Balanced Self-Paced Learning for AUC Maximization

Bin Gu[1,2*], **Chenkang Zhang**[2*], **Huan Xiong**[1,3*], **Heng Huang**[4]

[1]MBZUAI, United Arab Emirates
[2]School of Computer & Software, Nanjing University of Information Science & Technology, P.R.China
[3]Institute for Advanced Study in Mathematics, Harbin Institute of Technology, P.R.China
[4]Department of Electrical & Computer Engineering, University of Pittsburgh, PA, USA
bin.gu@mbzuai.ac.ae, 20201221058@nuist.edu.cn, huan.xiong.math@gmail.com, heng.huang@pitt.edu

Abstract

Learning to improve AUC performance is an important topic in machine learning. However, AUC maximization algorithms may decrease generalization performance due to the noisy data. Self-paced learning is an effective method for handling noisy data. However, existing self-paced learning methods are limited to pointwise learning, while AUC maximization is a pairwise learning problem. To solve this challenging problem, we innovatively propose a balanced self-paced AUC maximization algorithm (BSPAUC). Specifically, we first provide a statistical objective for self-paced AUC. Based on this, we propose our self-paced AUC maximization formulation, where a novel balanced self-paced regularization term is embedded to ensure that the selected positive and negative samples have proper proportions. Specially, the sub-problem with respect to all weight variables may be non-convex in our formulation, while the one is normally convex in existing self-paced problems. To address this, we propose a doubly cyclic block coordinate descent method. More importantly, we prove that the sub-problem with respect to all weight variables converges to a stationary point on the basis of closed-form solutions, and our BSPAUC converges to a stationary point of our fixed optimization objective under a mild assumption. Considering both the deep learning and kernel-based implementations, experimental results on several large-scale datasets demonstrate that our BSPAUC has a better generalization performance than existing state-of-the-art AUC maximization methods.

Introduction

Learning to improve AUC performance is an important topic in machine learning, especially for imbalanced datasets (Huang and Ling 2005; Ling, Huang, and Zhang 2003; Cortes and Mohri 2004). Specifically, for severely imbalanced binary classification datasets, a classifier may achieve a high prediction accuracy if it predicts all samples to be the dominant class. However, the classifier actually has a poor generalization performance because it cannot properly classify samples from non-dominant class. AUC (area under the ROC curve) (Hanley and McNeil 1982), which measures the probability of a randomly drawn positive sample having a higher decision value than a randomly drawn negative sample (McKnight and Najab 2010), would be a better evaluation criterion for imbalanced datasets.

Real-world data tend to be massive in quantity, but with quite a few unreliable noisy data that can lead to decreased generalization performance. Many studies have tried to address this, with some degree of success (Wu and Liu 2007; Zhai et al. 2020; Zhang et al. 2020). However, most of these studies only consider the impact of noisy data on accuracy, rather than on AUC. In many cases, AUC maximization algorithms may decrease generalization performance due to the noisy data. Thus, how to deal with noisy data in AUC maximization problems is still an open topic.

Since its birth (Kumar, Packer, and Koller 2010), self-paced learning (SPL) has attracted increasing attention (Wan et al. 2020; Klink et al. 2020; Ghasedi et al. 2019) because it can simulate the learning principle of humans, *i.e.*, starting with easy samples and then gradually introducing more complex samples into training. Complex samples are considered to own larger loss than easy samples, and noise samples normally have a relatively large loss. Thus, SPL could reduce the importance weight of noise samples because they are treated as complex samples. Under the learning paradigm of SPL, the model is constantly corrected and its robustness is improved. Thus, SPL is an effective method for handling noisy data. Many experimental and theoretical analyses have proved its robustness (Meng, Zhao, and Jiang 2017; Liu, Ma, and Meng 2018; Zhang et al. 2020). However, existing SPL methods are limited to pointwise learning, while AUC maximization is a pairwise learning problem.

To solve this challenging problem, we innovatively propose a balanced self-paced AUC maximization algorithm (BSPAUC). Specifically, we first provide an upper bound of expected AUC risk by the empirical AUC risk on training samples obeying pace distribution plus two more terms related to SPL. Inspired by this, we propose our balanced self-paced AUC maximization formulation. In particular, the sub-problem with respect to all weight variables may be non-convex in our formulation, while the one is normally convex in existing self-paced problems. To solve this challenging difficulty, we propose a doubly cyclic block coordinate descent method to optimize our formulation.

The main contributions of this paper are summarized as follows.

*These authors contributed equally.

1. Inspired by our statistical explanation for self-paced AUC, we propose a balanced self-paced AUC maximization formulation with a novel balanced self-paced regularization term. To the best of our knowledge, this is the first objective formulation introducing SPL into the AUC maximization problem.
2. We propose a doubly cyclic block coordinate descent method to optimize our formulation. Importantly, we give closed-form solutions of the two weight variable blocks and provide two instantiations of optimizing the model parameter block on kernel learning and deep learning, respectively.
3. We prove that the sub-problem with respect to all weight variables converges to a stationary point on the basis of closed-form solutions, and our BSPAUC converges to a stationary point of our fixed optimization objective under a mild assumption.

Self-Paced AUC

In this section, we first provide a statistical objective for self-paced AUC. Inspired by this, we provide our optimization objective.

Statistical Objective

Empirical and Expected AUC Objective for IID Data: Let X be a compact subset of $\mathbb{R}^d$, $Y = \{-1, +1\}$ be the label set and $Z = X \times Y$. Given a distribution $P(z)$ and let $S = \{z_i = (x_i, y_i)\}_{i=1}^n$ be an independent and identically distributed (IID) training set drawn from $P(z)$, where $x_i \in X$, $y_i \in Y$ and $z_i \in Z$. Thus, empirical AUC risk on S can be formulated as:

$$R_{emp}(S; f) = \frac{1}{n(n-1)} \sum_{z_i, z_j \in S, z_i \neq z_j} L_f(z_i, z_j). \quad (1)$$

Here, $f \in \mathcal{F} : \mathbb{R}^d \to \mathbb{R}$ is one real-valued function and the pairwise loss function $L_f(z_i, z_j)$ for AUC is defined as:

$$L_f(z_i, z_j) = \begin{cases} 0 & \text{if } y_i = y_j \\ \mathbb{I}(f(x_i) \leq f(x_j)) & \text{if } y_i = +1 \& \ y_j = -1 \\ \mathbb{I}(f(x_j) \leq f(x_i)) & \text{if } y_j = +1 \& \ y_i = -1 \end{cases}$$

where $\mathbb{I}(\cdot)$ is the indicator function such that $\mathbb{I}(\pi)$ equals 1 if π is true and 0 otherwise.

Further, the expected AUC risk for the distribution $P(z)$ can be defined as:

$$\begin{aligned} &R_{exp}(P(z); f) := \mathbb{E}_{z_1, z_2 \sim P(z)^2} L_f(z_1, z_2) \\ =&\mathbb{E}_{S \sim P(z)^n} \left[\frac{1}{n(n-1)} \sum_{z_i, z_j \in S, z_i \neq z_j} L_f(z_i, z_j) \right] \\ =&\mathbb{E}_S[R_{emp}(S; f)]. \end{aligned} \quad (2)$$

Compound Data: In practice, it is expensive to collect completely pure dataset because that would involve domain experts to evaluate the quality of collected data. Thus, it is a reasonable assumption that our training set in reality is composed of not only clean target data but also a proportion of noise samples (Natarajan et al. 2013; Kang et al. 2019). If we denote the distribution of clean target data by $P_{target}(z)$, and one of noisy data by $P_{noise}(z)$, the distribution for the real training data can be formulated as $P_{train}(z) = \alpha P_{target}(z) + (1 - \alpha) P_{noise}(z)$, where $\alpha \in [0, 1]$ is a weight to balance $P_{target}(z)$ and $P_{noise}(z)$. We also illustrate this compound training data in Figure 1. Note that we assume noise samples normally have a relatively large loss, and thus they are treated as complex samples in SPL as discussed previously.

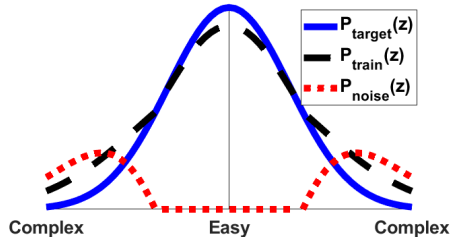


Figure 1: Data distribution on the degree of complexity.

Upper Bound of Expected AUC for Compound Data: Gong et al. (Gong et al. 2016) connect the distribution $P_{train}(z)$ of the training set with the distribution $P_{target}(z)$ of the target set using a weight function $W_\lambda(z)$:

$$P_{target}(z) = \frac{1}{\alpha_*} W_\lambda(z) P_{train}(z), \quad (3)$$

where $0 \leq W_\lambda(z) \leq 1$ and $\alpha_* = \int_Z W_\lambda(z) P_{train}(z) dz$ denotes the normalization factor. Intuitively, $W_\lambda(z)$ gives larger weights to easy samples than to complex samples and with the increase of pace parameter λ, all samples tend to be assigned larger weights.

Then, Eq. (3) can be reformulated as:

$$\begin{aligned} P_{train}(z) &= \alpha_* P_{target}(z) + (1 - \alpha_*) E(z), \quad (4) \\ E(z) &= \frac{1}{1 - \alpha_*} (1 - W_{\lambda_*}(z)) P_{train}(z). \end{aligned}$$

Here, $E(z)$ is related to $P_{noise}(z)$. Based on (4), we define the pace distribution $Q_\lambda(z)$ as:

$$Q_\lambda(z) = \alpha_\lambda P_{target}(z) + (1 - \alpha_\lambda) E(z), \quad (5)$$

where α_λ varies from 1 to α_* with increasing pace parameter λ. Correspondingly, $Q_\lambda(z)$ simulates the changing process from $P_{target}(z)$ to $P_{train}(z)$. Note that $Q_\lambda(z)$ can also be regularized into the following formulation:

$$Q_\lambda(z) \propto W_\lambda(z) P_{train}(z),$$

where $0 \leq W_\lambda(z) \leq 1$ through normalizing its maximal value to 1.

We derive the following result on the upper bound of the expected AUC risk. Please refer to Appendix for the proof.

Theorem 1 *For any $\delta > 0$ and any $f \in \mathcal{F}$, with confidence at least $1 - \delta$ over a training set S, we have:*

$$R_{exp}(P_{target}; f) \leq \frac{1}{n_\lambda(n_\lambda - 1)} \sum_{\substack{z_i, z_j \in S \\ z_i \neq z_j}} W_\lambda(z_i) W_\lambda(z_j) L_f(z_i, z_j)$$

$$+e_\lambda + \sqrt{\frac{\ln(1/\delta)}{n_\lambda/2}}, \tag{6}$$

where n_λ denotes the number of selected samples from the training set and e_λ decreases monotonically from 0 with the increasing of λ.

We will give a detailed explanation on the three terms of the upper bound (6) for Theorem 1 as follows.

1. The first term corresponds to the empirical AUC risk on training samples obeying pace distribution Q_λ. With increasing λ, the weights $W_\lambda(z)$ of complex samples gradually increase and these complex samples are gradually involved in training.
2. The second term reflects the expressive capability of training samples on the pace distribution Q_λ. With increasing λ, more samples are considered, the pace distribution Q_λ can be expressed better.
3. The last term measures the generalization capability of the learned model. As shown in Eq. (5), with increasing λ, α_λ gets smaller and the generalization of the learned model becomes worse. This is due to the gradually more evident deviation $E(z)$ from Q_λ to P_{target}.

Inspired by the upper bound (6) and the above explanations, we will propose our self-paced AUC maximization formulation in the next subsection.

Optimization Objective

First of all, we give the definition of some necessary notations. Let θ represent the model parameters, n and m denote the number of positive and negative samples respectively, $\mathbf{v} \in [0,1]^n$ and $\mathbf{u} \in [0,1]^m$ be the weights of positive and negative samples respectively, λ be the pace parameter for controlling the learning pace, and μ balance the proportions of selected positive and negative samples. The zero-one loss is replaced by the pairwise hinge loss which is a common surrogate loss in AUC maximization problems (Brefeld and Scheffer 2005; Zhao et al. 2011; Gao and Zhou 2015). Then, inspired by the upper bound (6), we have the following optimization objective:

$$\begin{aligned}
&\min_{\theta,\mathbf{v},\mathbf{u}} \ \mathcal{L}(\theta,\mathbf{v},\mathbf{u};\lambda) \\
&= \min_{\theta,\mathbf{v},\mathbf{u}} \underbrace{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} v_i u_j \xi_{ij}}_{\mathbf{1}} + \underbrace{\tau\Omega(\theta)}_{\mathbf{3}} \underbrace{-\lambda\left(\frac{1}{n}\sum_{i=1}^{n} v_i + \frac{1}{m}\sum_{j=1}^{m} u_j\right)}_{\mathbf{2}} \\
&\underbrace{+\mu\left(\frac{1}{n}\sum_{i=1}^{n} v_i - \frac{1}{m}\sum_{j=1}^{m} u_j\right)^2}_{\mathbf{4}} \\
&s.t.\ \mathbf{v} \in [0,1]^n, \mathbf{u} \in [0,1]^m
\end{aligned} \tag{7}$$

where $\xi_{ij} = \max\{1 - f(x_i^+) + f(x_j^-), 0\}$ is the pairwise hinge loss and $\Omega(\theta)$ is the regularization term to avoid overfitting. Specifically, θ is a matrix composed of weights and biases of each layer and $\Omega(\theta)$ is formulated as $\frac{1}{2}||\theta||_F^2$ in the deep learning setting and in the kernel-based setting, $\Omega(\theta)$ is formulated as $\frac{1}{2}||\theta||_{\mathcal{H}}^2$, where $||\cdot||_{\mathcal{H}}$ denotes the norm in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$.

As the explanation about the upper bound (6) shows, the upper bound is composed of three aspects: empirical risk, sample expression ability and model generalization ability. Inspired by this, we construct our optimization objective (7) which also considers the above three aspects. Specifically, the term **1** in Eq. (7) corresponds to the (weighted) empirical risk . The term **2** in Eq. (7) corresponds to the sample expression ability. As we explained before, sample expression ability is related to the number of selected samples and pace parameter λ in term **2** is used to control the number of selected samples. The term **3** in Eq. (7) corresponds to the model generalization ability which is a common model regularization term used to avoid model overfitting.

In addition, the term **4** in Eq. (7) is our new proposed balanced self-paced regularization term, which is used to balance the proportions of selected positive and negative samples as Figure 2 shows. Specifically, if this term is not used, only the degree of sample complexity is considered and this would lead to severe imbalance between the proportions of selected positive and negative samples in practice. But the proportions of selected positive and negative samples could be ensured properly if this term is enforced.

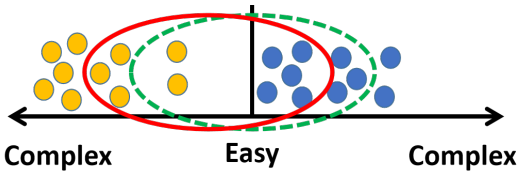


Figure 2: Contrast between whether or not using the balanced self-paced regularization term. (The yellow ball represents the positive sample, and the blue ball represents the negative sample. The green dotted ellipsoid represents the selected samples without using the term, and the red solid ellipsoid denotes the selected samples by using the term.)

BSPAUC Algorithm

In this section, we propose our BSPAUC algorithm (*i.e.*, Algorithm 1) to solve the problem (7). Different from traditional SPL algorithms (Wan et al. 2020; Klink et al. 2020; Ghasedi et al. 2019) which have two blocks of variables, our problem (7) has three blocks of variables which makes the optimization process more challenging. To address this issue, we propose a doubly cyclic block coordinate descent algorithm as shown in Algorithm 1, which consists of two layers of cyclic block coordinate descent algorithms. The outer layer cyclic block coordinate descent procedure (*i.e.*, lines 2-9 of Algorithm 1) follows the general optimization procedure of SPL to optimize all weight variables and model parameters alternatively. The inner layer cyclic block coordinate descent procedure (*i.e.*, lines 3-6 of Algorithm 1) is aimed to optimize the two blocks of weight variables (*i.e.*, $\mathbf{v}$ and $\mathbf{u}$) alternatively.

In the following, we revolve around the outer layer cyclic block coordinate descent procedure to discuss how to optimize all weight variables (*i.e.*, $\mathbf{v}$ and $\mathbf{u}$) and model parameters (*i.e.*, θ) respectively.

Algorithm 1: Balanced self-paced learning for AUC maximization

Input: The training set, θ^0, T, λ^0, c, λ_∞ and μ.

Initialize $\mathbf{v}^0 = \mathbf{1}_n$ and $\mathbf{u}^0 = \mathbf{1}_m$.
for $t = 1, \cdots, T$ **do**
  repeat
    Update $\mathbf{v}^t$ through Eq. (10).
    Update $\mathbf{u}^t$ through Eq. (11).
  until Converge to a stationary point.
  Update θ^t through solving (12).
  $\lambda^t = \min\{c\lambda^{t-1}, \lambda_\infty\}$.
end for

Output: The model solution θ.

Algorithm 2: Deep learning implementation of solving Eq. (12)

Input: $T, \eta, \tau, \hat{X}^+, \hat{X}^-, \pi, \theta^0$.

for $i = 1, \cdots, T$ **do**
  Sample $\hat{x}_1^+, ..., \hat{x}_\pi^+$ from $\hat{X}^+$.
  Sample $\hat{x}_1^-, ..., \hat{x}_\pi^-$ from $\hat{X}^-$.
  Calculate $f(x)$.
  Update θ by the following formula:

$$\theta = (1 - \eta_i\tau)\theta - \frac{\eta_i}{\pi}\sum_{j=1}^{\pi} v_j u_j \frac{\partial \xi_{jj}}{\partial \theta}.$$

end for

Output: θ.

Optimizing $\mathbf{v}$ and $\mathbf{u}$

Firstly, we consider the sub-problem with respect to all weight variables which is normally convex in existing self-paced problems. However, if we fix θ in Eq. (7), the sub-problem with respect to $\mathbf{v}$ and $\mathbf{u}$ could be non-convex as shown in Theorem 2. Please refer to Appendix for the proof of Theorem 2.

Theorem 2 *If we fix θ in Eq. (7), the sub-problem with respect to $\mathbf{v}$ and $\mathbf{u}$ maybe non-convex.*

In order to address the non-convexity of sub-problem, we further divide all weight variables into two disjoint blocks, *i.e.*, weight variables $\mathbf{v}$ of positive samples and weight variables $\mathbf{u}$ of negative samples. Note that the sub-problems *w.r.t.* $\mathbf{v}$ and $\mathbf{u}$ respectively are convex. Thus, we can solve the following two convex sub-problems to update $\mathbf{v}$ and $\mathbf{u}$ alternatively

$$\mathbf{v}^t = \arg\min_{\mathbf{v}\in[0,1]^n} \mathcal{L}(\mathbf{v}; \theta, \mathbf{u}, \lambda), \tag{8}$$

$$\mathbf{u}^t = \arg\min_{\mathbf{u}\in[0,1]^m} \mathcal{L}(\mathbf{u}; \theta, \mathbf{v}, \lambda). \tag{9}$$

We derive the closed-form solutions of the optimization problems (8) and (9) respectively in the following theorem. Note that sorted index represents the index of the sorted loss set $\{l_1, l_2, \dots\}$ which satisfies $l_i \le l_{i+1}$. Please refer to Appendix for the detailed proof.

Theorem 3 *The following formula gives one global optimal solution for problem* (8)*:*

$$\begin{cases} v_p = 1 & \text{if } l_p^+ < \lambda - 2\mu\left(\frac{p}{n} - \frac{\sum_{j=1}^m u_j}{m}\right) \\ v_p = n\left(\frac{\sum_{j=1}^m u_j}{m} - \frac{l_p^+ - \lambda}{2\mu} - \frac{p-1}{n}\right) & \text{otherwise} \\ v_p = 0 & \text{if } l_p^+ > \lambda - 2\mu\left(\frac{p-1}{n} - \frac{\sum_{j=1}^m u_j}{m}\right) \end{cases} \tag{10}$$

where $p \in \{1, ..., n\}$ is the sorted index based on the loss values $l_p^+ = \frac{1}{m}\sum_{j=1}^m u_j \xi_{pj}$.
The following formula gives one global optimal solution for problem (9)*:*

$$\begin{cases} u_q = 1 & \text{if } l_q^- < \lambda - 2\mu\left(\frac{q}{m} - \frac{\sum_{i=1}^n v_i}{n}\right) \\ u_q = m\left(\frac{\sum_{i=1}^n v_i}{n} - \frac{l_q^- - \lambda}{2\mu} - \frac{q-1}{m}\right) & \text{otherwise} \\ u_q = 0 & \text{if } l_q^- > \lambda - 2\mu\left(\frac{q-1}{m} - \frac{\sum_{i=1}^n v_i}{n}\right) \end{cases} \tag{11}$$

where $q \in \{1, ..., m\}$ is the sorted index based on the loss values $l_q^- = \frac{1}{n}\sum_{i=1}^n v_i \xi_{iq}$.

In this case, the solution (10) of problem (8) implies our advantages. Obviously, a sample with a loss greater/less than the threshold, *i.e.*, $\lambda - 2\mu(\frac{p-1}{n} - \frac{\sum_{j=1}^m u_j}{m})$ is ignored/involved in current training. In particular, the threshold is also a function of the sorted index, and consequently decreases as the sorted index increases. In this case, easy samples with less loss are given more preference. Besides, the proportion, *i.e.*, $\frac{\sum_{j=1}^m u_j}{m}$ of selected negative samples also affects the threshold. This means that the higher/lower the proportion of selected negative samples is, the more/fewer positive samples will be assigned high weights. Because of this, the proportions of selected positive and negative samples can be guaranteed to be balanced. What's more, we can easily find that the solution (11) of problem (9) yields similar conclusions. In summary, our algorithm can not only give preference to easy samples, but also ensure that the selected positive and negative samples have proper proportions.

Optimizing θ

In this step, we fix $\mathbf{v}, \mathbf{u}$ to update θ:

$$\theta^t = \arg\min_\theta \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m v_i u_j \xi_{ij} + \tau\Omega(\theta) + \text{const}, \tag{12}$$

where $\xi_{ij} = \max\{1 - f(x_i^+) + f(x_j^-), 0\}$. Obviously, this is a weighted AUC maximization problem. We provide two instantiations of optimizing the problem on kernel learning and deep learning settings, respectively.

For the deep learning implementation, we compute the gradient on random pairs of weighted samples which are selected from two subsets $\hat{X}^+$ and $\hat{X}^-$ respectively. $\hat{X}^+$ is a set of selected positive samples with weights $\hat{x}^+ = (v_i, x_i^+), \forall v_i > 0$ and $\hat{X}^-$ is a set of selected negative samples with weights $\hat{x}^- = (u_j, x_j^-), \forall u_j > 0$. In this case, we introduce the weighted batch AUC loss:

$$\sum_{j=1}^{\pi} v_j u_j \xi_{jj} = \sum_{j=1}^{\pi} v_j u_j \max\{1 - f(x_j^+) + f(x_j^-), 0\}, \tag{13}$$

and obtain Algorithm 2 by applying the doubly stochastic gradient descent method (DSGD) (Gu, Huo, and Huang 2019) *w.r.t.* random pairs of weighted samples, where η means the learning rate.

For the kernel-based implementation, we apply random Fourier feature method to approximate the kernel function (Rahimi and Recht 2008; Dai et al. 2014) for large-scale problems. The mapping function of D random features is defined as

$$\phi_\omega(x) = \sqrt{1/D}[\cos(\omega_1 x), \ldots, \cos(\omega_D x), \sin(\omega_1 x), \ldots, \sin(\omega_D x)]^T,$$

where ω_i is randomly sampled according to the density function $p(\omega)$ associated with $k(x, x')$ (Odland 2017). Then, based on the weighted batch AUC loss (13) and the random feature mapping function $\phi_\omega(x)$, we obtain Algorithm 3 by applying the triply stochastic gradient descent method (TSGD) (Dang et al. 2020) *w.r.t.* random pairs of weighted samples and random features which can be found in our appendix.

Theoretical Analysis

In this section, we first prove that the sub-problem with respect to all weight variables converges to a stationary point on the basis of closed-form solutions (*i.e.*, Theorem 3), and then prove that our BSPAUC converges to a stationary point of $\mathcal{L}(\theta, \mathbf{v}, \mathbf{u}; \lambda_\infty)$ under a mild assumption. All the proof details are available in the Appendix.

For the sake of clarity, we define $\mathcal{K}(\mathbf{v}, \mathbf{u}) = \mathcal{L}(\mathbf{v}, \mathbf{u}; \theta, \lambda)$ as the sub-problem of (7) where θ and λ are fixed, and then prove that $\mathcal{K}$ converges to a stationary point based on the closed-form solutions (*i.e.*, Theorem 3).

Theorem 4 *With the inner layer cyclic block coordinate descent procedure (*i.e.*, lines 3-6 of Algorithm 1), the sub-problem $\mathcal{K}$ with respect to all weight variables converges to a stationary point.*

Next, we prove that our BSPAUC converges along with the increase of hyper-parameter λ under a mild assumption.

Theorem 5 *If Algorithm 2 or Algorithm 3 (in appendix) optimizes θ such that $\mathcal{L}(\theta^{t+1}; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t) \leq \mathcal{L}(\theta^t; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t)$, BSPAUC converges along with the increase of hyper-parameter λ.*

Remark 1 *No matter the sub-problem (12) is convex or not, it is a basic requirement for a solver (e.g., Algorithm 2 and Algorithm 3 in appendix) such that the solution θ^{t+1} satisfies:*

$$\mathcal{L}(\theta^{t+1}; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t) \leq \mathcal{L}(\theta^t; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t).$$

Thus, we can have that our BSPAUC converges along with the increase of hyper-parameter λ.

Considering that the hyper-parameter λ reaches its maximum λ_∞, we will have that our BSPAUC converges to a stationary point of $\mathcal{L}(\theta, \mathbf{v}, \mathbf{u}; \lambda_\infty)$ if the iteration number T is large enough.

Theorem 6 *If Algorithm 2 or Algorithm 3 (in appendix) optimizes θ such that $\mathcal{L}(\theta^{t+1}; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t) \leq \mathcal{L}(\theta^t; \mathbf{v}^{t+1}, \mathbf{u}^{t+1}, \lambda^t)$, and λ reaches its maximum λ_∞, we will have that our BSPAUC converges to a stationary point of $\mathcal{L}(\theta, \mathbf{v}, \mathbf{u}; \lambda_\infty)$ if the iteration number T is large enough.*

Table 1: Datasets. (N_- means the number of negative samples and N_+ means the number of positive samples)

Dataset	Size	Dimensions	$\mathbf{N_-\backslash N_+}$
sector	9,619	55,197	95.18
rcv1	20,242	47,236	1.07
a9a	32,561	123	3.15
shuttle	43,500	9	328.54
aloi	108,000	128	999.00
skin_nonskin	245,057	3	3.81
cod-rna	331,152	8	2.00
poker	1,000,000	10	701.24

Experiments

In this section, we first describe the experimental setup, and then provide our experimental results and discussion.

Experimental Setup

Design of Experiments: To demonstrate the advantage of our BSPAUC for handling noisy data, we compare our BSPAUC with some state-of-the-art AUC maximization methods on a variety of benchmark datasets with/without artificial noisy data. The compared AUC maximization algorithms are based on linear, deep or kernel-based models, and some of them are batch algorithms, and the other ones are online algorithms. Specifically, the compared algorithms are summarized as follows.

TSAM: A kernel-based algorithm which updates the solution based on triply stochastic gradient descents *w.r.t.* random pairwise loss and random features (Dang et al. 2020).

DSAM: A modified deep learning algorithm which updates the solution based on the doubly stochastic gradient descents *w.r.t.* random pairwise loss (Gu, Huo, and Huang 2019).

KOIL$_{FIFO++}$ [1]: A kernelized online imbalanced learning algorithm which directly maximizes the AUC objective with fixed budgets for positive and negative class (Hu et al. 2015).

PPD$_{SG}$ [2]: A deep learning algorithm that builds on the saddle point reformulation and explores Polyak-Łojasiewicz condition (Liu et al. 2019).

OPAUC [3] : A linear method based on a regression algorithm, which only needs to maintain the first and second order statistics of data in memory (Gao et al. 2013).

In addition, considering hyper-parameters μ and λ_∞, we also design experiments to analyze their roles. Note that we introduce a new variable ν to illustrate the value of μ, *i.e.*, $\mu = \nu\lambda_\infty$, and a new indicator called absolute proportion difference (APD):

$$\text{APD} = \left| \frac{1}{n}\sum_{i=1}^{n} v_i - \frac{1}{m}\sum_{j=1}^{m} u_j \right|.$$

[1] KOIL$_{FIFO++}$ is available at https://github.com/JunjieHu.

[2] PPD$_{SG}$ is available at https://github.com/yzhuoning/Deep_AUC.

[3] OPAUC is available at http://lamda.nju.edu.cn/files/OPAUC.zip.

Table 2: Mean AUC results with the corresponding standard deviation on original benchmark datasets. ('–' means out of memory or unable to handle severely imbalanced datasets.)

Datasets	Non Deep Learning Methods				Deep Learning Methods		
	BSPAUC	**TSAM**	**KOIL**$_{FIFO++}$	**OPAUC**	**BSPAUC**	**DSAM**	**PPD**$_{SG}$
sector	**0.991±0.005**	0.986±0.005	0.953±0.014	0.971±0.018	**0.991±0.002**	0.978±0.007	0.935±0.008
rcv1	**0.979±0.001**	0.970±0.001	0.913±0.018	0.966±0.012	**0.993±0.001**	0.990±0.001	0.988±0.001
a9a	**0.926±0.008**	0.904±0.015	0.858±0.020	0.869±0.013	**0.927±0.005**	0.908±0.003	0.906±0.001
shuttle	**0.978±0.002**	0.970±0.004	0.948±0.010	0.684±0.036	**0.994±0.003**	0.989±0.004	—
aloi	**0.999±0.001**	**0.999±0.001**	**0.999±0.001**	0.998±0.001	**0.999±0.001**	**0.999±0.001**	—
skin_nonskin	**0.958±0.004**	0.946±0.004	—	0.943±0.007	**0.999±0.001**	**0.999±0.001**	0.949±0.001
cod-rna	**0.973±0.006**	0.966±0.010	—	0.924±0.024	**0.994±0.001**	0.992±0.001	0.988±0.001
poker	**0.934±0.013**	0.901±0.021	—	0.662±0.025	**0.990±0.004**	0.976±0.015	—

Table 3: Mean AUC results on noisy datasets. The corresponding standard deviations can be found in Appendix. (FP means the proportion of noise samples constructed by flipping labels, PP denotes the proportion of injected poison samples and '–' means out of memory.)

Datasets	rcv1			a9a			skin_nonskin			cod-rna		
FP	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
OPAUC	0.958	0.933	0.845	0.824	0.804	0.778	0.925	0.884	0.815	0.902	0.864	0.783
KOIL$_{FIFO++}$	0.901	0.889	0.804	0.836	0.806	0.726	—	—	—	—	—	—
TSAM	0.961	0.946	0.838	0.877	0.846	0.752	0.937	0.913	0.842	0.933	0.880	0.808
PDD$_{SG}$	0.964	0.936	0.855	0.881	0.849	0.739	0.940	0.912	0.852	0.937	0.873	0.788
DSAM	0.983	0.962	0.862	0.886	0.837	0.811	0.961	0.917	0.819	0.975	0.922	0.781
BSPAUC	**0.991**	**0.985**	**0.945**	**0.914**	**0.894**	**0.883**	**0.979**	**0.944**	**0.912**	**0.990**	**0.956**	**0.874**
PP	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
OPAUC	0.923	0.863	0.803	0.833	0.816	0.797	0.927	0.880	0.856	0.914	0.893	0.858
KOIL$_{FIFO++}$	0.891	0.838	0.793	0.831	0.823	0.806	—	—	—	—	—	—
TSAM	0.930	0.859	0.809	0.872	0.849	0.838	0.933	0.911	0.877	0.953	0.903	0.886
PDD$_{SG}$	0.934	0.918	0.828	0.885	0.873	0.839	0.935	0.927	0.898	0.972	0.941	0.881
DSAM	0.902	0.845	0.757	0.881	0.852	0.843	0.980	0.954	0.902	0.973	0.938	0.913
BSPAUC	**0.955**	**0.937**	**0.876**	**0.911**	**0.907**	**0.896**	**0.995**	**0.982**	**0.965**	**0.991**	**0.973**	**0.953**

Datasets: The benchmark datasets are obtained from the LIBSVM repository[4] which take into account different dimensions and imbalance ratios, as summarized in Table 1. The features have been scaled to $[-1, 1]$ for all datasets, and the multiclass classification datasets have been transformed to class imbalanced binary classification datasets. Specifically, we denote one class as the positive class, and the remaining classes as the negative class. We randomly partition each dataset into 75% for training and 25% for testing.

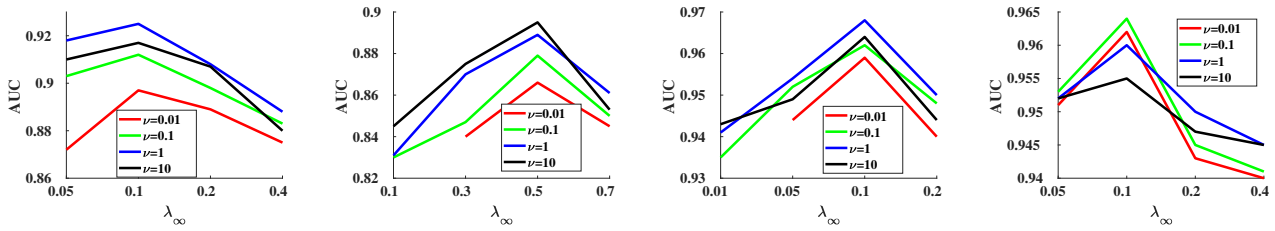
In order to test the robustness of all methods, we construct two types of artificial noisy datasets. The first method is to turn normal samples into noise samples by flipping their labels (Frénay and Verleysen 2013; Ghosh, Kumar, and Sastry 2017). Specifically, we first utilize training set to obtain a discriminant hyperplane, and then stochastically select samples far away from the discriminant hyperplane to flip their labels. Since these correctly classified samples far away from the discriminant hyperplane can obviously be regarded as clean samples, flipping their labels can affect the performance of model. Thus, they can be treated as noisy data. Another method is to inject poison samples (Jiang et al. 2019; Kwon, Yoon, and Park 2019; Zhang, Zhu, and Lessard 2020). Specifically, we generate poison samples for each dataset according to the poisoning attack method [5] (Biggio, Nelson, and Laskov 2012), and inject these poison samples into training set to form a noisy dataset. We conduct experiments with different noise proportions (from 10% to 30%).

Implementation: All the experiments are conducted on a PC with 48 2.2GHz cores, 80GB RAM and 4 Nvidia 1080ti GPUs and all the results are the average of 10 trials. We complete the deep learning and kernel-based implementations of our BSPAUC in python, we also implement the TSAM and the modified DSAM in python. We use the open codes as the implementations of KOIL$_{FIFO++}$, PPD$_{SG}$ and OPAUC, which are provided by their authors.

For the OPAUC algorithm on high dimensional datasets (the feature size is larger than 1000), we use the low-rank version, and set the rank parameter at 50. For all kernel-based methods, we use Gaussian kernel $k(x, x') = \exp(-\frac{||x-x'||^2}{2\sigma^2})$ and tune its hyper-parameter $\sigma \in 2^{[-5,5]}$ by a 5-fold cross-validation. For TSAM method and our Algorithm 3 (in appendix), the number of random Fourier features is selected from $[500 : 500 : 4000]$. For KOIL$_{FIFO++}$ method, the buffer size is set at 100 for each class. For all deep learning methods, we utilize the same network structure which consists of eight full connection layers and uses the ReLu activation function. For PPD$_{SG}$ method, the initial stage is tuned from 200 to 2000. For our BSPAUC, the hyper-parameters are chosen according to the proportion of selected samples. Specifically, we start training with about

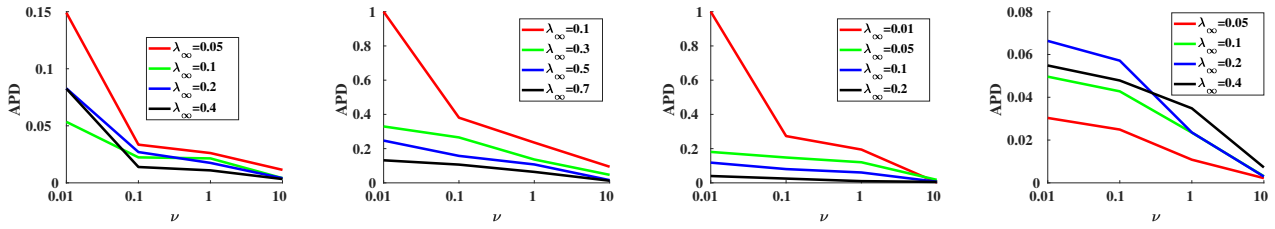
[4]Datasets are available at https://www.csie.ntu.edu.tw/˜cjlin/libsvmtools/datasets.

[5]The poisoning attack method is available at https://github.com/Trusted-AI/adversarial-robustness-toolbox.



(a) rcv1 (b) a9a (c) skin_nonskin (d) cod-rna

Figure 3: AUC results with different values of ν and λ_∞ on datasets with 20% injected poison samples. (Missing results are due to only positive or negative samples are selected.)



(a) rcv1 (b) a9a (c) skin_nonskin (d) cod-rna

Figure 4: The results of absolute proportion difference (APD) with different values of ν and λ_∞ on datasets with 20% injected poison samples.

50% samples, and then linearly increase λ to include more samples. Note that all algorithms have the same pre-training that we select a small number of samples for training and take the trained model as the initial state of experiments.

Results and Discussion

First of all, we explain the missing part of the experimental results on Tables 2 and 3. The KOIL$_{FIFO++}$ is a kernel-based method and needs to calculate and save the kernel matrix. For large datasets, such operation could cause out of memory. The PPD$_{SG}$ does not consider the severe imbalance of datasets and it can only produce negligible updates when the stochastic batch samples are severely imbalanced. If facing these issues, these two algorithms could not work as usual. Thus, we cannot provide the corresponding results.

Table 2 presents the mean AUC results with the corresponding standard deviation of all algorithms on the original benchmark datasets. The results show that, due to the SPL used in our BSPAUC, the deep learning and kernel-based implementations of our BSPAUC outperforms the DSAM and TSAM, which are the non self-paced versions of our implementations. At the same time, our BSPAUC also obtains better AUC results compared to other existing state-of-the-art AUC maximization methods (*i.e.*, OPAUC and PPD$_{SG}$).

Table 3 shows the performance of the deep learning implementation of our BSPAUC and other compared methods on the two types of noisy datasets. The results clearly show that our BSPAUC achieves the best performance on all noisy datasets. Specifically, the larger the proportion of noisy data is, the more obvious the advantage is. Our BSPAUC excludes the noise samples from training by giving a zero weight and thus has a better robustness.

Figures 3 and 4 show the results of the deep learning implementation of our BSPAUC with different hyper-parameter values on datasets with 20% injected poison samples. Figure 3 clearly reveals that with the increase of λ_∞, AUC increases first and then decreases gradually. This phenomenon is expected. When λ_∞ is small, increasing λ_∞ will cause more easy samples to join the training and thus the generalization of model is improved. However, when λ_∞ is large enough, complex (noise) samples start being included and then AUC decreases. What's more, Figure 4 directly reflects that with the increase of μ, which can be calculated by $\mu = \nu\lambda_\infty$, the proportions of selected positive and negative samples gradually approach. Further more, combining with the above two figures, we observe that too large APD often tends to low AUC. Large APD often implies that some easy samples in the class with low proportion of selected samples don't join the training while some complex samples in the other class with high proportion are selected. The above case leads to the reduction of generalization ability and thus causes low AUC. Importantly, our balanced self-paced regularization term is proposed for this issue.

Conclusion

In this paper, we first provide a statistical explanation to self-paced AUC. Inspired by this, we propose our self-paced AUC maximization formulation with a novel balanced self-paced regularization term. Then we propose a doubly cyclic block coordinate descent algorithm (*i.e.*, BSPAUC) to optimize our objective function. Importantly, we prove that the sub-problem with respect to all weight variables converges to a stationary point on the basis of closed-form solutions, and our BSPAUC converges to a stationary point of our fixed optimization objective under a mild assumption. The experimental results demonstrate that our BSPAUC outperforms existing state-of-the-art AUC maximization methods and has better robustness.

Acknowledgments

Bin Gu was partially supported by the National Natural Science Foundation of China (No:61573191).

References

Biggio, B.; Nelson, B.; and Laskov, P. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*.

Brefeld, U.; and Scheffer, T. 2005. AUC maximizing support vector learning. In *Proceedings of the ICML 2005 workshop on ROC Analysis in Machine Learning*.

Cortes, C.; and Mohri, M. 2004. AUC optimization vs. error rate minimization. In *Advances in neural information processing systems*, 313–320.

Dai, B.; Xie, B.; He, N.; Liang, Y.; Raj, A.; Balcan, M.-F.; and Song, L. 2014. Scalable kernel methods via doubly stochastic gradients. *arXiv preprint arXiv:1407.5599*.

Dang, Z.; Li, X.; Gu, B.; Deng, C.; and Huang, H. 2020. Large-Scale Nonlinear AUC Maximization via Triply Stochastic Gradients. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Frénay, B.; and Verleysen, M. 2013. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5): 845–869.

Gao, W.; Jin, R.; Zhu, S.; and Zhou, Z.-H. 2013. One-pass AUC optimization. In *International conference on machine learning*, 906–914.

Gao, W.; and Zhou, Z.-H. 2015. On the Consistency of AUC Pairwise Optimization. In *IJCAI*, 939–945.

Ghasedi, K.; Wang, X.; Deng, C.; and Huang, H. 2019. Balanced self-paced learning for generative adversarial clustering network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4391–4400.

Ghosh, A.; Kumar, H.; and Sastry, P. 2017. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Gong, T.; Zhao, Q.; Meng, D.; and Xu, Z. 2016. Why curriculum learning & self-paced learning work in big/noisy data: A theoretical perspective. *Big Data & Information Analytics*, 1(1): 111.

Gu, B.; Huo, Z.; and Huang, H. 2019. Scalable and Efficient Pairwise Learning to Achieve Statistical Accuracy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3697–3704.

Hanley, J. A.; and McNeil, B. J. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1): 29–36.

Hu, J.; Yang, H.; King, I.; Lyu, M. R.; and So, A. M.-C. 2015. Kernelized Online Imbalanced Learning with Fixed Budgets. In *AAAI*, 2666–2672.

Huang, J.; and Ling, C. X. 2005. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3): 299–310.

Jiang, W.; Li, H.; Liu, S.; Ren, Y.; and He, M. 2019. A flexible poisoning attack against machine learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 1–6. IEEE.

Kang, Z.; Pan, H.; Hoi, S. C.; and Xu, Z. 2019. Robust graph learning from noisy data. *IEEE transactions on cybernetics*, 50(5): 1833–1843.

Klink, P.; Abdulsamad, H.; Belousov, B.; and Peters, J. 2020. Self-paced contextual reinforcement learning. In *Conference on Robot Learning*, 513–529. PMLR.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *Advances in neural information processing systems*, 1189–1197.

Kwon, H.; Yoon, H.; and Park, K.-W. 2019. Selective poisoning attack on deep neural networks. *Symmetry*, 11(7): 892.

Ling, C. X.; Huang, J.; and Zhang, H. 2003. AUC: a better measure than accuracy in comparing learning algorithms. In *Conference of the canadian society for computational studies of intelligence*, 329–341. Springer.

Liu, M.; Yuan, Z.; Ying, Y.; and Yang, T. 2019. Stochastic auc maximization with deep neural networks. *arXiv preprint arXiv:1908.10831*.

Liu, S.; Ma, Z.; and Meng, D. 2018. Understanding self-paced learning under concave conjugacy theory. *arXiv preprint arXiv:1805.08096*.

McKnight, P. E.; and Najab, J. 2010. Mann-Whitney U Test. *The Corsini encyclopedia of psychology*, 1–1.

Meng, D.; Zhao, Q.; and Jiang, L. 2017. A theoretical understanding of self-paced learning. *Information Sciences*, 414: 319–328.

Natarajan, N.; Dhillon, I. S.; Ravikumar, P. K.; and Tewari, A. 2013. Learning with noisy labels. *Advances in neural information processing systems*, 26: 1196–1204.

Odland, T. 2017. *Fourier analysis on abelian groups; theory and applications*. Master's thesis, The University of Bergen.

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

Wan, Y.; Yang, B.; Wong, D. F.; Zhou, Y.; Chao, L. S.; Zhang, H.; and Chen, B. 2020. Self-Paced Learning for Neural Machine Translation. *arXiv preprint arXiv:2010.04505*.

Wu, Y.; and Liu, Y. 2007. Robust truncated hinge loss support vector machines. *Journal of the American Statistical Association*, 102(479): 974–983.

Zhai, Z.; Gu, B.; Li, X.; and Huang, H. 2020. Safe sample screening for robust support vector machine. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 6981–6988.

Zhang, X.; Wu, X.; Chen, F.; Zhao, L.; and Lu, C.-T. 2020. Self-Paced Robust Learning for Leveraging Clean Labels in Noisy Data. In *AAAI*, 6853–6860.

Zhang, X.; Zhu, X.; and Lessard, L. 2020. Online data poisoning attacks. In *Learning for Dynamics and Control*, 201–210. PMLR.

Zhao, P.; Hoi, S. C.; Jin, R.; and YANG, T. 2011. Online AUC maximization.