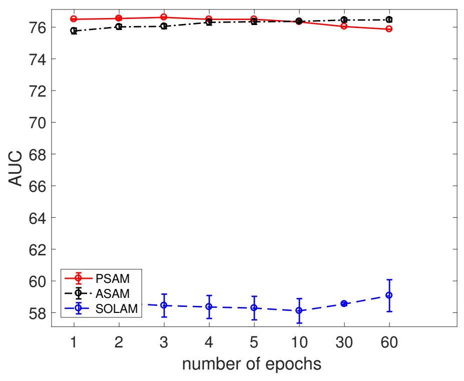
(c) w8a



(d) a9a

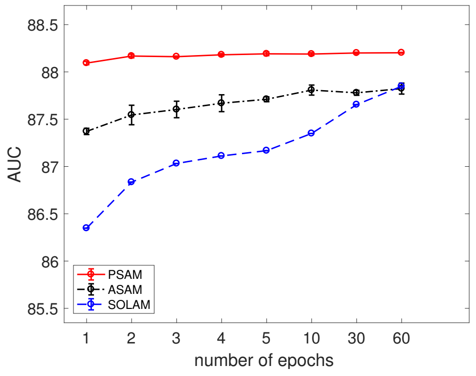


(e) ijcnn1

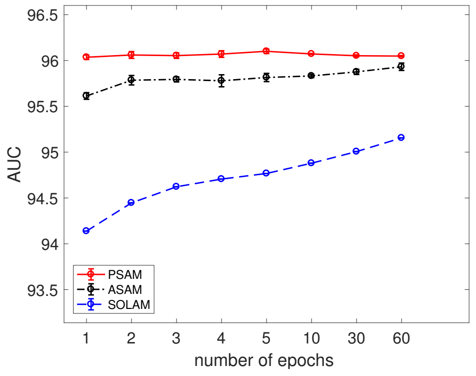


(f) cifar10

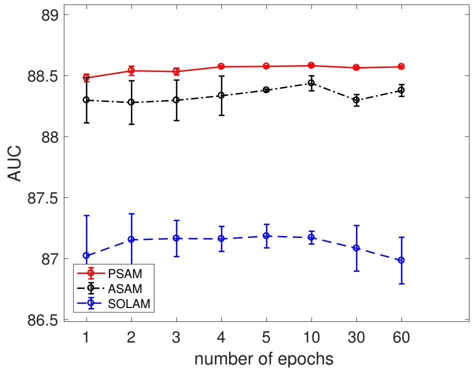
Figure 7.1: AUC classification accuracy with respect to the number of epochs for the stochastic linear AUC methods. We randomly pick a positive and negative instance in each iteration for PSAM and ASAM, where n iterations correspond to one epoch.



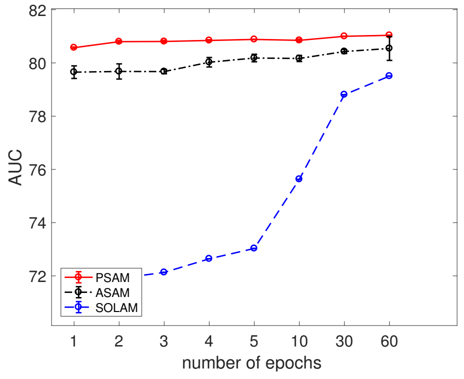
(g) connect-4



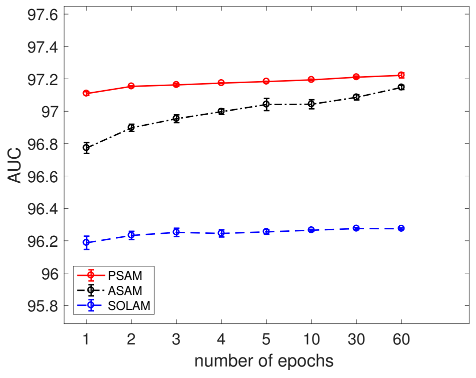
(h) mnist



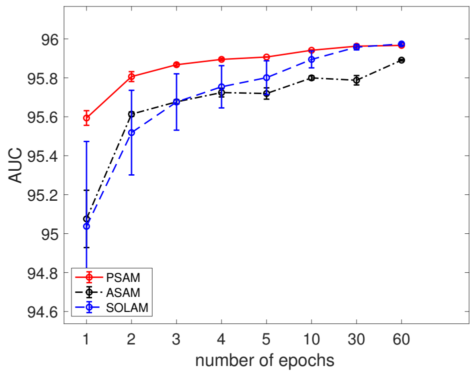
(i) acoustic



(j) aloi

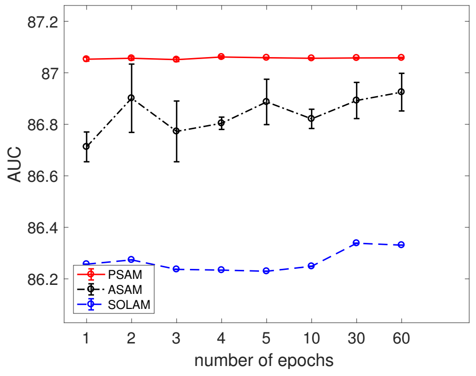


(k) webspam

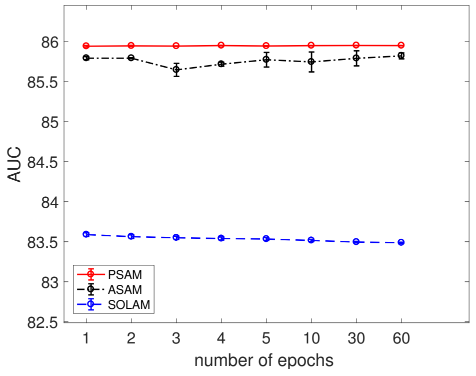


(l) epsilon

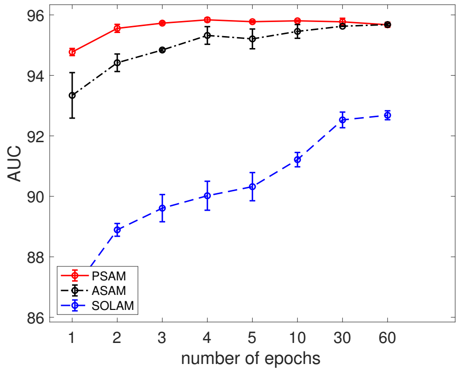
Figure 7.1: AUC classification accuracy with respect to the number of epochs for the stochastic linear AUC methods. We randomly pick a positive and negative instance in each iteration for PSAM and ASAM, where n iterations correspond to one epoch.



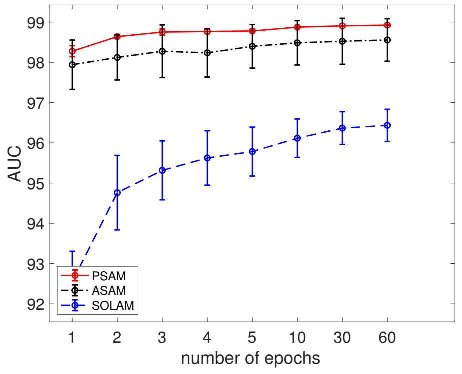
(m) covtype



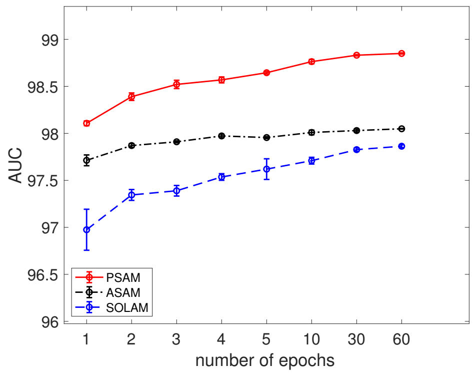
(n) susy



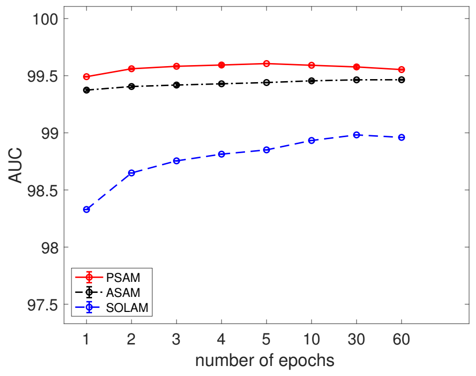
(o) farm-ads



(p) sector

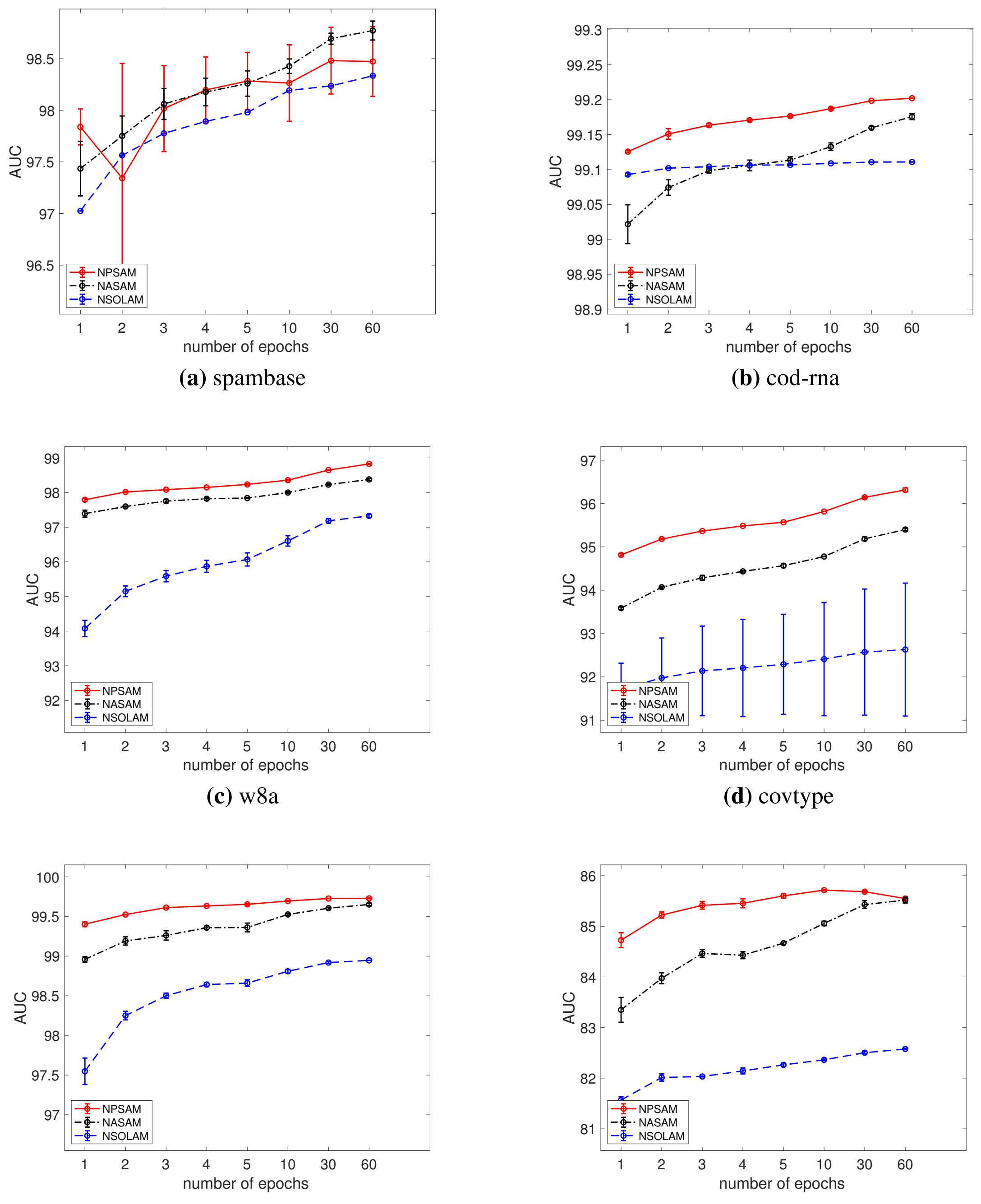


(q) news20



(r) real-sim

Figure 7.1: AUC classification accuracy with respect to the number of epochs for the stochastic linear AUC methods. We randomly pick a positive and negative instance in each iteration for PSAM and ASAM, where n iterations correspond to one epoch.

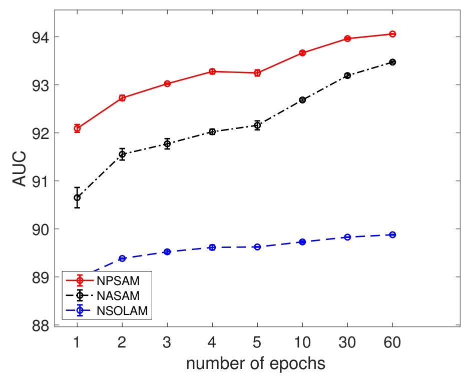


(a) spambase **(b)** cod-rna

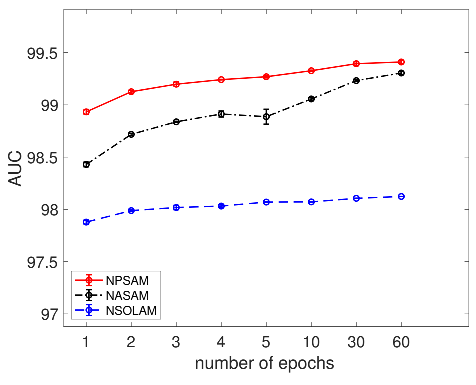
(c) w8a **(d)** covtype

(e) ijcnn1 **(f)** cifar10

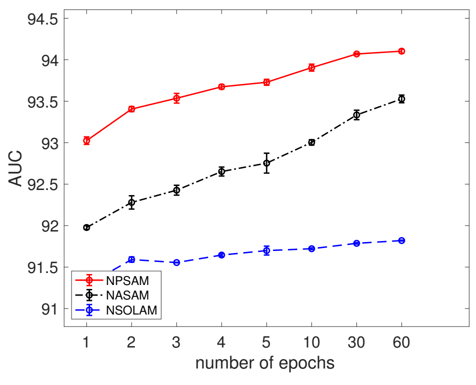
Figure 7.2: AUC classification accuracy with respect to the number of epochs for the kernelized stochastic AUC maximization algorithms. We randomly pick a positive and negative instance in each iteration for NPSAM and NASAM, where n iterations correspond to one epoch.



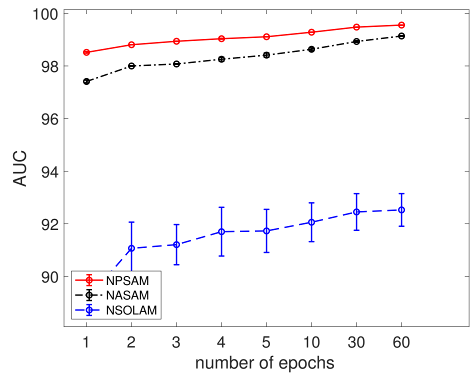
(g) connect-4



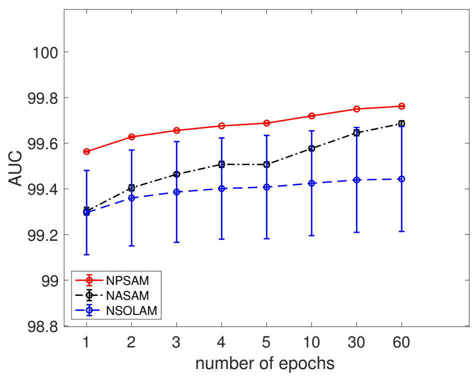
(h) mnist



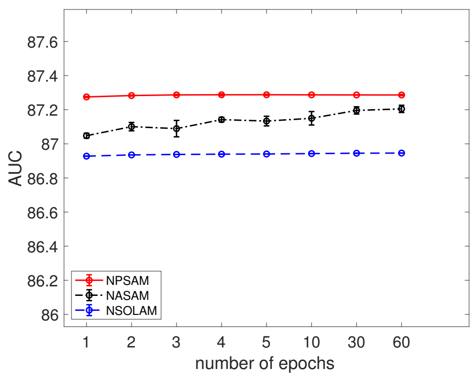
(i) acoustic



(j) aloi



(k) webspam



(l) susy

Figure 7.2: AUC classification accuracy with respect to the number of epochs for the kernelized stochastic AUC maximization algorithms. We randomly pick a positive and negative instance in each iteration for NPSAM and NASAM, where n iterations correspond to one epoch.

7.5 Conclusion

In this chapter, we have developed a stochastic AUC maximization method using proximal operator. We applied the proximal operator to the pairwise hinge loss function and optimized the model by scheduling both the regularization and the averaging steps. The experimental results on several benchmark datasets show that our proposed method surpasses the state-of-the-art scalable AUC maximization algorithms in terms of AUC classification accuracy. We also demonstrate via experiments that our proximal stochastic AUC maximization algorithm is able to reach the optimal solution from the first epoch. For future work, we will study our proximal stochastic AUC maximization algorithm with other loss function such as smoothed hinge loss function and squared loss. Further, the presented proximal point approach can also be extended to work with SOLAM algorithm that maximizes the AUC using a univariate loss function in a saddle point framework [68, 99].

Chapter 8

Conclusion and Future Work

The AUC measure is a critical evaluation tool for various machine learning and data mining applications. The importance of the AUC measure lies in its insensitivity to class distribution, while other measures, such as error rate and accuracy, are affected by the ratio of positive to negative classes. The standard SVM machines lack scalability to optimize the AUC due to its multivariate nature. Thus, many studies have developed SVM machines designed to maximize the AUC metric. However, the high complexity of batch kernelized AUC machines renders them infeasible for training large-scale datasets, while their linear variants are prone to the under-fitting problem when applied to datasets having complex nonlinear decision boundaries.

The first contribution of this thesis is a large-scale nonlinear AUC maximization algorithm that approximates a kernel matrix using the k-means Nyström method and random Fourier approaches. This nonlinear model is constructed by implementing linear AUC machines on the approximate feature mappings. However, the complexity of each matrix-vector product in batch linear AUC machines is linear in the data size at best, which hinders their scalability. Recently, many methods have been developed using online and stochastic learning optimization techniques to scale up the AUC maximization algorithm to massively large imbalanced datasets.

However, the online AUC maximization methods provide fast training algorithms at the expense of the optimal convergence rate. This issue results in inferior AUC classification accuracy for complex real-world datasets. The second contribution of this thesis is developing scalable online learning algorithms to improve the convergence of the AUC maximization algorithm. The improvement is achieved by learning a confidence-weighted algorithm, which is a second-order online learning method, to optimize a pairwise loss function and using a buffering strategy to deal with the multivariate nature of the loss function. We have also developed a diagonal variant for this second-order AUC maximization algorithm to address imbalanced datasets with very high dimensionality. For online nonlinear learning, we have extended our confidence-weighted AUC maximization algorithm to learn on an approximate feature space constructed via random Fourier features.

As a third contribution, we have developed an accelerated stochastic AUC maximization algorithm that can converge faster than the competing AUC maximization methods. The acceleration in our stochastic AUC maximization algorithm is attained by scheduling both the regularization update and the averaging step. The nonlinear variants are developed by learning the accelerated stochastic AUC maximization classifier on an approximate feature space constructed via the k-means Nyström and random Fourier methods.

The fourth contribution of this work concerns further improving the convergence rate of our accelerated stochastic algorithm for AUC maximization. We boost the rate of convergence of our stochastic AUC maximization algorithm by implementing a proximal operator of the pairwise hinge loss function. The proximal algorithm achieved the state-of-the-art convergence rate.

8.1 Future Work

One possible future work is the application of our accelerated stochastic techniques to optimize other non-decomposable measures such as partial AUC [105], precision@k, and F1-measure [106]. Moreover, the objective functions of our scalable AUC maximization methods can be modified to incorporate unlabelled instances [107]. The accelerated stochastic AUC maximization algorithms that are presented in this work can be utilized in developing a variant of factorization machines [108] for AUC maximization with a fast rate of convergence.

Another interesting future work is devising a scalable AUC maximization algorithm under fairness constraints [109]. A fairness-aware machine learning algorithm is an emerging field that considers learning algorithms with impartial predictive classifiers. Development of a fairness-aware AUC maximization algorithm is of importance because imbalanced phenomenon in real-world applications would worsen the bias of the data.

Bibliography

[1] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

[2] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239. ACM, 2008.

[3] Shai Fine and Katya Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2(Dec):243–264, 2001.

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

[5] S Sathiya Keerthi, Olivier Chapelle, and Dennis DeCoste. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7(Jul):1493–1515, 2006.

[6] Thorsten Joachims and Chun-Nam John Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2-3):179–193, 2009.

[7] Lubor Ladicky and Philip Torr. Locally linear support vector machines. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 985–992, 2011.

[8] Quanquan Gu and Jiawei Han. Clustered support vector machines. In *Artificial Intelligence and Statistics*, pages 307–315, 2013.

[9] Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. Local deep kernel learning for efficient non-linear svm prediction. In *International Conference on Machine Learning*, pages 486–494, 2013.

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

[11] Jing Lu, Steven CH Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. Large scale online kernel learning. *Journal of Machine Learning Research*, 17(47):1, 2016.

[12] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[13] Corinna Cortes and Mehryar Mohri. Auc optimization vs. error rate minimization. In *Advances in neural information processing systems*, pages 313–320, 2004.

[14] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[17] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[18] Byron C Wallace, Kevin Small, Carla E Brodley, and Thomas A Trikalinos. Class imbalance, redux. In *2011 IEEE 11th international conference on data mining*, pages 754–763. IEEE, 2011.

[19] Tobias Schnabel, Paul N Bennett, and Thorsten Joachims. Improving recommender systems beyond the algorithm. *arXiv preprint arXiv:1802.07578*, 2018.

[20] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186. Nashville, USA, 1997.

[21] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[22] Yi Lin, Yoonkyung Lee, and Grace Wahba. Support vector machines for classification in nonstandard situations. *Machine learning*, 46(1-3):191–202, 2002.

[23] Shuichi Katsumata and Akiko Takeda. Robust cost sensitive support vector machine. In *Artificial Intelligence and Statistics*, pages 434–443, 2015.

[24] Sougata Chaudhuri, Georgios Theocharous, and Mohammad Ghavamzadeh. Recommending advertisements using ranking functions, January 18 2016. US Patent App. 14/997,987.

[25] Shivani Agarwal, Deepak Dugar, and Shiladitya Sengupta. Ranking chemical structures for drug discovery: a new machine learning approach. *Journal of chemical information and modeling*, 50(5):716–731, 2010.

[26] Zheng Xie and Ming Li. Cutting the software building efforts in continuous integration by semi-supervised online auc optimization. In *IJCAI*, pages 2875–2881, 2018.

[27] Jonathan Root, Jing Qian, and Venkatesh Saligrama. Learning efficient anomaly detectors from k-nn graphs. In *Artificial Intelligence and Statistics*, pages 790–799, 2015.

[28] Sougata Chaudhuri, Georgios Theocharous, and Mohammad Ghavamzadeh. Importance of recommendation policy space in addressing click sparsity in personalized advertisement display. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 32–46. Springer, 2017.

[29] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71, 1997.

[30] Tong Tong, Robin Wolz, Qinquan Gao, Ricardo Guerrero, Joseph V Hajnal, and Daniel Rueckert. Multiple instance learning for classification of dementia in brain mri. *Medical image analysis*, 18(5):808–818, 2014.

[31] Fayyaz ul Amir Afsar Minhas and Asa Ben-Hur. Multiple instance learning of calmodulin binding sites. *Bioinformatics*, 28(18):i416–i422, 2012.

[32] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 426–437. ACM, 2016.

[33] Lech Madeyski and Marcin Kawalerowicz. Continuous defect prediction: the idea and a related dataset. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 515–518. IEEE Press, 2017.

[34] Jacqui Finlay, Russel Pears, and Andy M Connor. Data stream mining for predicting software build outcomes using source code metrics. *Information and Software Technology*, 56(2):183–198, 2014.

[35] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. *Information and Software Technology*, 87:206–220, 2017.

[36] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E Hassan. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, 21(5):2072–2106, 2016.

[37] Harikrishna Narasimhan and Shivani Agarwal. Support vector algorithms for optimizing the partial area under the roc curve. *Neural Computation*, 2017.

[38] Wei Gao and Zhi-Hua Zhou. On the consistency of auc pairwise optimization. In *IJCAI*, pages 939–945, 2015.

[39] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998.

[40] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[41] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[42] Olivier Chapelle and S Sathiya Keerthi. Efficient algorithms for ranking with svms. *Information Retrieval*, 13(3):201–215, 2010.

[43] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble nystrom method. In *Advances in Neural Information Processing Systems*, pages 1060–1068, 2009.

[44] Djallel Bouneffouf and Inanc Birol. Sampling with minimum sum of squared similarities for nystrom-based large scale spectral clustering. In *IJCAI*, pages 2313–2319, 2015.

[45] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling techniques for the nystrom method. In *Artificial Intelligence and Statistics*, pages 304–311, 2009.

[46] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(Dec):2153–2175, 2005.

[47] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13(Apr):981–1006, 2012.

[48] Dino Oglic and Thomas Gärtner. Nyström method with kernel k-means++ samples as landmarks. In *International Conference on Machine Learning*, pages 2652–2660, 2017.

[49] Salomon Bochner. *Harmonic analysis and the theory of probability*. Courier Corporation, 2005.

[50] Felix X Yu, Sanjiv Kumar, Henry Rowley, and Shih-Fu Chang. Compact nonlinear maps and circulant extensions. *arXiv preprint arXiv:1503.03893*, 2015.

[51] Alain Rakotomamonjy. Optimizing area under roc curve with svms. In *ROCAI*, pages 71–80, 2004.

[52] Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384. ACM, 2005.

[53] Ching-Pei Lee and Chih-Jen Lin. Large-scale linear ranksvm. *Neural computation*, 26(4):781–817, 2014.

[54] Tzu-Ming Kuo, Ching-Pei Lee, and Chih-Jen Lin. Large-scale kernel ranksvm. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 812–820. SIAM, 2014.

[55] Antti Airola, Tapio Pahikkala, and Tapio Salakoski. Training linear ranking svms in linearithmic time using red–black trees. *Pattern Recognition Letters*, 32(9):1328–1336, 2011.

[56] Toon Calders and Szymon Jaroszewicz. Efficient auc optimization for classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer, 2007.

[57] K Chen, R Li, Y Dou, Z Liang, and Q Lv. Ranking support vector machine with kernel approximation. *Computational intelligence and neuroscience*, 2017:4629534, 2017.

[58] Vishal Kakkar, Shirish Shevade, S Sundararajan, and Dinesh Garg. A sparse nonlinear classifier design using auc optimization. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 291–299. SIAM, 2017.

[59] Jialei Wang, Peilin Zhao, and Steven C Hoi. Exact soft confidence-weighted learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 121–128, 2012.

[60] Koby Crammer, Mark Dredze, and Alex Kulesza. Multi-class confidence weighted algorithms. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 496–504. Association for Computational Linguistics, 2009.

[61] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.

[62] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[63] Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In *Advances in neural information processing systems*, pages 414–422, 2009.

[64] Wojciech Kotlowski, Krzysztof J Dembczynski, and Eyke Huellermeier. Bipartite ranking through minimization of univariate loss. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1113–1120, 2011.

[65] Peilin Zhao, Rong Jin, Tianbao Yang, and Steven C Hoi. Online auc maximization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 233–240, 2011.

[66] Wei Gao, Rong Jin, Shenghuo Zhu, and Zhi-Hua Zhou. One-pass auc optimization. In *ICML (3)*, pages 906–914, 2013.

[67] Yi Ding, Peilin Zhao, Steven CH Hoi, and Yew-Soon Ong. An adaptive gradient method for online auc maximization. In *AAAI*, pages 2568–2574, 2015.

[68] Yiming Ying, Longyin Wen, and Siwei Lyu. Stochastic online auc maximization. In *Advances in Neural Information Processing Systems*, pages 451–459, 2016.

[69] Balázs Szörényi, Snir Cohen, and Shie Mannor. Non-parametric online auc maximization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 575–590. Springer, 2017.

[70] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[71] Koby Crammer, Mark Dredze, and Fernando Pereira. Confidence-weighted linear classification for text categorization. *The Journal of Machine Learning Research*, 13(1):1891–1926, 2012.

[72] Justin Ma, Alex Kulesza, Mark Dredze, Koby Crammer, Lawrence K Saul, and Fernando Pereira. Exploiting feature covariance in high-dimensional online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 493–500, 2010.

[73] Deng Cai, Xiaofei He, and Jiawei Han. Locally consistent concept factorization for document clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):902–913, 2011.

[74] Koby Crammer, Jaz Kandola, and Yoram Singer. Online classification on a budget. In *Advances in neural information processing systems*, pages 225–232, 2004.

[75] Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

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

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

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

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

[80] Junjie Hu, Haiqin Yang, Michael R Lyu, Irwin King, and Anthony Man-Cho So. Online nonlinear auc maximization for imbalanced data sets. *IEEE transactions on neural networks and learning systems*, 29(4):882–895, 2018.

[81] Yi Ding, Chenghao Liu, Peilin Zhao, and Steven CH Hoi. Large scale kernel methods for online auc maximization. In *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 91–100. IEEE, 2017.

[82] Majdi Khalid, Indrakshi Ray, and Hamidreza Chitsaz. Confidence-weighted bipartite ranking. In *Advanced Data Mining and Applications: 12th International Conference, ADMA 2016, Gold Coast, QLD, Australia, December 12-15, 2016, Proceedings 12*, pages 35–49. Springer, 2016.

[83] Trung Le, Tu Nguyen, Vu Nguyen, and Dinh Phung. Dual space gradient descent for online learning. In *Advances in Neural Information Processing Systems*, pages 4583–4591, 2016.

[84] Tu Dinh Nguyen, Trung Le, Hung Bui, and Dinh Phung. Large-scale online kernel learning with random feature reparameterization. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2543–2549. AAAI Press, 2017.

[85] Yan Yan, Tianbao Yang, Yi Yang, and Jianhui Chen. A framework of online learning with imbalanced streaming data. In *AAAI*, pages 2817–2823, 2017.

[86] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

[87] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[88] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[89] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[90] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[91] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaing. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[92] Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10(Jul):1737–1754, 2009.

[93] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[94] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[95] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[96] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[97] Guillaume Papa, Stéphan Clémençon, and Aurélien Bellet. Sgd algorithms based on incomplete u-statistics: large-scale minimization of empirical risk. In *Advances in Neural Information Processing Systems*, pages 1027–1035, 2015.

[98] Mingrui Liu, Xiaoxuan Zhang, Zaiyi Chen, Xiaoyu Wang, and Tianbao Yang. Fast stochastic auc maximization with o (1/n)-convergence rate. In *International Conference on Machine Learning*, pages 3195–3203, 2018.

[99] Michael Natole, Yiming Ying, and Siwei Lyu. Stochastic proximal algorithms for auc maximization. In *International Conference on Machine Learning*, pages 3707–3716, 2018.

[100] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[101] Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011.

[102] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[103] Majdi Khalid, Indrakshi Ray, and Hamidreza Chitsaz. Scalable nonlinear auc maximization methods. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 292–307. Springer, 2018.

[104] Aaron Defazio. A simple practical accelerated method for finite sums. In *Advances in Neural Information Processing Systems*, pages 676–684, 2016.

[105] Harikrishna Narasimhan and Shivani Agarwal. A structural svm based approach for optimizing partial auc. In *International Conference on Machine Learning*, pages 516–524, 2013.

[106] Purushottam Kar, Harikrishna Narasimhan, and Prateek Jain. Online and stochastic gradient methods for non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, pages 694–702, 2014.

[107] Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Semi-supervised auc optimization based on positive-unlabeled learning. *Machine Learning*, 107(4):767–794, 2018.

[108] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[109] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*, pages 962–970, 2017.

Appendix A

The Proximal Operator of the Hinge Loss

Proximal operators for most loss functions have efficient or closed form solutions. In what follows, we derive the analytical solution for the proximal operator of the pairwise hinge loss function, which is similar to the solution presented in [90]. Let x_i^+ and x_j^- represent a random positive and negative instance, respectively. We assume that $x \in \mathcal{X}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathbb{R}$ is a Euclidean space, meaning that the magnitude of any vector in $\mathbb{R}$ is obtained by l_2-norm. Let $x_t = (x_i - x_j)$ denotes the difference vector at the t-th iteration. The pairwise hinge loss function is defined as:

$$f(w_t) = max\{0, 1 - w_t^T x_t\}.$$

The proximal mapping of $f(w_t)$ is achieved by the optimality condition that implies the following minimization problem:

$$\text{prox}_{\lambda t}(w) = \underset{v \in \mathbb{R}^d}{\text{argmin}} \left\{ \lambda f_t(v) + \frac{1}{2} ||v - w||^2 \right\}.$$

The precedent expression is a minimization problem that approximates to the vector v while taking into account the cost of this approximation $f(v)$. A closed form solution of this minimization problem can be attained by the optimality condition of the proximal operator. Recall that the gradient of the hinge loss function is defined as $f^{'}(w_t)x_t$, where $f^{'}$ is the subgradient of f, and has the following definition:

$$f^{'}(w_t) = \begin{cases} -1 & 1 - w_t^T x_t \geq 1 \\ 0 & 1 - w_t^T x_t \leq 0 \\ w_t^T x_t - 1 & \text{otherwise} \end{cases} .$$

For each case, the evaluation of the gradient is determined based on the projection onto the hyperplane or half-spaces. Consequently, the proximal operator of the hinge loss function can be redefined as an orthogonal projection [102]

$$\text{prox}_{\lambda t}(w) = \underset{v \in \mathbb{R}^d}{\text{argmin}} \left\{ \lambda f_t(v) + \frac{1}{2}||v - w||^2 \right\} = P_{\lambda H}(w),$$

where $H = \{v \in \mathbb{R}^d : v^T x = 1\}$. We can derive an explicit form for the problem of finding $P_H(w)$ as follows:

$$\begin{aligned} &\underset{v \in \mathbb{R}^d}{\text{argmin}}\, ||v - w||^2 \\ &s.t. \quad v^T x = 1 \ \ . \end{aligned}$$

The precedent constrained minimization problem can be solved using its optimality condition (i.e., KKT conditions), which yields the following solution:

$$P_H(w) = w - \frac{w^T x - 1}{||x||^2} x.$$

Therefore, the proximal operator of the hinge loss has the following closed-form solution,

$$\text{prox}(w) = w - \lambda g x_t,$$

where:

$$z = \frac{1 - w^T x_t}{\lambda ||x_t||^2}.$$

$$g = \begin{cases} 0 & \text{z} \ \le \ 0 \\ -1 & z \ \ge \ 1 \\ -z & 0 < z < 1 \end{cases} .$$