MBA: Mini-Batch AUC Optimization

San Gultekin, Avishek Saha, Adwait Ratnaparkhi, and John Paisley

***Abstract*—Area under the receiver operating characteristics curve (AUC) is an important metric for a wide range of machine learning problems, and scalable methods for optimizing AUC have recently been proposed. However, handling very large datasets remains an open challenge for this problem. This paper proposes a novel approach to AUC maximization, based on sampling mini-batches of positive/negative instance pairs and computing U-statistics to approximate a global risk minimization problem. The resulting algorithm is simple, fast, and learning-rate free. We show that the number of samples required for good performance is independent of the number of pairs available, which is a quadratic function of the positive and negative instances. Extensive experiments show the practical utility of the proposed method.**

***Index Terms*—Area under curve, machine learning, U-statistics, mini-batch, convex optimization, matrix concentration inequalities.**

I. Introduction

Bipartite ranking is an important problem in machine learning; given a set of positive and negative inputs, this problem is concerned with building a scoring system, such that the positive samples are ranked higher than the negative ones. Historically the problem has been studied extensively in signal processing literature [1], in particular for radar, as missing the presence of a target (miss detection) could have dire consequences. In this setting, a received signal is assigned a score, which is then compared against a threshold to decide if the target is present. The receiver operating characteristics (ROC) curve plots the ratio of true positives (detection) to false positives (false alarm) as a function of this threshold, and provides information about the behavior of the system. The area under the ROC curve (AUC) is a threshold-independent metric which measures the fraction of times a positive instance is ranked higher than a negative one. Therefore, it is a natural measure for bipartite ranking performance. These have a wide variety of applications such as imaging [2], dictionary learning [3], signal detection [4], and noise-enhanced detection [5]–[7].

The bipartite ranking problem and ROC curves are not limited to traditional applications. In fact, these tools are ubiquitous in modern machine learning problems. Important subfields include cost-sensitive learning [8], [9] and imbalanced data processing [10]. In the latter, one is given a dataset where the number of negative samples is dominant. This means a classifier which predicts all incoming instances to be negative will have very high prediction accuracy. On the other hand, it will have an AUC of 0. This is worse than random guessing, which would give 0.5, and so the AUC in a sense may be a better choice for performance metric, and devising methods to achieve higher AUC is a meaningful goal. For this reason, the AUC metric is heavily used for website ad click prediction problems [11], where only a very small fraction of web history contains ads clicked by visitors. In this case, a system with high AUC is the one which can distinguish the ads that are "interesting" for a user from the rest, whereas a simple classifier that maximizes prediction accuracy may simply predict "not interesting." Pairwise and triplet loss functions are also useful in computer vision tasks [12], [13].

Given that AUC is the primary performance measure for many problems, it is useful to devise algorithms that directly optimize this metric during the training phase. AUC optimization has been studied within the context of well-known machine learning methods, such as support vector machines [14], boosting [15], and decision trees [16]. However, most of these traditional approaches do not scale well as the size of the dataset grows—since AUC is defined over positive/negative *pairs*, it has quadratic growth of $\mathcal{O}(N_+N_-)$. Moreover, the AUC itself is a sum of indicator functions, and its direct optimization is NP-hard.

Recent research in this direction increasingly focuses on convex surrogate loss functions to represent the AUC. This enables one to use stochastic gradient methods to efficiently learn a ranking function [17] while avoiding the intricacies of non-convex optimization [18]. The first work in this direction is [19], where an online AUC maximization method based on proxy hinge loss was proposed. Later, [20] use the pairwise squared loss function to eliminate the need for buffering previous instances; [21] propose adaptive gradient/subgradient methods which can also handle sparse inputs, while [22], [23] consider the nonlinear AUC maximization problem using kernel and multiple-kernel methods. Most recently, [24] focuses on scalable kernel methods. Another related pairwise loss is used for training Siamese neural networks [12], which rely on the learned representations to distinguish different categories of inputs; we show an example of this in the experiments. Higher order loss functions such as triplet loss [25] and quadruplet loss [26] are also used in deep learning applications, with the aim of capturing more information between pairs. However, for AUC optimization the focus is mainly on pairwise loss, as the original loss is also defined this way and consistency results for pairwise surrogate losses are available as well [27].

While these approaches can significantly increase scalability [28], for very large datasets their sequential nature can still be problematic. One widely used technique—particularly for training deep neural networks on large datasets [29]—is processing data in mini-batches. AUC maximization methods that can utilize mini-batch processing is thus desirable. In this paper we propose a novel algorithm for fast AUC optimization. Our approach, called Mini-Batch AUC Optimization (MBA) is based on a convex relaxation of the AUC function. However instead of using stochastic gradients, it uses first and second order U-statistics of pairwise differences. A distinctive feature of our approach is its being *learning-rate free*, unlike mini-

batch gradient descent methods. This is important, as tuning the step size *a priori* is a difficult task and generic approaches such as cross-validation are inefficient when the dataset is large.

One major challenge of AUC optimization is that, even if convex relaxation is applied, the resulting problem is defined over pairs of positive/negative samples and the optimization has a sample cost of $\mathcal{O}(N_+N_-)$. This grows prohibitively large even for moderate datasets. Since mini-batch optimization is based on sub-sampling it is important to understand the behavior of MBA as a function of sample size. Our theoretical analysis shows that the solution returned by MBA concentrates around the batch solution based on the *entire pair ensemble* of the data, with an exponential tail bound. Unlike previous work, our proofs are based on the more recent results on matrix concentration [30] and are quite straightforward. In terms of related work, while U-processes for ranking problems have previously been explored by [31], scalable mini-batch algorithms using U-statistics have not been developed. The nearest work of [24] uses mini-batch techniques, but for gradient descent. It is also worthwhile to note that discrete optimization such as submodular optimization has also been studied within the context of pairwise learning and computer vision [32], [33]. In contrast, the continuous relaxation in our approach is key to obtain favorable sample complexity bounds.

Our contributions are (i) introduction of a novel AUC opimization algorithm that does not require gradients or learning rates, (ii) theoretical analysis of the algorithm showing its favorable sample complexity, and (iii) extensive experiments on real datasets, including applications to ad click prediction and one-shot image recognition.

We organize the paper as follows: In Section II we start with an overview of the bipartite ranking problem and the statistical learning setup. In Section III we develop the MBA algorithm and its theoretical analysis. Section IV contains extensive experiments, including a simulation study, fifteen datasets from UCI/LIBSVM repositories, and three large-scale commercial web click data. We conclude in Section V.

II. Background

Binary classification is widely studied in machine learning. In this problem, for each given input $\boldsymbol{x} \in \mathcal{X}$ there is a a corresponding label $y \in \mathcal{Y} = \{+, -\}$. In the learning setup, a set of labeled data is provided for training, and the aim is to make accurate predictions on the unknown inputs. Similar to the PAC learning framework, we assume that the samples provided for training are i.i.d. with the following unknown distributions[1]

$$[\boldsymbol{X} \mid y = +] \sim \mathcal{P}^+ \quad , \quad [\boldsymbol{X} \mid y = -] \sim \mathcal{P}^- \tag{1}$$

Since the distributions are unknown, a model $f(\cdot)$ is fitted to the training data using a specific loss function, such as cross-entropy loss for logistic regression and hinge loss for support vector machines. The loss function we focus on is the area under receiver operating characteristics (ROC) curve, or AUC. Its functional form is given by

$$\text{AUC} = \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \Big[\ \mathbb{1}\{f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-) > 0\} \Big]. \tag{2}$$

We note that this is a measure of how well $f(\cdot)$ ranks a positive sample higher than a negative one, both of which come from the data generating distribution.

Eq. (2) shows that the AUC is the expectation of an indicator function. This means the corresponding empirical risk function will be a sum of indicators, which renders the optimization NP-hard. For this reason we first discuss a relaxation of the original problem using convex surrogate loss functions.

Replacing $\mathbb{1}[f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-) > 0]$ in Eq. (2) with the pairwise convex surrogate loss $\phi(\boldsymbol{x}^+, \boldsymbol{x}^-) = \phi(f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-))$, the aim now becomes to minimize the ϕ-risk [34]

$$R_\phi(f) = \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \left[\phi(f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-)) \right]. \tag{3}$$

This is the Bayes risk of the scoring function [35]. There are many possible choices for surrogate function; some common choices are the pairwise squared loss (PSL), pairwise hinge loss (PHL), pairwise exponential loss (PEL), and pairwise logistic loss (PLL) [27]:

$$\begin{aligned} &\phi_{\text{PSL}}(t) = (1-t)^2 \,, \ \phi_{\text{PHL}}(t) = \max(0, \ 1-t) \,, \\ &\phi_{\text{PEL}}(t) = \exp(-t) \,, \ \phi_{\text{PLL}}(t) = \log(1 + \exp(-t)) \,, \end{aligned} \tag{4}$$

where $t := f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-)$ is the pairwise scoring difference. Among the recent works on AUC optimization, [19] and [36] use PHL, whereas [20] and [21] focus on PSL. On the other hand, all these studies are focused on deriving a stochastic gradient-based algorithm. In this paper, we use the PSL function for two reasons: (i) its consistency with the original AUC loss is proven [27], and (ii) the structure of PSL allows for a mini-batch algorithm, for which theoretical guarantees can be derived. Unlike stochastic gradient methods, this formulation is learning rate-free, which increases its practical usefulness.

We now make one further assumption that the scoring function is linear in the original input space. (We will discuss nonlinear extensions in Section III.) In this case we have the further simplification $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ and the ϕ-risk becomes

$$\begin{aligned} R_\phi(f) &= \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \left[\big(\ 1 - \boldsymbol{w}^\top (\boldsymbol{x}^+ - \boldsymbol{x}^-) \ \big)^2 \right] \\ &= 1 - 2\boldsymbol{w}^\top \mathbb{E}\left[(\boldsymbol{x}^+ - \boldsymbol{x}^-) \right] \\ &\qquad + \boldsymbol{w}^\top \mathbb{E}\left[(\boldsymbol{x}^+ - \boldsymbol{x}^-)(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top \right] \boldsymbol{w} \\ &= 1 - 2\boldsymbol{w}^\top \boldsymbol{\mu} + \boldsymbol{w}^\top \boldsymbol{\Sigma} \boldsymbol{w}, \end{aligned} \tag{5}$$

where we define $\boldsymbol{x}_{\text{diff}} := (\boldsymbol{x}^+ - \boldsymbol{x}^-)$, and $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}_{\text{diff}}]$ and $\boldsymbol{\Sigma} = \mathbb{E}[\boldsymbol{x}_{\text{diff}} \ \boldsymbol{x}_{\text{diff}}^\top]$. Note that $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ characterize the first and second order statistics of the pairwise difference. This is important because the quality of bipartite ranking does not directly depend on the positive and negative inputs, but the differences between them. This observation will form the basis of our mini-batch algorithm in the next section. Finally, by definition $\boldsymbol{\Sigma}$ is at least positive semi-definite, so there is a unique $\boldsymbol{w}^\star$ that achieves the minimum $R_\phi(f)$.

Optimizing the objectives in Eq. (2) or Eq. (5) would require knowledge of $\mathcal{P}^+$ and $\mathcal{P}^-$, which is unavailable. Instead we

[1]We consider the case with no label noise, but the modification to that case is straightforward.

assume an i.i.d. sample from these distributions. The task is to learn a score function which yields high AUC on the test data, also known as the generalization performance. Here the corresponding empirical AUC metric is

$$\text{AUC} = \sum_{i=1}^{N_+}\sum_{j=1}^{N_-} \mathbb{1}\{f(\boldsymbol{x}_i^+) - f(\boldsymbol{x}_j^-) > 0\} \tag{6}$$

which replaces the expectation in Eq. (2) with a sample-based average. As mentioned before, direct optimization of the empirical AUC is NP-hard as it is a sum of indicators. Furthermore, the number of summands grows quadratically with the training data. To sidestep this difficulty, a surrogate loss function $\phi(\cdot)$ can be chosen to replace the Eq. (6), as was done for Eq. (2). In particular, the empirical ϕ-risk (c.f. Eq. (3)) has the following general form

$$\widehat{R}_\phi(f) = \frac{1}{2N_+N_-}\sum_{i=1}^{N_+}\sum_{j=1}^{N_-} \phi(f(\boldsymbol{x}_i^+) - f(\boldsymbol{x}_j^-)) \ . \tag{7}$$

Based on this final equation we derive the MBA algorithm in the next section.

III. Mini-Batch AUC Optimization

Using the pairwise squared loss function, we obtained a convex optimization problem in place of the original NP-hard problem. However, Eq. (3) relies on the unknown data generating distribution $\mathcal{P}$. We instead substitute the empirical risk in Eq. (7) as an approximation. We stress that the term $N := N_+N_-$ corresponds to the total number of pairs in the data. Similar to the ϕ-risk, optimizing the empirical risk yields a convex problem, but the number of pairs grows quadratically in the number of data points. Therefore, even for moderate datasets, minimizing the empirical risk in Eq. (7) becomes computationally intractable.

After making these substitutions, the empirical risk becomes

$$\begin{aligned}\widehat{R}_\phi(\boldsymbol{w}) = -\boldsymbol{w}^\top &\left[\frac{1}{N_+N_-}\sum_{i=1}^{N_+}\sum_{j=1}^{N_-}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\right] \\ &+ \frac{1}{2}\boldsymbol{w}^\top\left[\frac{1}{N_+N_-}\sum_{i=1}^{N_+}\sum_{j=1}^{N_-}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top\right]\boldsymbol{w}\end{aligned} \tag{8}$$

and we define

$$\begin{aligned}\boldsymbol{\mu}_N &= \frac{1}{N_+N_-}\sum_{i=1}^{N_+}\sum_{j=1}^{N_-}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_N &= \frac{1}{N_+N_-}\sum_{i=1}^{N_+}\sum_{j=1}^{N_-}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top.\end{aligned} \tag{9}$$

The variables in Eq. (9) are sample-based approximations to the first and second moments of $\boldsymbol{x}_{\text{diff}}$, which are denoted by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in Eq. (3). Overall, the optimization problem to be solved is

$$\boldsymbol{w}_N^\star = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{\Sigma}_N\boldsymbol{w} - \boldsymbol{w}^\top\boldsymbol{\mu}_N + R_{\text{EL}}(\boldsymbol{w}), \tag{10}$$

where $R_{\text{EL}}(\boldsymbol{w}) := \lambda_1\|\boldsymbol{w}\|_1 + (1/2)\lambda_2\|\boldsymbol{w}\|_2^2$ is the elastic net regularizer [37], which we add to prevent overfitting. Note that, unlike Eq. (3), there is always a unique optimum since the elastic net penalty makes the objective strictly convex. In addition, this regularizer encourages solution which combines small ℓ_2 norm with sparsity. By substituting appropriate values for λ_1 and λ_2 we also recover ridge and lasso regression. In this paper, we report results for all three cases.

Since it is impractical to use all N samples, we propose to use mini-batches to obtain estimates of the moments. This is a simple process which only requires the computation of U-statistics. Note that, given a parameter θ and symmetric measurable function h which satisfies $\theta = h(X_1, \ldots, X_m)$, the corresponding U-statistic is given by

$$U_n = \binom{n}{m}^{-1}\sum_{C_{n,m}} h(X_1, \ldots, X_n), \tag{11}$$

where $C_{n,m}$ is the set of all length-m combinations with increasing indices. U-statistics are able to provide best unbiased estimators [38], and for $\boldsymbol{\Sigma}_N$ they also provided a building block for an exponential concentration bound [30]. Our theoretical analysis will use this property.

A. The MBA algorithm

We now describe the proposed MBA algorithm. Let T be the total number of rounds. At round t we sample B positive and B negative samples from the entire population with replacement. Let $\mathcal{S}_t^+$ and $\mathcal{S}_t^-$ be the arrays of sample indices and let $\mathcal{S}_t$ be the array of pairs stored as tuples of the form $(\mathcal{S}_t^+(i), \mathcal{S}_t^-(i))$—note that we do not form the Cartesian product. The expressions for U-statistics of the first and second moments simplify from Eq. (11) as

$$\begin{aligned}\boldsymbol{\mu}_t &:= \frac{1}{B}\sum_{(i,j)\in\mathcal{S}_t}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_t &:= \frac{1}{B}\sum_{(i,j)\in\mathcal{S}_t}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top.\end{aligned} \tag{12}$$

Finally let $S = BT$ denote the total number of pairs sampled by our algorithm. We also introduce the notation $\mathcal{S}_{1:T}$ for the entire array of pairs sampled during all rounds. The overall moment approximations are therefore

$$\begin{aligned}\boldsymbol{\mu}_S &:= \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\mu}_t = \frac{1}{BT}\sum_{(i,j)\in\mathcal{S}_{1:T}}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_S &:= \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\Sigma}_t = \frac{1}{BT}\sum_{(i,j)\in\mathcal{S}_{1:T}}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top,\end{aligned} \tag{13}$$

and the optimization problem constructed by MBA is

$$\boldsymbol{w}_S^\star = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{\Sigma}_S\boldsymbol{w} - \boldsymbol{w}^\top\boldsymbol{\mu}_S + R_{\text{EL}}(\boldsymbol{w}). \tag{14}$$

MBA aims to construct and solve this approximation to the global risk minimization problem in Eq. (3). In contrast, stochastic gradient-based approaches seek a solution through local gradient approximations to the global function. As we

Algorithm 1 Mini-Batch AUC Optimization (MBA)

1: **Require:** B, T, λ_1, λ_2
2: **Input:** $\boldsymbol{X}^+$, $\boldsymbol{X}^-$
3: **Output:** $\boldsymbol{w}^\star$
4: Initialize $\boldsymbol{\mu}_S = \mathbf{0}$ and $\boldsymbol{\Sigma}_S = \mathbf{0}$.
5: **for** $t = 1, \ldots, T$ **do**
6: Construct index set $\mathcal{S}_t^+$ of size B sampling positive examples uniformly with replacement.
7: Construct index set $\mathcal{S}_t^-$ of size B sampling negative examples uniformly with replacement.
8: Construct $\mathcal{S}_t(i) = (\mathcal{S}_t^+(i), \mathcal{S}_t^-(i)),\ i = 1, \ldots, B$.
9: $\boldsymbol{\mu}_S \leftarrow \boldsymbol{\mu}_S + \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_t} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)$
10: $\boldsymbol{\Sigma}_S \leftarrow \boldsymbol{\Sigma}_S + \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_t} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top$
11: **end for**
12: $\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w}^\top - \boldsymbol{w}^\top \boldsymbol{\mu}_S + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{w}\|_2^2$
13: Note: Here $\boldsymbol{X}^+$ and $\boldsymbol{X}^-$ correspond to the $d \times N_+$ and $d \times N_-$ dimensional matrices of positive and negative instances. The indices in sets $\mathcal{S}$, $\mathcal{S}^+$ and $\mathcal{S}^-$ correspond to column numbers. Consequently, each vector $\boldsymbol{x}_i^+$ or $\boldsymbol{x}_j^-$ are $d \times 1$, $\boldsymbol{\mu}_S$ is $d \times 1$, $\boldsymbol{\Sigma}_S$ is $d \times d$.

will show in the experiments, this is an important difference and MBA can find better solutions since it constructs a global problem first. We summarize MBA in Algorithm 1. In Lines 9 and 10 the algorithm computes the empirical mean and covariance. Then in Line 12 the elastic net optimization of Eq. (10) is applied.

Mini-batch optimizations are a common feature of machine learning, including training of deep neural networks [29] and scalable Bayesian inference [39]. The main benefit of using mini-batches is its speed. Online methods for optimizing AUC, however, require sequential processing, as parameters are updated per input. This is the main reason MBA offers a significant improvement in speed. MBA offers several other advantages: Since sampling pairs and computing U-statistics is an isolated process, MBA can easily be distributed across machines, which can work asynchronously. Therefore MBA is suitable for cluster computing. Secondly, streaming and/or nonstationary data processing can be incorporated into the MBA framework, as it can process streams as blocks and give larger weights to more recent ones.

From an algorithmic perspective, as far as the sample size is concerned the only important parameter is S; we introduced B and T to further manage computational resources. For instance when S samples are too large to fit into memory we can instead use B samples in T rounds. Alternatively, S samples can be obtained by T machines with B samples per machine in a single round. In either case, we will have S uniform pairs sampled from the entire dataset.

B. Theoretical Analysis

Solving the regularized empirical risk minimization problem in Eq. (10) requires processing N pairwise samples. As this number grows quadratically with the number of positive and negative samples, it is often not possible to do this exactly. The proposed MBA addresses this problem by approximating the N-pair problem with an S-pair one, where S samples are collected in mini-batches and the total number of processed samples is much less than N. This results in the problem in Eq. (14). Clearly the success of this approach depends on how well the second problem approximates the first. In this section we derive performance guarantees.

Formally, given N samples we can solve the following regularized empirical risk minimization problem

$$\boldsymbol{w}_N^\star = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_N \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_N + R_{\text{EL}}(\boldsymbol{w})$$

however as N can be very large we instead solve

$$\boldsymbol{w}_S^\star = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_S + R_{\text{EL}}(\boldsymbol{w})$$

where S uniform samples are used. We would like to show that the two solutions are close to each other when $S \ll N$. We next derive a bound for this closeness. Using the difference between final costs $|\mathcal{L}_N(\boldsymbol{w}_N) - \mathcal{L}_N(\boldsymbol{w}_S)|$ results in a regret bound similar to those used in comparing online/batch versions of algorithms [40], while the Euclidean distance $d(\boldsymbol{w}_N - \boldsymbol{w}_S) = \|\boldsymbol{w}_N - \boldsymbol{w}_S\|_2$ measures how similar the two solutions are and is used by recent work on matrix sketching [41]. Here we show results primarily for the former, but we also provide an epsilon-delta based argument for parameter estimation.

We define the following two functions for convenience,

$$\begin{aligned}\mathcal{L}_N &= \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_N \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_N + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{w}\|_2^2 \\ \mathcal{L}_S &= \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_S + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{w}\|_2^2 \ , \end{aligned} \quad (15)$$

and let $\boldsymbol{w}_N^\star$ and $\boldsymbol{w}_S^\star$ denote their unique minimizers. Since $\mathcal{L}_N(\boldsymbol{w})$ is strictly convex, for a fixed δ there exists an ϵ such that

$$|\mathcal{L}_N(\boldsymbol{w}_N^\star) - \mathcal{L}_N(\boldsymbol{w}_S^\star)| \le \epsilon \implies \|\boldsymbol{w}_N^\star - \boldsymbol{w}_S^\star\|_2 \le \delta. \quad (16)$$

Clearly, ϵ is the infimum of $\mathcal{L}_N(\cdot)$ over the circle centered at $\boldsymbol{w}_N^\star$ with radius δ. Consequently, one can focus on bounding the objective function, and this will yield the desired bound on the solutions. We now introduce ℓ_2-norm bounds on data and weight vectors, and for any given input we assume that $\|\boldsymbol{x}\|_2^2 \le R_x$. Next, define the upper bound on weights such that $\max\{\|\boldsymbol{w}_N^\star\|_2^2, \|\boldsymbol{w}_S^\star\|_2^2\} \le R_w$. Note here that $R_w < \infty$ is guaranteed by the regularization. For $\boldsymbol{w}_1, \boldsymbol{w}_2$ define the following quantities

$$\Delta_{\boldsymbol{\Sigma}} = \boldsymbol{w}_1^\top (\boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N) \boldsymbol{w}_1 \ , \ \Delta_\sigma = (\boldsymbol{w}_1 - \boldsymbol{w}_2)^\top (\boldsymbol{\mu}_N - \boldsymbol{\mu}_S). \quad (17)$$

The main idea is to bound the difference between the objective functions in terms of $\Delta_{\boldsymbol{\Sigma}}$ and Δ_σ, which can then be bounded uniformly. In particular, we have the following result.

Lemma 1: Let $\Delta_{\boldsymbol{\Sigma}}$ and Δ_σ be defined as in Eq. (17) and

$\|\boldsymbol{w}_1\|_2^2, \|\boldsymbol{w}_2\|_2^2 \leq R_w$. For $\epsilon \geq 0$ the following hold

$$\text{(i)} \quad P\left(\sup_{\boldsymbol{w}_1} |\Delta_{\boldsymbol{\Sigma}}| \geq \epsilon\right) \leq 2d \exp\left\{-S \frac{\epsilon^2}{8\|\boldsymbol{\Sigma}_N\|_2 R_x R_w^2 + (16/3)\epsilon R_x R_w}\right\}$$

$$\text{(ii)} \quad P\left(\sup_{\boldsymbol{w}_1, \boldsymbol{w}_2} |\Delta_{\sigma}| \geq \epsilon\right) \leq 2 \exp\left\{-\frac{1}{2}\left(\frac{\epsilon\sqrt{S}}{2\sqrt{R_x R_w}} - 1\right)^2\right\}$$

We provide a proof in Appendix B. Both parts of this lemma use the unbiased statistics of $\boldsymbol{\mu}_N$ and $\boldsymbol{\Sigma}_N$, and matrix and vector concentration inequalities. This is made possible by the unbiased sampling procedure. Using Lemma 1 we can now state our main result.

Theorem 2: Let $\boldsymbol{w}_S^\star$ be the solution returned by MBA using S samples. For $\epsilon > 0$, if

$$S \geq \max\left\{\log(4d/p) \frac{\|\boldsymbol{\Sigma}_N\|_2 R_x R_w^2 + (1/3)\epsilon R_x R_w}{\epsilon^2/32}, \frac{4R_x R_w}{\epsilon^2}\left[\sqrt{2\log(4/p)} + 1\right]^2\right\} \tag{18}$$

then $\|\boldsymbol{w}_N^\star - \boldsymbol{w}_S^\star\|_2 \leq \delta$ with probability at least $1 - p$.

Proof: We define the following pointwise difference

$$\Delta(\boldsymbol{w}) = \mathcal{L}_S(\boldsymbol{w}) - \mathcal{L}_N(\boldsymbol{w}) \tag{19}$$

The starting point of the proof is the equality

$$\mathcal{L}_S(\boldsymbol{w}_N^\star) - \mathcal{L}_S(\boldsymbol{w}_S^\star) = \mathcal{L}_N(\boldsymbol{w}_N^\star) + \Delta(\boldsymbol{w}_N^\star) - \mathcal{L}_N(\boldsymbol{w}_S^\star) - \Delta(\boldsymbol{w}_S^\star) \ . \tag{20}$$

Here by construction

$$\mathcal{L}_N(\boldsymbol{w}_S^\star) - \mathcal{L}_N(\boldsymbol{w}_N^\star) = \mathcal{L}_N(\boldsymbol{w}_S^\star) - \arg\min_{\boldsymbol{w}} \mathcal{L}_N(\boldsymbol{w}) \geq 0 \tag{21}$$

$$\mathcal{L}_S(\boldsymbol{w}_N^\star) - \mathcal{L}_S(\boldsymbol{w}_S^\star) = \mathcal{L}_S(\boldsymbol{w}_N^\star) - \arg\min_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w}) \geq 0 \tag{22}$$

and we obtain

$$0 \leq \mathcal{L}_N(\boldsymbol{w}_S^\star) - \mathcal{L}_N(\boldsymbol{w}_N^\star) \leq \Delta(\boldsymbol{w}_N^\star) - \Delta(\boldsymbol{w}_S^\star) \ . \tag{23}$$

The left hand size of the inequality is a direct consequence of Eq. (21). For the right-hand side note that Eq. (20) can be manipulated as

$$\begin{aligned}\mathcal{L}_N(\boldsymbol{w}_S^\star) - \mathcal{L}_N(\boldsymbol{w}_N^\star) &= \Delta(\boldsymbol{w}_N^\star) - \Delta(\boldsymbol{w}_S^\star) \\ &\quad + [\mathcal{L}_S(\boldsymbol{w}_S^\star) - \mathcal{L}_S(\boldsymbol{w}_N^\star)] \\ &\leq \Delta(\boldsymbol{w}_N^\star) - \Delta(\boldsymbol{w}_S^\star)\end{aligned} \tag{24}$$

where the second line follows from $[\mathcal{L}_S(\boldsymbol{w}_S^\star) - \mathcal{L}_S(\boldsymbol{w}_N^\star)] < 0$ due to Eq. (22).

It is therefore sufficient to show that $\Delta(\boldsymbol{w}_N^\star) - \Delta(\boldsymbol{w}_S^\star) \leq \epsilon$ with high probability. Further expand this bounding term as

$$\Delta(\boldsymbol{w}_N^\star) - \Delta(\boldsymbol{w}_S^\star) = \frac{1}{2}\boldsymbol{w}_N^{\star\top}\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_N^\star - \frac{1}{2}\boldsymbol{w}_S^{\star\top}\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_S^\star + \boldsymbol{w}_N^{\star\top}\Delta_{\boldsymbol{\mu}} - \boldsymbol{w}_S^{\star\top}\Delta_{\boldsymbol{\mu}} \ , \tag{25}$$

and consider uniformly bounding the following two terms

- Quadratic: $\left|\frac{1}{2}\boldsymbol{w}_N^{\star\top}\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_N^\star - \frac{1}{2}\boldsymbol{w}_S^{\star\top}\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_S^\star\right| \leq \frac{\epsilon}{2}$
- Linear: $\left|\boldsymbol{w}_S^{\star\top}\Delta_{\boldsymbol{\mu}} - \boldsymbol{w}_N^{\star\top}\Delta_{\boldsymbol{\mu}}\right| \leq \frac{\epsilon}{2}$

with probability at least $1 - \delta/2$.

Note that both terms have weight vectors coupled with samples so we cannot apply concentration inequalities directly. However, they can be upper bounded by the expressions in Lemma 1. In particular, the quadratic term is upper bounded by $\sup_{\boldsymbol{w}_1} |\Delta_{\boldsymbol{\Sigma}}|$. We then apply Lemma 1(i) with threshold $\epsilon/2$ and probability $p/2$ which yields the first term in Eq. (18). Similarly the linear term is upper bounded by $\sup_{\boldsymbol{w}_1, \boldsymbol{w}_2} |\Delta_{\sigma}|$. Applying Lemma 1(ii) with threshold $\epsilon/2$ and probability $p/2$ we get the second term in Eq. (18). □

Theorem 2 shows that the sample size required to guarantee $|\mathcal{L}_N(\boldsymbol{w}_N^\star) - \mathcal{L}_N(\boldsymbol{w}_S^\star)| \leq \epsilon$ with high probability does not depend on the total number of pairs. Instead it grows logarithmically with the feature size. This result is useful in that, even though the total number of pairs in the data is too large, randomly sampling a small fraction guarantees a loss value that is close to the optimum. The same sample size also guarantees $\|\boldsymbol{w}_N^\star - \boldsymbol{w}_S^\star\|_2 \leq \delta$ with high probability. (The usefulness of this latter result also depends on the reverse Lipschitz condition.)

It might at first seem unreasonable that S does not depend on N; but only on the input dimension d. The reason for this favorable result is that the given samples are *fixed* and we are uniformly subsampling from them. So in this context N is actually the support size of a discrete uniform distribution on samples. Since concentration inequalities do not depend on the support size, we consequently do not have N in Eq. (18).

While linear models often work well, in practice we can have datasets that are not linearly separable. In such cases one is typically concerned with devising a nonlinear feature transform to obtain linear separability. In fact, for finite-dimensional transforms Theorem 2 readily extends. Such transforms include, for example, polynomial features and conjunctions, random Fourier features [42], and random shallow neural networks [43]. In more abstract terms, all these transformations are mappings from d dimensions to F dimensions. Given a fixed transformation, the result of Theorem 2 still holds, where we replace d by F. Therefore, when the input space is not good for a linear ranking function, we can first apply feature engineering, and then direcly use MBA.

IV. Experiments

In this section we conduct four types of experiments to demonstrate the performance benefits of MBA. In the first part we use simulation data from Gaussian mixtures to investigate the performance of various methods. Furthermore, since we know the distribution generating the data here, we can also use the Neyman-Pearson decision rule, which is theoretically optimum (we discuss this in Appendix A). Here we show that MBA can achieve better performance with a low number of samples, corroborating the theoretical analysis. For the second part, we experiment on 15 frequently used benchmark datasets

from the UCI[2] and LIBSVM[3] repositories; these datasets cover a wide range of application domains and show MBA performs better than the competing methods overall. We then use the same datasets in the third part to demonstrate how the linear framework can be extended to account for nonlinear features. Finally we consider a large scale click through rate (CTR) prediction problem with two publicly available industrial-sized datasets with tens of millions of samples.

For comparison we use the following algorithms: MBA-ℓ_2, MBA-ℓ_1, MBA-EL, which represent the three variants of our mini-batch AUC optimization using ridge regression, lasso, and elastic net respectively. OLR is simply the online logistic regression and SOLR is the sparse regression algorithm presented in [11]. OAM is the first proposed online AUC maximization algorithm using stochastic gradients [19] which uses PHL. On the other hand, AdaAUC is the adaptive gradient AUC maximization algorithm in [21]. This algorithm is proposed as an improvement to the one pass AUC optimization algorithm in [20]. AdaAUC has better performance empirically, so we use this version in our comparisons. Both algorithms are based on PSL. In addition, we implement two mini-batch stochastic gradient algorithms for large scale CTR prediction problems. One is MB-PHL, a mini-batch gradient descent algorithm which uses PHL. A variant of this approach is also proposed in the recent work of [24]. MB-PSL is another mini-batch gradient method that uses PSL. MB-PHL and MB-PSL can be thought of a substitutions for OAM and AdaAUC for larger datasets. This is necessary since for a large number of inputs, sequential processing becomes inefficient. All implementations are done in Python with NumPy and Scikit-Learn libraries.

TABLE I
SUMMARY STATISTICS OF DATASETS USED IN EXPERIMENTS. FOR EACH WE SHOW THE TRAIN | TEST SAMPLE SIZE, FEATURE SIZE, AND THE RATIO OF NEGATIVE SAMPLES TO POSITIVE SAMPLES IN THE TRAINING SET.

Dataset	# Samp.	# Feat.	T_- / T_+
a1a	1.6K | 30.9K	123	3.06
a9a	32.5K | 16.2K	123	3.15
amazon	750 | 750	10,000	2.33
bank	20.6K | 20.6K	100	7.88
codrna	29.8K | 29.8K	8	2.00
german	500 | 500	24	2.33
ijcnn	50K | 92K	22	9.30
madelon	2,000 | 600	500	1.00
mnist	60K | 10K	780	2.30
mushrooms	4K | 4K	112	0.93
phishing	5.5K | 5.5K	68	0.79
svmguide3	642 | 642	21	2.80
usps	7.2K | 2K	256	2.61
w1a	2.5K | 47.2K	300	33.40
w7a	25K | 25K	300	32.40
avazu app	12.6M | 2M	10,000	8.33
avazu site	23.6M | 2.6M	10,000	4.06
criteo	45.8M | 6M	10,000	2.92

[2]archive.ics.uci.edu/ml/

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

A. Simulation Study

For the simulations we consider the binary hypothesis testing problem of Eq. (28), which can represent, for example, a radar or communication channel setting. In particular, we employ Gaussian mixtures as data generating distributions. Namely, for hypothesis-i a K-component Gaussian mixture is given by

$$p_i(\boldsymbol{x}) = \sum_{k=1}^{K} c_{ik}\ (2\pi)^{-d/2}\ |\boldsymbol{\Sigma}_{ik}|^{-1/2} \times \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_{ik})^\top \boldsymbol{\Sigma}_{ik}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{ik})\right\}\ , \qquad (26)$$

where the weights c_{ik} are convex combination coefficients to ensure the function is a valid pdf. This distribution is completely characterized by the weights, means, and covariances. For hypothesis-i let c_i, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ denote the *set* of these parameters. For our experiments $k \in \{1, 2, 3\}$ and:

- $k = 1$: We set $c_0 = \{1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}\}$ and $c_1 = \{1\}$, $\boldsymbol{\mu}_1 = \{\mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}\}$.
- $k = 2$: We set $c_0 = \{0.9, 0.1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}, \boldsymbol{I}\}$ and $c_1 = \{0.1, 0.9\}$, $\boldsymbol{\mu}_1 = \{-\mathbf{0.1}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}, \boldsymbol{I}\}$.
- $k = 3$: We set $c_0 = \{0.8, 0.1, 0.1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}, \mathbf{0}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{I}\}$ and $c_1 = \{0.1, 0.1, 0.8\}$, $\boldsymbol{\mu}_1 = \{-\mathbf{0.1}, \mathbf{0}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{I}\}$

As can be seen, the distributions we choose for the two hypotheses are symmetric across the origin. As the number of components increase the distributions get less interspersed, making separation more challenging. All covariances are set to be unit-variance and isotropic. We finally note that the bold numbers for the mean sets correspond to a vector of the bold entry replicated. For our experiments, for each value of k we form 50 training sets, where for each dataset we sample 20,000 points from the generative distribution. Furthermore we create an imbalanced dataset, where roughly 90% of the data has label 0 (i.e. sampled from hypothesis 0). We also create a separate test set, with 100,000 samples and the same imbalance ratio.

Since the generative distributions are assumed known in this setting, we can analytically derive the Neyman Pearson (NP) decision rule which maximizes the AUC. For any given k, the NP rule computes the scores through $p_1(\boldsymbol{x})/p_0(\boldsymbol{x})$. Note that, this does not require any training data, as the generating distributions are already known. A particularly interesting case is $k = 1$. Here the log likelihood is

$$\log \frac{p_1(\boldsymbol{x})}{p_0(\boldsymbol{x})} = \boldsymbol{x}^\top(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \frac{1}{2}(\boldsymbol{\mu}_0^\top \boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^\top \boldsymbol{\mu}_1) \qquad (27)$$

Since the constant term does not affect AUC, we see that the optimum ranking rule is a linear function of $\boldsymbol{x}$. So in this special case we have a consistent learning problem where the optimum ranking function is contained within the hypothesis set of linear functions [34].

For the comparisons in this section we use MBA-ℓ_2, ONLR, and AdaAUC, as the latter two typically have the best competitive performance against the former. We run the experiments for three different sample ratio (SR), namely SR $\in \{1\%, 10\%, 100\%\}$; this represents the percentage of available data points used for training. As mentioned above we average

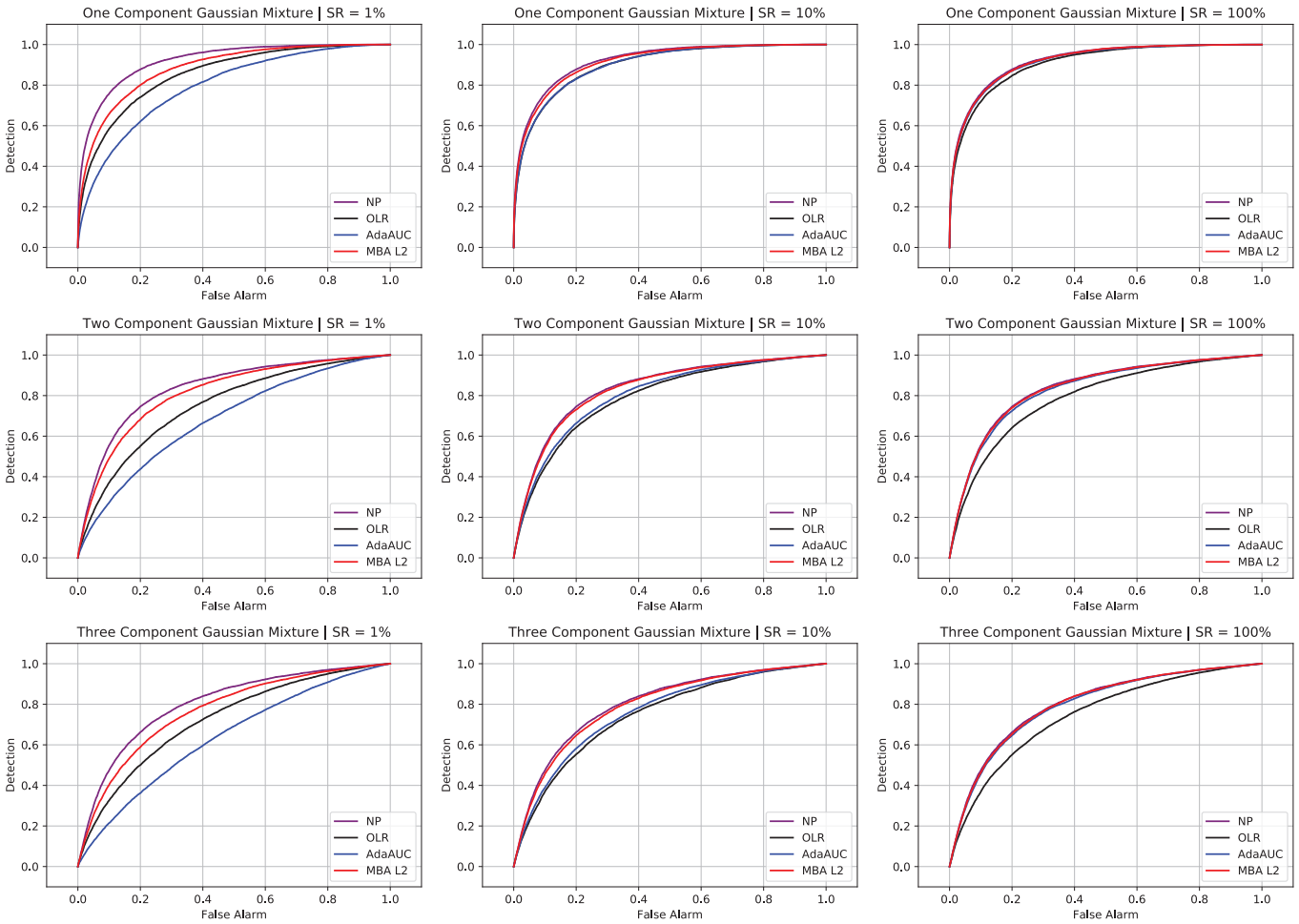

Fig. 1. The ROC curves obtained by the Neyman-Pearson detector and three learning algorithms on the simulated data. The rows are in increasing order of mixture components (k) and the columns are in increasing order of sample ratio (SR).

TABLE II

COMPARISONS OF ALGORITHMS ON SIMULATED DATA, THE PERFORMANCE OF MBA-ℓ_2, ONLR, AND ADAAUC ARE REPORTED FOR $k \in \{1, 2, 3\}$ AND SR $\in \{1\%, 10\%, 100\%\}$. THE FILLED CIRCLE INDICATES THAT MBA IS (STATISTICALLY) SIGNIFICANTLY BETTER.

Distribution	SR	Neyman-Pearson	MBA-ℓ_2	ONLR	AdaAUC
1-Component Gaussian Mixture	1 %	92.13	87.43 ± 0.69	• 86.79 ± 0.99	• 80.94 ± 2.77
	10 %		91.44 ± 0.10	• 90.54 ± 0.29	• 90.25 ± 0.31
	100 %		91.88 ± 0.04	• 90.69 ± 0.24	• 91.70 ± 0.07
2-Component Gaussian Mixture	1 %	83.71	80.15 ± 0.68	• 77.64 ± 1.41	• 69.75 ± 3.24
	10 %		83.15 ± 0.10	• 80.19 ± 0.69	• 80.12 ± 0.69
	100 %		83.47 ± 0.04	• 80.19 ± 0.69	• 83.01 ± 0.11
3-Component Gaussian Mixture	1 %	80.22	76.39 ± 0.47	• 73.07 ± 1.06	• 66.38 ± 1.95
	10 %		79.52 ± 0.11	• 75.94 ± 0.64	• 76.72 ± 0.33
	100 %		79.93 ± 0.11	• 75.91 ± 0.83	• 79.61 ± 0.06

the generalization performance over 50 training samples, and report the average values and standard deviations. Table II shows the resulting AUC values and Figure 1 displays the corresponding ROC curves.

Firstly note that, as k increases, the AUC value achieved by the optimal NP rule decreases; this shows that adding more mixture components progressively make the problem harder. As we increase the SR, all three learning algorithms improve, as expected. However, we can see that the starting point for MBA is significantly higher than the the other two. In particular, AdaAUC is worse for small sample sizes. This shows the difficulty of optimizing a bivariate loss function as opposed to the univariate logistic loss of ONLR. The particular difficulty comes from selecting the step size, and for smaller number of samples the stochastic gradient cannot efficiently optimize.

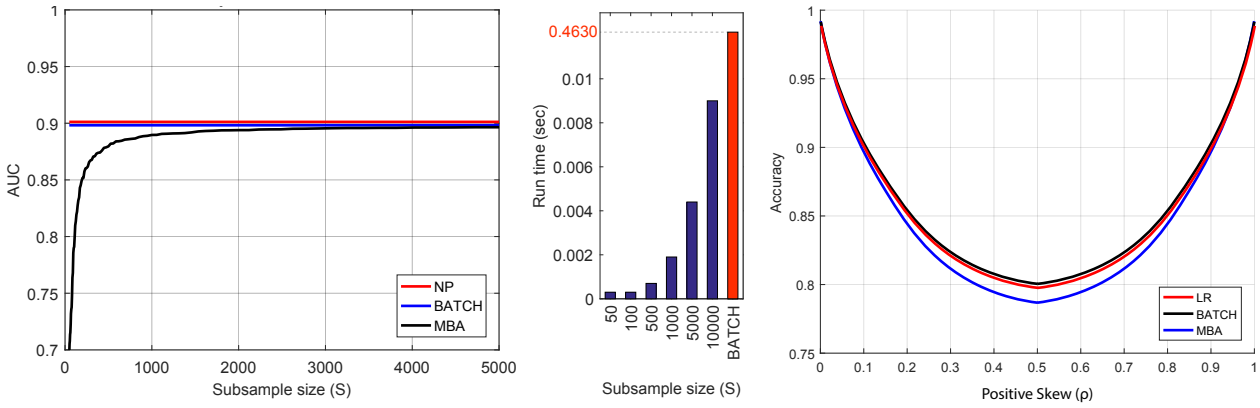

Fig. 2. Comparison of MBA to the batch algorithm using three component Gaussian mixture distribution. Left panel: Generalization AUC of MBA as a function of sample size. Mid panel: Runtime comparison. Right panel: Classification performance of MBA and the corresponding batch algorithm with respect to batch logistic regression.

Being learning rate and gradient free, MBA does not suffer from these drawbacks and always performs better than ONLR.

We also use pairwise t-test to assess the statistical significance of results, using 95% confidence level, as proposed and used by [20] initially for this problem. For all cases considered we see that MBA achieves better results than its competitors with significance. This is true even when $SR = 100\%$ and the average values are close, as the standard deviations are low and a large number of experiments are performed.

Interestingly, comparing the performance of NP and MBA we see that in all three cases the achieved AUC is quite close. This is the case even when $k > 1$; therefore even if the optimal scores are a nonlinear function of $\boldsymbol{x}$ for these cases, the linear approximation is still reasonable. With that said, if the data is highly nonlinear it might be difficult to find a good separating hyperplane.

We also compare MBA to its batch version—i.e. the case where all the pairs are used. To do so we use the three-component Gaussian mixture distribution, where 10^3 positive and 10^3 negative samples are generated. The batch algorithm uses all 10^6 pairs, whereas for MBA we consider various sample sizes. Figure 2 displays the results of this experiment. In the left panel we compare the generalization AUC of the optimal NP decision rule, the batch algorithm, and MBA (as a function of sample size). It can be seen that the NP rule and batch algorithm give very similar results. As the sample size increases MBA quickly catches up with the other two, and around $S = 5,000$ the difference becomes indistinguishable. This is a significant reduction in sample cost, which corresponds to a runtime speed-up of more than 50 times, as shown in the middle panel. We also show how MBA can be extended to a classifier. Here the label cut-off is selected by the ratio in training set. In terms of generalization accuracy, the batch version of MBA and a batch implementation of logistic regression give similar results. For comparison we use MBA with $S = 500$ samples. While this performs slightly worse than the batch algorithm, once again there is a significant improvement in sample complexity.

B. UCI and LIBSVM Benchmark Data

In this section we experiment with 15 frequently used benchmark datasets from the UCI and LIBSVM repositories (see Table I). It can be seen that the chosen datasets cover a wide range of sample/feature sizes. The distribution of samples vary from being linearly separable to highly nonlinear. The datasets also exhibit significant differences in label imbalance. In terms of the features present, the data fall into three categories: numerical only, categorical only, and mixed. We use the train/test splits provided in LIBSVM website; if splits are not available we use 50/50 splitting with stratification. For multiclass datasets we map the classes to binary labels.

Table III shows the AUC values obtained by six competing algorithms on benchmark datasets. Here the results are reported along with standard deviations. In addition, we once again conduct a pairwise t-test with 95% significance level. To perform this test, we compare each algorithm in the last four columns to the two MBA algorithms in the first two columns. If MBA performs significantly better/worse we represent this with a filled/empty circle. Table III shows that there is a clear benefit in using the proposed MBA, whereas the recent AdaAUC is the second best competitor. The mini-batch processing phase of MBA is learning-rate free and this brings an important advantage. AdaAUC adapts the gradient steps, while we used the learning rate $\mathcal{O}(1/\sqrt{t})$ for all other stochastic gradient algorithms, which performed well with this choice. However, though MBA does not need this parameter, it still performs significantly better than AdaAUC in 9/15 cases. It is also worth noting that MBA-ℓ_1 obtains 100% AUC for the mushrooms data, and for the svmguide3 dataset MBA is at least 9% better than the others. While logistic regression does not directly optimize AUC, it is frequently used in practice, where AUC is the main performance metric. Here we see that logistic regression has a decent performance, and in fact beats MBA on the a9a data.

Another important performance measure is the ability to rank as a function of sample size. We show comparisons for this in Figure 3. We see that the stochastic gradient based methods keep improving as the sample size increases, whereas MBA

TABLE III
COMPARISON OF ALGORITHMS ON 15 BENCHMARK DATASETS FROM UCI AND LIBSVM REPOSITORIES. THE SYMBOLS FILLED/EMPTY CIRCLE INDICATE ONE OF THE MBA IS (STATISTICALLY) SIGNIFICANTLY BETTER/WORSE.

Dataset	MBA-ℓ_2	MBA-ℓ_1	OLR	SOLR	OAM	AdaAUC
a1a	88.98 ± 0.14	88.67 ± 0.15	• 88.64 ± 0.27	• 88.04 ± 0.14	• 87.61 ± 0.45	• 88.51 ± 0.34
a9a	89.97 ± 0.01	89.97 ± 0.02	◦ 90.17 ± 0.03	• 89.88 ± 0.03	• 89.30 ± 0.22	89.99 ± 0.04
amazon	77.12 ± 0.44	71.35 ± 2.50	• 69.90 ± 2.21	• 71.87 ± 0.74	• 60.23 ± 3.90	• 74.97 ± 0.89
bank	93.22 ± 0.06	93.22 ± 0.03	• 82.89 ± 0.21	• 80.23 ± 0.33	• 81.51 ± 0.49	• 89.46 ± 0.12
codrna	97.68 ± 0.00	97.63 ± 0.01	• 95.69 ± 0.19	• 92.36 ± 0.85	• 97.31 ± 0.12	• 94.34 ± 0.40
german	80.34 ± 0.80	80.41 ± 0.64	• 76.39 ± 1.69	• 75.07 ± 1.22	• 74.59 ± 1.79	• 77.83 ± 1.29
ijcnn	90.53 ± 0.05	90.40 ± 0.07	• 89.50 ± 0.53	• 88.93 ± 0.48	• 88.52 ± 1.76	90.59 ± 0.28
madelon	62.39 ± 0.44	62.34 ± 0.51	61.97 ± 0.69	• 61.81 ± 0.48	• 60.64 ± 0.44	61.82 ± 1.58
mnist	95.81 ± 0.02	95.77 ± 0.02	• 95.63 ± 0.27	• 95.49 ± 0.12	• 94.82 ± 0.23	• 95.47 ± 0.09
mushrooms	100.00 ± 0.00	100.00 ± 0.00	99.88 ± 0.03	• 99.73 ± 0.07	• 99.62 ± 0.28	• 99.98 ± 0.00
phishing	98.32 ± 0.01	98.32 ± 0.05	98.49 ± 0.01	98.38 ± 0.03	• 98.08 ± 0.27	98.36 ± 0.02
svmguide3	81.16 ± 0.80	82.05 ± 0.66	• 63.80 ± 0.81	• 57.65 ± 2.98	• 66.97 ± 3.45	• 69.14 ± 1.95
usps	95.89 ± 0.04	95.83 ± 0.06	• 95.82 ± 0.15	• 95.71 ± 0.07	• 94.65 ± 0.53	• 95.74 ± 0.13
w1a	92.28 ± 0.23	91.21 ± 0.36	• 84.81 ± 1.34	• 79.84 ± 1.15	• 87.83 ± 1.59	• 90.70 ± 0.67
w7a	96.27 ± 0.07	96.17 ± 0.08	• 93.05 ± 0.29	• 89.27 ± 0.82	• 93.92 ± 0.55	• 95.09 ± 0.26
Win/Tie/Loss	-	-	11/3/1	14/1/0	15/0/0	11/4/0

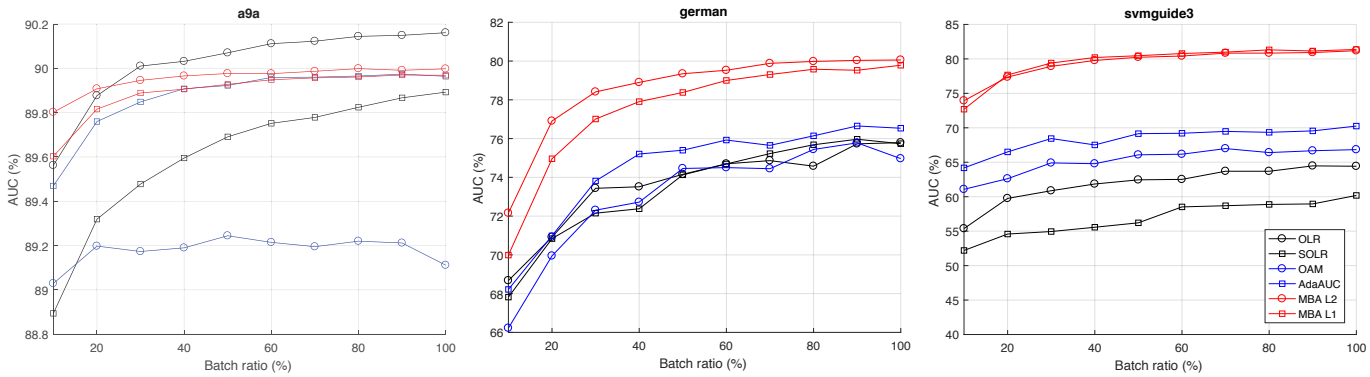

Fig. 3. AUC performance of six algorithms as a function of sample size for a9a, german, and svmguide3 selected from LIBSVM.

has a relatively steady performance, and it converges faster. This indicates that for these benchmark datasets MBA can already construct a good approximation of the global problem at this point. This result is not surprising given Theorem 4, as good performance is independent of the number of pairs or instances, and only related to the dimensionality of the optimization problem to be solved. In the first panel of Figure 3 the best performer is logistic regression although the difference is rather small. In the second plot, MBA-ℓ_2 gives the best result, although AdaAUC is good as well. For the other two plots, both MBA methods have a clear advantage.

Finally we provide runtime comparisons in Figure 4, shown in log scale where the smallest runtime is mapped to 1 in order to compare across different datasets. For 13 cases MBA has the lowest runtime, which shows its advantage in terms of sample complexity. Clearly, the case where it performs much slower is the Amazon dataset, where we have 10,000 dense features. For this case the computation bottleneck is the covariance matrix computations, which is also an issue for AdaAUC. (In general, PSL-based methods will be slower in this dataset.) However in many practical cases we can either apply dimensionality reduction, or sparsity can be exploited as we show later.

C. Nonlinear Features

So far we have demonstrated how MBA can achieve better performance than its competitors on learning a linear ranking function. As mentioned in Section III, MBA can easily be extended to nonlinear feature spaces as well. Here we show an application using three datasets from the previous section: german, mnist, and svmguide3. Our baseline will be the best values obtained by the linear MBA (LIN) from table III. We obtain nonlinear features using a variety of methods: random thresholds to obtain binary features (RT) [43], random Fourier features (RFF) [42] which approximate kernel machines, random neural networks (RNN) where the weights are obtained via randomization, and finally a single layer binary classification network (NN) which is learned from the training data. Note that in the latter case we can also use the network itself to get scores, however we instead feed the representations in the hidden layer to MBA, for demonstration purposes.

For the german and svmguide datasets we use 500 random features, whereas mnist uses 1000 as it has higher dimensionality. For the NN the hidden layer number is half of the input dimension, and both RNN and NN use hyperbolic tangent

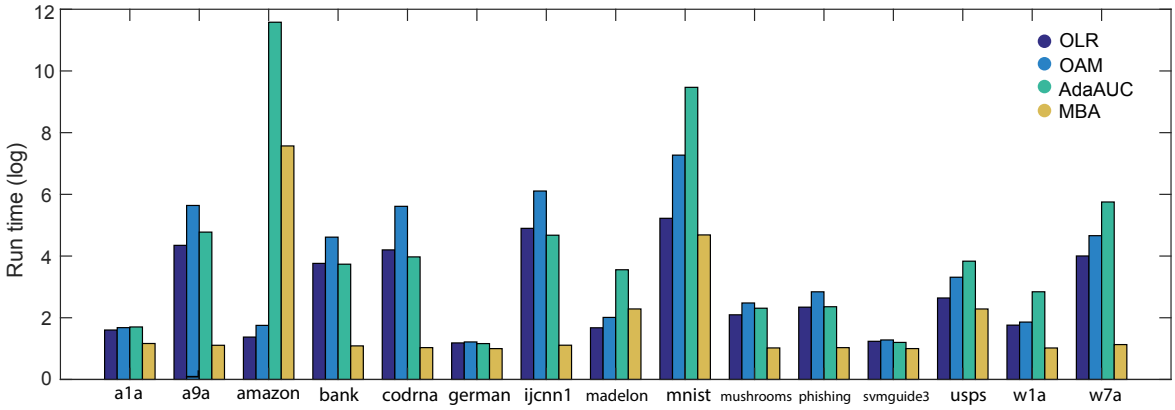

Fig. 4. Runtime (log) comparison of MBA and three competing algorithms on 15 UCI/LIBSVM datasets.

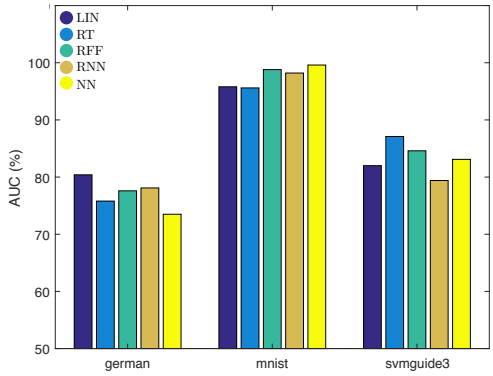

Fig. 5. AUC performance of four nonlinear feature generation methods, compared to the linear case. All of the training is done via MBA-L2.

activation function. NN is trained using Adam optimizer [44]. Given the nonlinear features all the ranking functions are learned with MBA-L2.

We report the AUC test values in Figure 5. For the german dataset we see that the linear model has the best performance, and applying a nonlinear feature transform actually degrades performance. For the mnist dataset, we see that while RT does not provide an improvement, RFF, RNN, and NN all yield substantially higher AUC values. This dataset is a widely used benchmark for training neural networks [45] and the features learned by neural nets are highly effective. The best performance in this case is given by NN, which shows that learning representations in a data-dependent manner is the best choice here. Finally, for the svmguide3 dataset we see that the best performance is given by RT. While RFF and NN also provide improvement, RT has a clear edge here. (This reinforces our previous claim that there is not a one-shot solution to the feature engineering problem.)

D. One-Shot Image Recognition

We now describe how MBA can be applied in the context of image recognition. As this approach is gradient-free, it is particularly suitable for cases where the number of training samples is low. For this reason we consider the one-shot image recognition problem using the Omniglot dataset [12]. This dataset consists of 50 different alphabets, splitted as 30/20 for training/testing. We consider two settings; in the first one we use use linear features obtained by applying PCA to the $105,000$ dimensional inputs, resulting in a feature size of 512. For the second one we feed these features into a Siamese Neural Network (NN), as described in [12]. Two layers are used to obtain the final feature representation, which also has a dimension of 512. The Siamese NN is trained on the 30 training alphabets, while no training is necessary for the linear case. For testing, MBA is applied either to test images with the same PCA-based decomposition or Siamese NN features. As each character contains a total of 20 examples, 10 is used to train MBA and the other 10 is used for testing. Regularization parameter is learned by leave-one-out cross-validation on training examples.

Figure 6 shows the results of this experiment. A total number of 2500 pairs are sampled uniformly from all possible combinations of distinct characters, which provides a sufficient size for reliable AUC comparison. Overall, the AUC values for the Linear and Siamese NN features are 0.9527 and 0.9555, in that order. Applying the Siamese network provides a limited improvement and it also requires an initial training over the set of 30 alphabets. The first two panels of Figure 6 shows AUC values for all pairs, organized in a 50×50 grid. The last panel of this Figure shows the top 15 character pairs where the features learned by the Siamese NN provide the largest improvement; it can be seen that these correspond to pairs with high visual simiarity. Therefore, the Siamese network is useful in providing representations that render different but similar looking characters distinguishable for MBA. The top improvement obtained is approximately 30%.

The approach here can also be extended to other computer vision applications, for example in deep learning based image cropping [46] it can be used for fast and efficient comparisons of the candidate frames.

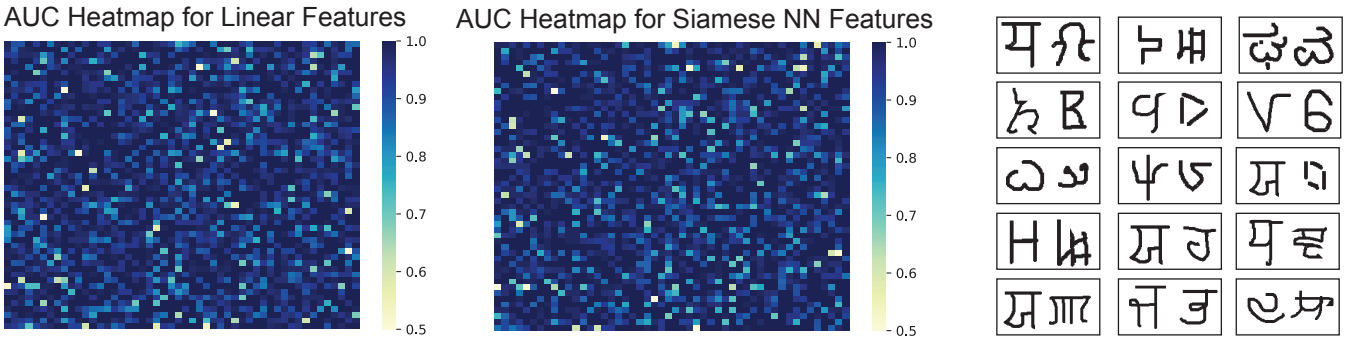

Fig. 6. One-shot image recognition experiments with the Omniglot dataset. The first two panels show AUC values on a total number of 2500 uniformly drawn pairs of characters, arranged in a 50×50 grid, for the Linear and Siamese NN features respectively. The last panel shows the top 15 character pairs where the features learned by the Siamese NN provide the largest improvement.

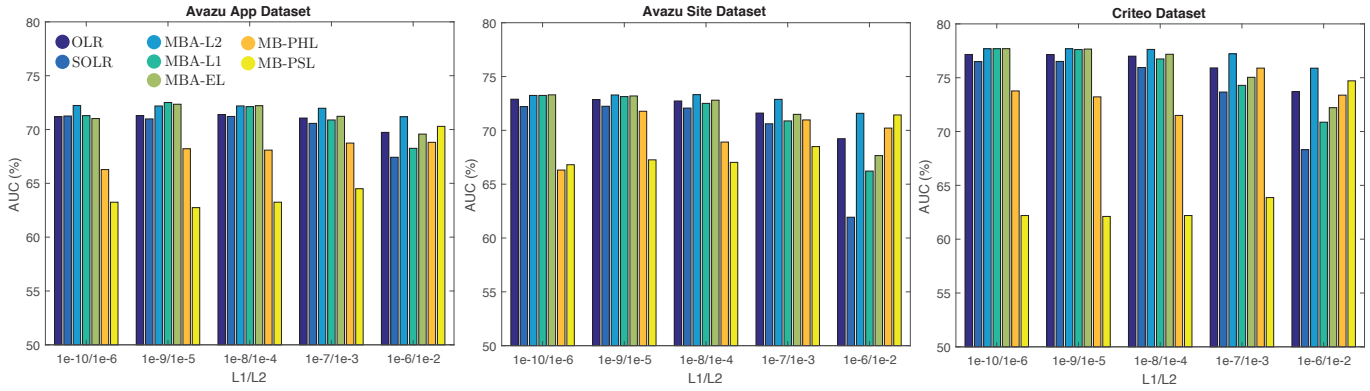

Fig. 7. AUC achieved by all algorithms on the Avazu App, Avazu Site, and Criteo datasets. Here the performance is plotted as a function of regularization parameters. The elastic net uses one half of ℓ_1-penalty for both ℓ_1 and ℓ_2 regularization.

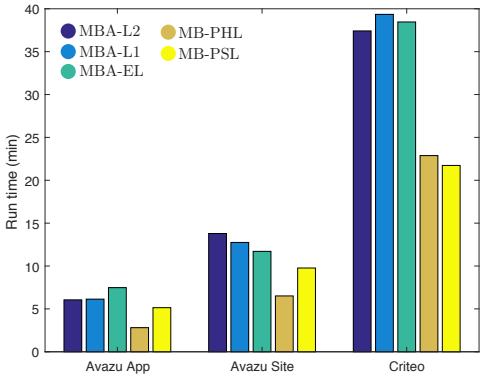

Fig. 8. Runtime comparison of MBA with MB-PSL and MB-PHL. As the latter two only require a gradient computation they are faster than MBA, but with significantly reduced performance. On the other hand MBA can process ten million samples under an hour, which shows the scalability of this approach.

E. Large-scale Web Click Data

For the last experiment we use a large scale click through rate (CTR) prediction dataset. Estimating user clicks in web advertising is one of the major areas of AUC optimization since the number of users who click a given ad is typically very low. The datasets used come from Avazu and Criteo, available at LIBSVM (see Table I). It is worth noting that these datasets are an order of magnitude larger than the ones used in previous studies, showing the scaling benefit MBA brings. This time we shuffle and split the entire dataset into chunks of 100 (Avazu App) and 200 (Avazu Site and Criteo). We then make a single pass over these chunks with randomized sampling and report the results. For these datasets the variation across different runs is very small. Therefore we do not show the confidence intervals in the bar charts, but note that all results are statistically significant. For the CTR problem, a 0.1% improvement in AUC is considered significant, while an increase of 0.5% results in noticable revenue gain.

In Figure 7 we show the AUC performance of seven algorithms. For the Avazu App data, MBA-ℓ_2 gives the best results while for Avazu Site and Criteo all MBA algorithms give similar results. Comparing the proposed MBA with the best non-MBA algorithm, the performance improvements are 1.20%, 0.43% and 0.54%. The mini-batch gradient descent algorithms do not perform as well, especially when the regularization parameter is small, and they get better as this parameter increases. For these experiments the step size is $\mathcal{O}(1/\sqrt{t})$, and while logistic regression has good performance with this choice, optimizing pairwise losses seems less robust. As the

regularization increases the variation in the gradients decreases, which helps improve the AUC scores. We also experiment with a small constant step size, and this yields similar results. On the other hand MBA does not require this parameter at all. For this reason our algorithm is well-suited for large-scale problems.

Another important concern is the running time. Here, we do not make a relative comparison, but instead state how much time it takes to find the result. This is because comparing the running time to logistic regression is not very informative; if a sequential logistic regression is implemented in Python script, then the mini-batch algorithm is roughly 10 times faster, as sequential processing is slow. However, if an optimized package is used, then it can be 100 times faster than MBA, as the underlying code is optimized. For this reason we show the running time of the vanilla implementation of MBA in Figure 8. As it can be seen, even for the Criteo dataset, which contains the largest number of instances, the runtime is under an hour. As we briefly mentioned in Section III, the mini-batch portion of MBA can be distributed without loss of accuracy, therefore using cluster computing, MBA can easily scale to billion-sample datasets, which are several orders of magnitude larger than the datasets that can be handled by sequential methods.

V. Conclusion

This paper has introduced a fast, scalable algorithm to optimize the AUC metric. Our proposed approach, called MBA, uses the specific structure of the squared pairwise surrogate loss function. In particular, it is shown that one can approximate the global risk minimization problem simply by approximating the first and second moments of pairwise differences of positive and negative inputs. This suggests an efficient mini-batch scheme, where the moments are estimated by U-statistics. MBA comes with theoretical guarantees, and importantly the number of samples required for good performance is independent of the number of pairs present, which is typically a very large number. Our experiments demonstrate the advantages of MBA in terms of speed and performance. We think MBA would be particularly useful for applications where AUC is the prime metric, the data size is massive and parallel processing is necessary.

Appendix A
AUC Optimization and Signal Detection

In this section we discuss the connection of AUC optimization to the signal detection framework. This framework is concerned with a probabilistic setup, where the optimal solution with maximum AUC can be obtained in analytical form. This is in contrast to the statistical learning setup which assumes that the probability distributions generating the observations are unknown to the modeler.

In binary signal detection we have two hypotheses

$$\mathcal{H}^+ \;:\; \boldsymbol{X} \sim \mathcal{P}^+ \quad , \quad \mathcal{H}^- \;:\; \boldsymbol{X} \sim \mathcal{P}^- \tag{28}$$

This setting arises frequently in many different applications, such as radar systems and communication channels [1]. In this setup, it is commonly assumed that the generating distributions

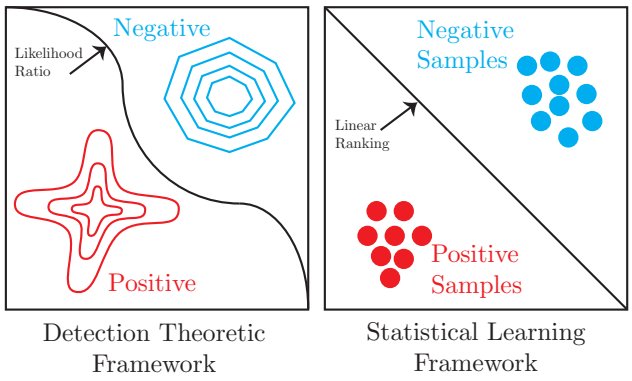

Fig. 9. A cartoon illustration of the signal detection (left) and statistical learning (right) frameworks for AUC maximization.

$\mathcal{P}^+$ and $\mathcal{P}^-$ are known (c.f. Eq. (1)). For our purposes we consider the *Neyman-Pearson (NP)* hypothesis testing scenario, where neither the priors for hypotheses are known, nor the costs of making a wrong decision are directly available. In this case the optimal detector is designed based on the following two metrics: detection and false alarm, defined for a fixed decision rule as

$$\begin{aligned} \text{Detection: } P_D(\Gamma) &= \int \Gamma(\boldsymbol{x})\; \mathcal{P}^+(\boldsymbol{x})\; d\boldsymbol{x} \\ \text{False Alarm: } P_F(\Gamma) &= \int \Gamma(\boldsymbol{x})\; \mathcal{P}^-(\boldsymbol{x})\; d\boldsymbol{x} \end{aligned} \tag{29}$$

Two immediate observations follow: (i) The metrics measure the performance of the rule itself, so they are a function of Γ. (ii) The detector $\Gamma(\boldsymbol{x})$, in turn, is a mapping from the observed signal $\boldsymbol{x}$ to the hypotheses/labels $\{+,-\}$. As the names imply, detection is the probability of correctly choosing the positive hypothesis, whereas false alarm is choosing the positive whereas the correct hypothesis was the negative. In general, the positive hypothesis corresponds to the presence of a target/message, whereas the negative one indicates absence, hence the names.

In NP hypothesis testing, the optimal detector is the solution to the optimization

$$\Gamma'(\boldsymbol{x}) := \arg\max_{\Gamma} P_D(\Gamma) \quad s.t. \quad P_F(\Gamma) \leq \alpha \;. \tag{30}$$

We therefore seek the detector with highest detection probability while setting a limit on the false alarm rate ($0 \leq \alpha \leq 1$). Note that, without this limit (i.e. $\alpha = 1$) we can use a trivial decision rule that maps all observations to positive hypothesis and obtain $P_D(\Gamma) = 1$. The solution of this optimization is given by the following.

Lemma 1 (Neyman-Pearson, [1]): For α, let Γ be any decision rule with $P_D(\Gamma) \leq \alpha$ and let Γ' be the decision rule of form

$$\Gamma'(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad p_1(\boldsymbol{x}) > \eta\; p_0(\boldsymbol{x}) \\ \gamma(x) & \text{if} \quad p_1(\boldsymbol{x}) = \eta\; p_0(\boldsymbol{x}) \\ 0 & \text{if} \quad p_1(\boldsymbol{x}) < \eta\; p_0(\boldsymbol{x}) \end{cases} \tag{31}$$

where $\eta \geq 0$ and $0 \leq \gamma(\boldsymbol{x}) \leq 1$ are chosen such that $P_F(\Gamma') = \alpha$. Then $P_D(\Gamma') \geq P_D(\Gamma)$.

We note that a decision rule that is optimal in the NP sense satisfies the false alarm inequality on the boundary. The structure in Eq. (31) reveals that, for any given input $\boldsymbol{x}$ this rule computes a score based on the likelihood ratio $p_1(x)/p_0(x)$ and compares it to a threshold. Since the ROC curve is the plot of detection vs false alarm, when $\mathcal{P}^+$ and $\mathcal{P}^-$ are known, the likelihood ratio function yields maximum AUC.

The framework outlined in this section is displayed in Figure 9, left panel. In general, the likelihood ratio test would yield non-linear decision boundaries. On the other hand, the right panel of Figure 9 displays the statistical learning approach to AUC optimization, where the ϕ-risk is minimized under the linear classifier assumption, i.e. this is what the MBA does.

In contrast to the signal detection problem of left panel, here we only have access to samples, instead of the class-conditional densities. As the figure suggests, the linear scoring function assumption is meaningful when the two classes are linearly separable.

APPENDIX B
PROOF OF LEMMA 1

We will use the shorthand $\Delta\boldsymbol{x}_s = (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)$ where $\mathcal{S}[s] = (i, j)$.

(i) Recall the Matrix Bernstein Inequality for a $d \times d$ symmetric, random matrix $\boldsymbol{Z}$ and threshold γ:

$$P[\|\boldsymbol{Z}\|_2 \geq \gamma] \leq 2d \exp\left(\frac{-\gamma^2/2}{\mathbb{V}(\boldsymbol{Z}) + L\gamma/3}\right) \tag{32}$$

where $\|\boldsymbol{E}_s\| \leq L$.

First applying a spectral norm bound to the quadratic term gives $\sup_{\boldsymbol{w}} |\Delta_{\boldsymbol{\Sigma}}| \leq R_w \|\boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N\|_2$. We now take $\boldsymbol{Z} = \boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N$. We bound the spectral norm using the argument in [30]. For the matrix we can write $\boldsymbol{Z} = \sum_{s=1}^{S} \frac{1}{S}[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N]$. We denote each summand by $\boldsymbol{E}_s = \frac{1}{S}[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N]$. It then follows from triangle inequality that

$$\|\boldsymbol{E}_s\|_2 \leq \frac{1}{S}\left[\left\|\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top\right\|_2 + \|\boldsymbol{\Sigma}_N\|_2\right] = \frac{8R_x}{S} \ . \tag{33}$$

As each summand is centered and iid, the variance of sum decomposes as $\mathbb{V}(\Delta_{\boldsymbol{\Sigma}}) = \left\|\sum_{s\in\mathcal{S}} \mathbb{E}[\boldsymbol{E}_s^2]\right\|$. We note that the presence of $\boldsymbol{\Sigma}_N$ in the summands does not hinder independence, as this is just a constant quantity. For a single summand the second moment can be bounded as

$$\begin{aligned}
\mathbb{E}[\boldsymbol{E}_s^2] &= \frac{1}{S^2}\boldsymbol{E}\left[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N\right]^2 \\
&= \frac{1}{S^2}\left[\boldsymbol{E}\left[\|\Delta\boldsymbol{x}_s\|_2^2\,\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top\right] - \boldsymbol{\Sigma}_N^2\right] \\
&\preceq \frac{1}{S^2}\left[2R_x\boldsymbol{\Sigma}_N - \boldsymbol{\Sigma}_N^2\right] \\
&\preceq \frac{4R_x\boldsymbol{\Sigma}_N}{S^2} \ ,
\end{aligned} \tag{34}$$

from which the variance inequality $\mathbb{V}(\Delta_{\boldsymbol{\Sigma}}) \leq \frac{4R_x\|\boldsymbol{\Sigma}_N\|_2}{S}$ follows. Substituting Eqs. (33) and (34) into Eq. (32) yields the result.

(ii) We recall the following inequality for iid and bounded random vectors with mean $\bar{\boldsymbol{x}}$ [42]:

$$P\left(\left\|\frac{1}{S}\sum_{s=1}^{S}\boldsymbol{x}_s - \bar{\boldsymbol{x}}\right\|_2 \geq \gamma\right) \leq \exp\left\{-\frac{1}{2}\left(\frac{\gamma\sqrt{S}}{L} - 1\right)^2\right\} \ . \tag{35}$$

From the following inequalities

$$\begin{aligned}
\sup_{\boldsymbol{w}_1,\boldsymbol{w}_2} \|\Delta_\sigma\| &\leq \sup_{\boldsymbol{w}_1,\boldsymbol{w}_2} \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|_2 \,\|\boldsymbol{\mu}_N - \boldsymbol{\mu}_S\|_2 \\
&\leq 2\sqrt{R_w}\left\|\frac{1}{S}\sum_{s=1}^{S}\Delta\boldsymbol{x}_s - \boldsymbol{\mu}_N\right\|_2
\end{aligned} \tag{36}$$

the result is obtained by setting the threshold to $\epsilon = \gamma/(2\sqrt{R_w})$ and $L = \sqrt{R_x}$ in Eq. (35). □

REFERENCES

[1] H. Vincent Poor, *An Introduction to Signal Detection and Estimation*, Springer, 1998.

[2] Daniel J. Lingenfelter, Jeffrey A. Fessler, Clayton D. Scott, and Zhong He, "Asymptotic source detection performance of gamma-ray imaging systems under model mismatch," *IEEE Transactions on Signal Processing*, 2011.

[3] Jianshu Chen, Zaid J. Towfic, and Ali H. Sayed, "Dictionary learning over distributed models," *IEEE Transactions on Signal Processing*, 2015.

[4] Satish G. Iyengar, Pramod K. Varshney, and Thyagaraju Damarla, "A parametric copula-based framework for hypothesis testing using heterogeneous data," *IEEE Transactions on Signal Processing*, 2011.

[5] Hao Chen, Pramod K. Varshney, Steven M. Kay, and James H. Michels, "Theory of the stochastic resonance effect in signal detection: Part I - Fixed detectors," *IEEE Transactions on Signal Processing*, 2007.

[6] Hao Chen, Pramod K. Varshney, Steven M. Kay, and James H. Michels, "Theory of the stochastic resonance effect in signal detection: Part II - Variable detectors," *IEEE Transactions on Signal Processing*, 2008.

[7] Suat Bayram, San Gultekin, and Sinan Gezici, "Noise enhanced hypothesis testing according to restricted neyman pearson criterion," *Digital Signal Processing*, 2014.

[8] Xia Hong, Sheng Chen, and Chris J. Harris, "A kernel-based two-class classifier for imbalanced data sets," *IEEE Transactions on Neural Networks*, 2007.

[9] Cristiano L. Castro and Antonio P. Braga, "Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data," *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[10] Minlong Lin, Ke Tang, and Xin Yao, "Dynamic sampling approach to training neural networks for multiclass imbalance classification," in *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[11] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica, "Ad click prediction: a view from the trenches," in *Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[12] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *International Conference on Machine Learning*, 2015.

[13] Florian Schroff, Dmitry Kalenichenko, and James Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE conference on Computer Vision and Pattern Recognition*, 2015.

[14] Ulf Brefeld and Tobias Scheffer, "Auc maximizing support vector learning," in *ICML Workshop on ROC Analysis in Machine Learning*, 2005.

[15] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, 2003.

[16] Cesar Ferri, Peter Flach, and Jose Hernandez-Orallo, "Learning decision trees using the area under the roc curve," in *International Conference on Machine Learning (ICML)*, 2012.

[17] Tong Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *International Conference on Machine Learning (ICML)*, 2004.

[18] San Gultekin and John Paisley, "Asymptotic simulated annealing for variational inference," in *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[19] Peilin Zhao, Steven C. H. Hoi, Rong Jin, and Tianbao Yang, "Online AUC maximization," in *International Conference on Machine Learning (ICML)*, 2011.

[20] Wei Gao, Rong Jin, Shenghuo Zhu, and Zhi-Hua Zhou, "One-pass AUC optimization," in *International Conference on Machine Learning (ICML)*, 2013.

[21] Yi Ding, Peilin Zhao, Steven C. H. Hoi, and Yew Soon Ong, "An adaptive gradient method for online auc maximization," in *Association for Advancement of Artificial Intelligence (AAAI)*, 2015.

[22] Junjie Hu, Haikin Yang, Michael Lyu, Irwin King, and Anthony So, "Kernelized online imbalanced learning with fixed budgets," in *Association for Advancement of Artificial Intelligence (AAAI)*, 2015.

[23] Junjie Hu, Haikin Yang, Michael Lyu, Irwin King, and Anthony So, "Online nonlinear auc maximization for imbalanced data sets," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[24] Yi Ding, Chenghao Liu, Peilin Zhao, and Steven C.H. Hoi, "Large scale kernel methods for online auc maximization," in *International Conference on Data Mining (ICDM)*, 2017.

[25] Xinping Dong and Jianbing Shen, "Triplet loss in siamese network for object tracking," in *European Conference on Computer Vision (ECCV)*, 2018.

[26] Xinping Dong, Jianbing Shen, Dongming Wu, Kan Guo, Xiaogang Jin, and Fatih Porikli, "Quadruplet network with one-shot learning for fast visual object tracking," *IEEE Transactions on Image Processing*, 2019.

[27] Wei Gao and Zhi-Hua Zhou, "On the consistency of AUC pairwise optimization," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[28] San Gultekin and John Paisley, "Online forecasting matrix factorization," *IEEE Transactions on Signal Processing*, 2019.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016.

[30] Joel Tropp, "An introduction to matrix concentration inequalities," *Foundations and Trends in Machine Learning*, 2015.

[31] Stephan Clemencon, Gabor Lugosi, and Nicolas Vayatis, "Ranking and empirical minimization of u-statistics," *The Annals of Statistics*, 2008.

[32] Jianbing Shen, Xingping Dong, Jianteng Peng, Xiaogang Jin, Ling Shao, and Fatih Porikli, "Submodular function optimization for motion clustering and image segmentation," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[33] Jianbing Shen, Peng Jianteng, and Ling Shao, "Submodular trajectories for better motion segmentation in videos," *IEEE Transactions on Image Processing*, 2018.

[34] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, *Foundations of machine learning*, MIT Press, 2012.

[35] Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe, "Convexity, classification, and risk bounds," *Journal of the American Statistical Association*, 2006.

[36] Majdi Khalid, Indrakshi Ray, and Hamidreza Chitsaz, "Confidence-weighted bipartite ranking," in *Advanced Data Mining and Applications*, 2016.

[37] Hui Zou and Trevor Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology*, 2005.

[38] Aad W. Van der Vaart, *Asymptotic Statistics*, Cambridge University Press, 1998.

[39] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, May 2013.

[40] Nicolo Cesa-Bianchi and Gabor Lugosi, *Prediction, learning, and games*, Cambridge University Press, 2006.

[41] Mert Pilanci and Martin J. Wainwright, "Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares," *Journal of Machine Learning Research*, 2016.

[42] Ali Rahimi and Ben Recht, "Random features for large scale kernel machines," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[43] Ali Rahimi and Ben Recht, "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[44] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2014.

[45] Jiexiong Tang, Chenwei Deng, and Guang-Bin Huang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[46] Wenguan Wang, Jianbing Shen, and Haibin Ling, "A deep network solution for attention and aesthetics aware photo cropping," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.