Does It Pay to Optimize AUC?

Baojian Zhou[1], **Steven Skiena**[2]

[1]Fudan University, Shanghai, China
[2]Stony Brook University, New York, USA
bjzhou@fudan.edu.cn, skiena@cs.stonybrook.edu

Abstract

The Area Under the ROC Curve (AUC) is an important model metric for evaluating binary classifiers, and many algorithms have been proposed to optimize AUC approximately. It raises the question of whether the generally insignificant gains observed by previous studies are due to inherent limitations of the metric or the inadequate quality of optimization.

To better understand the value of optimizing for AUC, we present an efficient algorithm, namely AUC-opt, to find the provably optimal AUC linear classifier in $\mathbb{R}^2$, which runs in $\mathcal{O}(n_+n_- \log(n_+n_-))$ where n_+ and n_- are the number of positive and negative samples respectively. Furthermore, it can be naturally extended to $\mathbb{R}^d$ in $\mathcal{O}((n_+n_-)^{d-1} \log(n_+n_-))$ by calling AUC-opt in lower-dimensional spaces recursively. We prove the problem is NP-complete when d is not fixed, reducing from the *open hemisphere problem*.

Experiments show that compared with other methods, AUC-opt achieves statistically significant improvements on between 17 to 40 in $\mathbb{R}^2$ and between 4 to 42 in $\mathbb{R}^3$ of 50 t-SNE training datasets. However, generally the gain proves insignificant on most testing datasets compared to the best standard classifiers. Similar observations are found for nonlinear AUC methods under real-world datasets.

1 Introduction

The Area Under the ROC Curve (AUC) (Hanley and McNeil 1982) is an important model evaluation metric that can be applied to a wide range of learning tasks such as binary classification (Bradley 1997), bipartite ranking (Freund et al. 2003), and recently fairness learning (Kallus and Zhou 2019; Vogel et al. 2021). It is a generally more reliable quality measure than the accuracy when the dataset is highly imbalanced, which often is the case in real-world problems. Multiple studies (Cortes and Mohri 2004; Joachims 2005) argue that optimizing classifiers for AUC may result in better classifiers than minimizing error rates.

A wide variety of algorithms (Provost and Fawcett 2001; Ferri, Flach, and Hernández-Orallo 2002; Yan et al. 2003; Cortes and Mohri 2004; Rakotomamonjy 2004; Herschtal and Raskutti 2004; Brefeld and Scheffer 2005; Joachims 2005; Calders and Jaroszewicz 2007; Qin, Liu, and Li 2010; Le et al. 2010; Zhao et al. 2011; Gao et al. 2013; Ying, Wen, and Lyu 2016; Eban et al. 2017; Liu et al. 2020) have been proposed to optimize AUC approximately under different learning settings. Typically, these methods relax the original *nonconvex nondifferentiable* objective to either convex or differentiable. Despite these advances, there exists no strong evidence that these algorithms generally perform better than standard classifiers. Empirical observations (Rakotomamonjy 2004; Joachims 2005) from previous indicate generally minor and statistically insignificant gains on particular datasets. This vagueness makes the question whether the observed results are due to the inherent limitations of the metric or the inadequate quality of optimization.

To better understand the virtues of optimizing for AUC, we investigate it from both *computational* and *algorithmic* viewpoints. Although AUC optimization is often reported to be NP-hard for linear hypothesis classes (Gao et al. 2013; Gao and Zhou 2015; Gultekin et al. 2020), we show that it is polynomial-time solvable if the data dimension d is fixed. We also prove that it is NP-complete if d is not fixed in advance. The key idea of our proof is a reduction from the *open hemisphere problem* (Johnson and Preparata 1978).

With the hope of polynomial-time solvable of linear classifiers, we present an efficient algorithm, namely AUC-opt, that can provably optimize AUC in $\mathbb{R}^2$. A key observation is that given any n training samples on the plane, the number of "interesting" classifiers is at most $\mathcal{O}(n_+n_-)$ where n_+ and n_- are the number of positive and negative samples respectively. Inspired by the idea of the topological sweep (Edelsbrunner and Guibas 1989), we calculate the AUC for the "minimal" slope once and then iterate through all other slopes by an ascending order using only constant update-time per iteration, yielding an $\mathcal{O}(n_+n_- \log n)$ algorithm. Furthermore, our method can be naturally extended to $\mathbb{R}^d$ in $\mathcal{O}((n_+n_-)^{d-1} \log(n))$ by calling AUC-opt in low-dimensional spaces recursively. This algorithm as an exponential depends on d and hence will be impractical on large real-world datasets where samples are usually high-dimensional. **Our goal here is not a general-purpose algorithm but to address whether the observed limitations of previous AUC optimizers result from inadequate optimization of a convex function or are an inherent result of the AUC objective criteria.** Doing such experiments requires the exact optimization of AUC-opt, even if we are

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

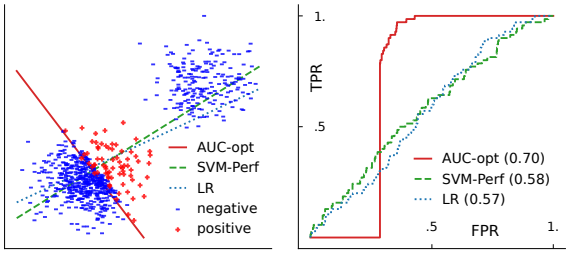

Figure 1: The popular AUC classifier SVM-Perf (Joachims 2005) fails to find a decent AUC separator on an adversarial example (left), performing similar to Logistic Regression (LR). The corresponding ROC curves and AUC scores (right) for these and our AUC-opt, which beats SVM-Perf and LR by a large margin.

limited to small data sets in low dimensions.

Fig. 1 presents a toy example as an illustration where there are significant improvements. To further validate AUC-opt, we conduct experiments on 50 real-world datasets projected onto both $\mathbb{R}^2$ and $\mathbb{R}^3$ by using t-SNE (Maaten and Hinton 2008). Experiments comparing AUC-opt against seven linear classifiers show that AUC-opt achieves statistically significant improvements on between 17 to 40 of 50 t-SNE datasets at the training stage. To summarize, our main contributions are:

- For the first time, we prove the linear AUC optimization problem is NP-complete when n and d are arbitrary but polynomial-time solvable for fixed d. A key to our proof is a reduction of the open hemisphere problem. Although NP-hard of the problem was often reported, we have not identified proof in the literature.
- We then present AUC-opt, to find the provably optimal AUC linear classifier in $\mathbb{R}^2$. It runs in $\mathcal{O}(n_+n_- \log(n_+n_-))$, which is optimal under the *algebraic computation tree model* (Ben-Or 1983). It can be naturally extended to $\mathbb{R}^d$ in $\mathcal{O}((n_+n_-)^{d-1} \log(n_+n_-))$ by decomposing original problem into same subproblems of lower dimensional spaces and calling AUC-opt recursively.
- Experiments comparing AUC-opt against seven other classification methods show that AUC-opt achieves significant improvements on between 17 to 40 in $\mathbb{R}^2$ and on between 4 to 42 in $\mathbb{R}^3$ of 50 t-SNE training datasets. But generally, the gain proves insignificant on most testing datasets compared to the best standard classifiers. Empirical results suggest that approximate AUC classifiers have space to improve.
- Similarly, empirical findings on nonlinear classifiers further suggest that the partial loss of significance of approximate AUC optimizers may be due to imperfect approximation, thus having space to improve current approximate algorithms.

Related Work

AUC optimization. In the seminal work, Cortes and Mohri (2004) shows that AUC is monotonically increasing with respect to the accuracy and is equal to the accuracy when $n_+ = n_-$. Yet, because the variance is not zero, optimizing directly for AUC may still yield better AUC values than that of standard classifiers. Many AUC optimization methods have been proposed over past years (Yan et al. 2003; Herschtal and Raskutti 2004; Brefeld and Scheffer 2005; Joachims 2005; Rakotomamonjy 2004; Calders and Jaroszewicz 2007; Ying, Wen, and Lyu 2016; Yang and Ying 2022). These approaches all focus on approximation due to the nonconvex and nondifferentiable of the AUC objective. To avoid this, Yan et al. (2003) propose to replace the 0-1 objective by a sum of differentiable sigmoid so that a gradient descent-based method can be applied. Joachims (2005) relaxes the problem to a convex one so that SVM can be used (see also (Rakotomamonjy 2004; Brefeld and Scheffer 2005)). A study of Rakotomamonjy (2004) indicates that optimizing SVM objective is also attend to optimize AUC (Rakotomamonjy 2004; Steck 2007). More recently, methods for optimizing AUC are studied under the online learning setting (Zhao et al. 2011; Gao et al. 2013; Ying, Wen, and Lyu 2016; Liu et al. 2018). However, performance gains are insignificant found in these studies, and there is a lack of comparison between AUC optimizers and standard methods. Different from these previous works, our goal is to optimize AUC score without approximation.

Bipartite ranking. The AUC optimization is closely related with the bipartite ranking problem (Freund et al. 2003; Le et al. 2010; Qin, Liu, and Li 2010; Rudin and Wang 2018) where minimizing the pairwise misranking error is equivalent to maximize the AUC score. For example, RankBoost (Freund et al. 2003), a popular ranking algorithm, implicitly optimizes AUC as proved in (Cortes and Mohri 2004). Kotłowski, Dembczyński, and Hüllermeier (2011) consider maximizing AUC as a minimization of the rank loss. Recently, Rudin and Wang (2018) propose to directly optimize rank statistics by using mixed-integer programming. More works can be found in Menon and Williamson (2016) and references therein.

Computational complexity results. Although NP-hard of the problem is often cited as folklore (Gao et al. 2013; Gao and Zhou 2015; Gultekin et al. 2020), we have not identified proof in the literature. Several NP-hardness results have previously been shown for both classifications of 0-1 loss and ranking (Feldman et al. 2012; Ben-David, Eiron, and Long 2003; Cohen, Schapire, and Singer 1998). Cohen, Schapire, and Singer (1998) show that finding the optimal ranking is NP-complete. Although Joachims (2005) shows that the AUC optimization can be reformulated as a classification problem, a proof of NP-hardness from it does not seem to follow naturally. Instead, we prove the NP-hardness by the reduction from the open hemisphere problem.

Paper Outline and Notations

The remainder of this paper is organized as follows. We first present preliminaries in §2. The proof of NP-hardness of AUC optimization is given in §3. AUC-opt and its generalization to high-dimensional space are given in §4. We empirically evaluate AUC-opt and then make our conclusion in §5 and §6, respectively. Throughout this paper, we re-

strict our attention to optimize AUC in a linear hypothesis class $\mathcal{H}$ i.e. $\boldsymbol{f} \in \mathcal{H} := \{\boldsymbol{w} : \boldsymbol{w} \in \mathbb{R}^d\}$. Given a set of n training examples $\mathcal{D} := \{(\boldsymbol{x}_i, y_i) : i \in \{1, 2, \ldots, n\}\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, we rewrite $\mathcal{D}$ as a union of $\mathcal{D}_+$ and $\mathcal{D}_-$ where $\mathcal{D}_+$ is the set of positive samples written as $\{(\boldsymbol{x}_1^+, y_1^+), \ldots, (\boldsymbol{x}_{n_+}^+, y_{n_+}^+)\}$ and $\mathcal{D}_-$ is the set of negative samples written as $\{(\boldsymbol{x}_1^-, y_1^-), \ldots, (\boldsymbol{x}_{n_-}^-, y_{n_-}^-)\}$, respectively. Clearly, $n = n_+ + n_-$ and $\mathcal{D} = \mathcal{D}_+ \cup \mathcal{D}_-$. x_{ij} is denoted as j-th entry of vector $\boldsymbol{x}_i$, i.e. $\boldsymbol{x}_i = [x_{i1}, x_{i1}, \ldots, x_{id}]^\top$.

2 Preliminaries

We first review the definition of AUC statistic and give the problem formulation under linear hypothesis $\mathcal{H}$. We discuss that the problem is efficiently solvable when $\mathcal{D}$ is separable.

Definition 1 (AUC Statistic (Clémençon et al. 2008))**.** *Let (X, Y) and (X', Y') be two pairs of random variables in $\mathbb{R}^d \times \{\pm 1\}$ following the same unknown distribution. Denote the probability of an event A condition on $\{Y = 1, Y' = -1\}$ as $\mathbb{P}\{A | Y = 1, Y' = -1\}$. Given a score function $\boldsymbol{f} : \mathbb{R}^d \mapsto \mathbb{R}$, the AUC statistic is*

$$\mathrm{AUC}(\boldsymbol{f}) := \mathbb{P}\{\boldsymbol{f}(X) > \boldsymbol{f}(X') | Y = 1, Y' = -1\}.$$

Statistically, $\mathrm{AUC}(\boldsymbol{f})$ is the probability that a randomly chosen positive sample is ranked by $\boldsymbol{f}$ higher than a randomly chosen negative one (tie-breaks lexicographically). It is equivalent to the Wilcoxon statistic (Hanley and McNeil 1982). Given $\mathcal{D}$, our linear AUC optimization is empirically defined as the following.

Problem 1 (Linear AUC Optimization (LAO))**.** *Given the dataset $\mathcal{D}$ and the hypothesis class $\mathcal{H} := \{\boldsymbol{w} : \boldsymbol{w} \in \mathbb{R}^d\}$, the LAO problem is to find a $\boldsymbol{w} \in \mathcal{H}$ such that the empirical AUC score is maximized, that is*

$$(LAO) \quad \boldsymbol{w}^\star \in \operatorname*{arg\,max}_{\boldsymbol{w} \in \mathcal{H}} \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \frac{\mathbf{1}[\boldsymbol{w}^\top \boldsymbol{x}_i^+ > \boldsymbol{w}^\top \boldsymbol{x}_j^-]}{n_+ n_-}, \quad (1)$$

where the indicator $\mathbf{1}[A] = 1$ if A is true 0 otherwise.

Due to the non-convexity of 0-1 loss in LAO, directly optimizing (1) is challenging. Notice that $\boldsymbol{w}^\star$ is not unique as $p + \alpha \boldsymbol{w}^\star$ with $\alpha > 0$ and $p \in \mathbb{R}$ is always an optimizer. Although (1) is hard to optimize, if $\mathcal{D}$ is *linearly separable*, that is, there exists a $\boldsymbol{w}$ such that for any $(\boldsymbol{x}_i^+, y_i^+) \in \mathcal{D}_+$, $\langle \boldsymbol{w}, \boldsymbol{x}_i^+ \rangle \geq 0$ and for any $(\boldsymbol{x}_j^-, y_j^-) \in \mathcal{D}_-$, $\langle \boldsymbol{w}, \boldsymbol{x}_j^- \rangle < 0$, then one can always find a $\boldsymbol{w}$ such that $\mathrm{AUC}(\boldsymbol{w}) = 1$ in polynomial-time (Elizondo 2006; Yogananda, Murty, and Gopal 2007) by using Perceptron (Freund and Schapire 1999) or linear programming techniques. For example, the worst time complexity of an iterative reduction algorithm is $\mathcal{O}(nr^3)$ where $r \leq \min(n, d + 1)$ (Elizondo 2006). In the rest of this paper, we assume $\mathcal{D}$ is not linearly separable.

Although previous studies have claimed the NP-hardness of LAO (Gao et al. 2013; Gao and Zhou 2015; Gultekin et al. 2020), no previous literature proves it even for the linear classifier case. It motivates us to prove the NP-hardness under the linear hypothesis.

3 NP-hardness of LAO

This section proves the LAO problem under the linear hypothesis is NP-complete if n and d are not fixed but polynomial-time solvable when d is fixed. We first introduce the open hemisphere problem and then prove the NP-complete by a reduction from it.

Definition 2 (Open hemisphere)**.** *Given the unit sphere $\mathcal{S}^{d-1} := \{\boldsymbol{s} \in \mathbb{R}^d : \|\boldsymbol{s}\|_2 = 1\}$, the open hemisphere of $\boldsymbol{w}$ is defined as a set $\{\boldsymbol{s} \in \mathcal{S}^{d-1} : \langle \boldsymbol{w}, \boldsymbol{s} \rangle > 0\}$.*

Problem 2 (Open hemisphere problem (Johnson and Preparata 1978))**.** *Let $\mathcal{K} := \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_t\}$ be a subset of $\mathbb{Q}^d \cap \mathcal{S}^{d-1}$ where $\mathbb{Q}$ is the set of rationals. The open hemisphere problem is to find an open hemisphere such that it contains a largest subset of $\mathcal{K}$, that is,*

$$\operatorname*{arg\,max}_{\boldsymbol{w} \in \mathbb{R}^d} |\{\boldsymbol{s}_i \in \mathcal{K} : \langle \boldsymbol{w}, \boldsymbol{s}_i \rangle > 0\}|.$$

To ease our analysis, we formulate the open hemisphere problem as a feasibility problem. Given positive integers d, m and a set $\mathcal{K}$, does there exist a hyperplane $\boldsymbol{w}$ such that at least m inequalities are satisfied, that is,

$$(OH) \qquad |\{\boldsymbol{s}_i \in \mathcal{K} : \langle \boldsymbol{w}, \boldsymbol{s}_i \rangle > 0\}| \geq m \, ? \qquad (2)$$

Lemma 1 (NP-complete of OH (Johnson and Preparata 1978))**.** *Given positive d, n, m, and $\mathcal{K} \subseteq \mathbb{Q}^d \cap \mathcal{S}^{d-1}$, the feasibility of OH problem defined in (2) is NP-complete when both n and d are not fixed.*

The above lemma shows the fact that OH is NP-complete. Based on this lemma, we have the following main theorem.

Theorem 1 (NP-complete of LAO)**.** *Consider the linear AUC optimization problem defined in Problem 1, if $\mathcal{D}$ is not linear separable, LAO is NP-complete when both n and d are arbitrary.*

Before proving Thm. 1, we first reformulate the LAO problem as a feasibility problem and then show this feasibility problem is NP-complete.

Problem 3 (The feasibility problem of LAO)**.** *Given a finite dataset $\mathcal{D}$, a set of linear classifiers $\mathcal{H}$, and positive integers d, t as an input, the feasibility problem of LAO is to ask, does there exist $\boldsymbol{w} \in \mathcal{H}$ such that*

$$\sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \mathbf{1}[\boldsymbol{w}^\top \boldsymbol{x}_i^+ > \boldsymbol{w}^\top \boldsymbol{x}_j^-] \geq t \, ? \qquad (3)$$

Proof Sketch.[1] Without loss of generality, let us assume the problem in $\mathbb{Q}^d$. To prove the NP-complete, we only need to show that the feasibility of LAO defined in Problem 3 is both in NP and is NP-hard by a reduction from OH. First of all, Problem 3 is in NP. Given any $\boldsymbol{w} \in \mathcal{H}$, one can find a polynomial-time verifier such that it finishes in $\mathcal{O}(n_p n_q d)$ time, and each certificate has a polynomial length for the input.

To show Problem 3 is NP-hard, given any instance of the OH problem, the goal is to prove that an instance of (3) can solve it. To do this, we construct the training dataset $\mathcal{D}$ so

[1] A detailed proof is in the supplementary.

that an instance of Problem 3 can be defined. Notice that one can rewrite vectors in $\mathcal{K}$ and construct new vectors $\boldsymbol{x}_i^+$ and $\boldsymbol{x}_1^-$ as the following

$$\boldsymbol{s}_1 = \underbrace{(\boldsymbol{s}_1 + \boldsymbol{x}_1^-)}_{\boldsymbol{x}_1^+} - \boldsymbol{x}_1^-, \ldots, \boldsymbol{s}_t = \underbrace{(\boldsymbol{s}_t + \boldsymbol{x}_1^-)}_{\boldsymbol{x}_t^+} - \boldsymbol{x}_1^-. \quad (4)$$

The set of training labels is constructed such that $y_1^+, y_2^+, \ldots, y_t^+$ are all ones and $y_1^- = -1$. Combining it with equations in (4) provides a dataset $\mathcal{D} = \{(\boldsymbol{x}_1^+, y_1^+), \ldots, (\boldsymbol{x}_t^+, y_t^+), (\boldsymbol{x}_1^-, y_1^-)\}$. The two left figures of Fig. 2 illustrate this reduction where the top figure is a sphere, and each positive sign represents $\boldsymbol{s}_i$ while the negative sign is $\boldsymbol{x}_1^- = \boldsymbol{0}$. The sphere contains 14 points, which correspond to 14 inequalities of the left-hand side of (3). The normal $\boldsymbol{w}$ defines a hyperplane which corresponds to $t = 8$ of the right-hand side of (3). The bottom figure shows an AUC curve where TPR and FPR are true positive rates and false positive rates, respectively. It indicates that $\boldsymbol{w}$ exists, so the number of inequalities that can be satisfied is at least m. This transformation and checking procedure can be done in polynomial time. Therefore, any answer to the instance of LAO is affirmative if and only if the instance of OH is affirmative; hence, the problem is NP-hard. ■

4 Proposed Methods for LAO

In this section, we first present a trivial method in $\mathbb{R}^2$ and then propose AUC-opt inspiring from *topological sweeping*. We then extend AUC-opt to $\mathbb{R}^d$ by projecting high-dimensional problems into low-dimensional ones.

A Trivial Method

Notice that every pair of training samples defines a supporting line that separates the rest training samples, and the number of these interesting lines are at most $\mathcal{O}(n^2)$. Given any two sample $\boldsymbol{x}_i := [x_{i1}, x_{i2}]^\top$ and $\boldsymbol{x}_j := [x_{j1}, x_{j2}]^\top$, the slope of the line defined by $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is $m = -(x_{i2} - x_{j2})/(x_{i1} - x_{j1})$. Hence, one can find an algorithm runs in $\mathcal{O}(n^3 \log n)$ and takes $\mathcal{O}(n^2)$ space by using the following two steps: 1) identify all $n(n-1)$ slopes m; and 2) for each slope m, let $\boldsymbol{w} := [m, 1]^\top$ and then calculate $\mathrm{AUC}(\boldsymbol{w})$ by using an $\mathcal{O}(n \log n)$ algorithm (Fawcett 2006). $\boldsymbol{w}^\star$ is the one that gives the highest $\mathrm{AUC}(\boldsymbol{w})$ value.

AUC-opt

However, the above trivial method can be significantly improved. There are two successive algorithmic improvements needed. The first is that the number of interesting slopes is at most $\mathcal{O}(n_+ n_-)$ by noticing that training pairs with the same labels are not interesting. The second improvement is inspired by the *topological sweep* (Edelsbrunner and Guibas 1989) that we can save n running time by sorting all slopes once; that is, *we only need to calculate the AUC score once for the minimal slope and then sweep over all the rest of the slopes in ascending order.*

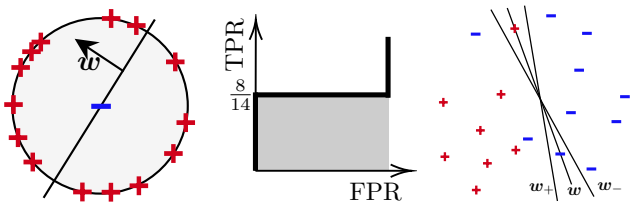

Figure 2: Sphere and its AUC score.

Algorithm 1: $[\mathrm{AUC}_{\mathrm{opt}}, \boldsymbol{w}]$ =AUC-opt($\mathcal{D}$)

1: $S = \{\}$ ▷ Initialize a slope set
2: **for** $(i, j) \in \{1, \ldots, n_+\} \times \{1, \ldots, n_-\}$ **do**
3: $\quad m = -(x_{i2}^+ - x_{j2}^-)/(x_{i1}^+ - x_{j1}^-)$ ▷ Calculate a slope
4: $\quad S = S \cup \{(m, i, j)\}$
5: **end for**
6: Let $\epsilon < \min_{(i,j) \neq (i',j')} \left| \frac{x_{i2} - x_{j2}}{x_{i1} - x_{j1}} - \frac{x_{i'2} - x_{j'2}}{x_{i'1} - x_{j'1}} \right|$ be a positive constant smaller than the minimal difference between any two slopes.
7: Let $\min(S) = \min\{m | (m, i, j) \in S\}$
8: $\boldsymbol{w} = [\min(S) - 1, 1]^\top$
9: $\mathrm{AUC}_{\mathrm{cur}} = \mathrm{AUC}(\boldsymbol{w})$ ▷ Algo. 1 in Fawcett (2006)
10: $\mathrm{AUC}_{\mathrm{opt}} = \mathrm{AUC}_{\mathrm{cur}}$
11: **for** $(m, i, j) \in \mathrm{sort}(S)$ **do**
12: $\quad c_1 = 0, c_2 = 0, \boldsymbol{u} = [m - \epsilon, 1]^\top, \boldsymbol{v} = [m + \epsilon, 1]^\top$
13: $\quad$ **if** $\boldsymbol{u}^\top \boldsymbol{x}_i^+ > \boldsymbol{u}^\top \boldsymbol{x}_j^-$ **then** $c_1 = 1$
14: $\quad$ **if** $\boldsymbol{v}^\top \boldsymbol{x}_i^+ > \boldsymbol{v}^\top \boldsymbol{x}_j^-$ **then** $c_2 = 1$
15: $\quad \mathrm{AUC}_{\mathrm{cur}} = \mathrm{AUC}_{\mathrm{cur}} + (c_2 - c_1)/(n_+ n_-)$
16: $\quad$ **if** $\mathrm{AUC}_{\mathrm{opt}} < \mathrm{AUC}_{\mathrm{cur}}$ **then**
17: $\quad\quad \mathrm{AUC}_{\mathrm{opt}} = \mathrm{AUC}_{\mathrm{cur}}, \boldsymbol{w} = [m + \epsilon, 1]^\top$
18: $\quad$ **end if**
19: $\quad$ **if** $\mathrm{AUC}_{\mathrm{opt}} < 1 - \mathrm{AUC}_{\mathrm{cur}}$ **then**
20: $\quad\quad \mathrm{AUC}_{\mathrm{opt}} = 1 - \mathrm{AUC}_{\mathrm{cur}}, \boldsymbol{w} = [-m - \epsilon, -1]^\top$
21: $\quad$ **end if**
22: **end for**

The description of AUC-opt is presented in Algo. 1. There are three main steps: 1) to obtain all possible slopes and store them into S (L1 to L5); 2) to calculate the AUC score of the minimal slope (L7 to L9); and 3) to update $\boldsymbol{w}$ and its AUC score (L10 to L21). The critical part is the update rules of sweeping $\boldsymbol{w}$. More specifically, at each iteration (from L10 to L21), the change of AUC score is from $\{0, \pm 1/(n_+ n_-)\}$ by "sweeping" $\boldsymbol{w}$ (L10 - L13). We illustrate this procedure on the upper right of Fig. 2 where red plus signs are positives while blue bar signs are negatives. Let $\boldsymbol{w} = [s, 1]^\top, \boldsymbol{w}_- = [s - \epsilon, 1]^\top$, and $\boldsymbol{w}_+ = [s + \epsilon, 1]^\top$. The increase of slope from $\boldsymbol{w}_-$ to $\boldsymbol{w}_+$ only changes the ordering of two samples which corresponds to a magnitude change of $|1/(n_+ n_-)|$ for $\mathrm{AUC}(\boldsymbol{w})$. The following theorem shows $\boldsymbol{w}$ returned by Algo. 1 is optimal.

Theorem 2. *Given the dataset* $\mathcal{D} := \{(\boldsymbol{x}_i, y_i) : i \in \{1, 2, \ldots, n\}\}$ *where* $\boldsymbol{x}_i \in \mathbb{R}^2$ *and* $y_i \in \{\pm 1\}$ *and a collection of linear separators* $\mathcal{H} := \{\boldsymbol{w} : \boldsymbol{w} \in \mathbb{R}^2\}$. *The proposed* AUC-opt, *solves the LAO problem (1) in* $\mathcal{O}(n_+ n_- \log(n_+ n_-))$. *This time complexity is tight under the algebraic computation tree model.*

Proof. We first show that $\boldsymbol{w}$ returned by AUC-opt is optimal. The key of our proof is to show that Algo. 1 iterates all possible AUC scores given by noticing that all slopes of lines between two consecutive slopes give the same AUC score.

Let $\boldsymbol{w} = [w_1, w_2]^\top$ be any line in $\mathbb{R}^2$. We assume that $\boldsymbol{w}$ is a normal vector of two training samples $\boldsymbol{x}_i^+, \boldsymbol{x}_j^-$, that is, $\boldsymbol{w}$ is given by $\langle \boldsymbol{w}, \boldsymbol{x}_i^+ - \boldsymbol{x}_j^- \rangle = 0$. The slopes of these normal vectors on $\mathbb{R}^2$ can be calculated. Let the collection of all such slopes be $\mathcal{S} := \left\{-(x_{i2}^+ - x_{j2}^-)/(x_{i1}^+ - x_{j1}^-) : \boldsymbol{x}_i^+, \boldsymbol{x}_j^- \in \mathcal{D}\right\}$. Sort the collection of slopes $\mathcal{S}$ and $\mathbb{R}^2$ has been partitioned into $n_+n_- + 1$ parts.

We consider two consecutive slopes in the sorted $\mathcal{S}$ more carefully. Let us denote two consecutive sorted slopes as s_1 and s_2 which associated with two pairs $(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-)$ and $(\boldsymbol{x}_{i'}^+, \boldsymbol{x}_{j'}^-)$, respectively. We only need to show that $\forall s \in (s_1, s_2), [s, 1]^\top$ has identical AUC score. To do so, all we need to do is to show that given any $\boldsymbol{x}_i^+, \boldsymbol{x}_j^-$, $\boldsymbol{w}$ scores $\boldsymbol{x}_i^+, \boldsymbol{x}_j^-$ as a same ordering. In other words, given any $\boldsymbol{w} := \{[s, 1]^\top : s \in (s_1, s_2)\}$, the quantity $\boldsymbol{x}_i^\top \boldsymbol{w} - \boldsymbol{x}_j^\top \boldsymbol{w}$ always has same sign. In the rest proof, we show this by carefully constructing a quantity function $s(\lambda)$ as the following

$$s(\lambda) := -\frac{x_{i2}^+ - x_{j2}^-}{x_{i1}^+ - x_{j1}^-}\lambda - \frac{x_{i2'}^+ - x_{j2'}^-}{x_{i1'}^+ - x_{j1'}^-}(1-\lambda),$$

where $\lambda \in (0, 1)$.

We need to study the monotonicity of $s(\lambda)$, we define another function $h(\lambda) := s(\lambda)(x_{i1} - x_{j1}) + x_{i2} - x_{j2}$. Clearly, $h(\lambda)$ is non-decreasing function by noticing that $h'(\lambda) = s_2 - s_1 \geq 0$. We just need to show $h(\lambda)$ never vanishes at $\lambda \in (0, 1)$. Assume that we have $h(\lambda) = 0$, then we have $s(\lambda) = -(x_{i2} - x_{j2})/(x_{i1} - x_{j1})$. It makes a contradiction since there is no existing slope between s_1 and s_2. Similarly, one can show that for any two consecutive slopes s_1, s_2, $\forall s \in (s_1, s_2), \boldsymbol{w} := [-s, -1]^\top$, also defines the same AUC score. Since Algo. 1 iterates all such lines, the best AUC score of $\boldsymbol{w}$ returned by AUC-opt is indeed optimal.

AUC-opt finishes in $\mathcal{O}(n_+n_- \log(n_+n_-))$ since the time complexity is dominated by sorting all n_+n_- slopes (L10). The tightness of time complexity follows by Lemma 3.6.16 of Lee and Preparata (1984). ■

Generalization to $\mathbb{R}^d$

When problem dimension $d \geq 3$, inspired from Johnson and Preparata (1978), the general idea of solving high-dimensional LAO problem is that one can decompose the original d dimensional problem into several $d-1$ subproblems. Notice that each hyperplane $H(\boldsymbol{u})$ uniquely defines an interesting subspace, and there are at most n_+n_- such subspaces. We project points onto $H(\boldsymbol{u})$ and then solve problem in $d-1$ dimensional subspace (changing the number of coordinates from d to $d-1$), recursively. Specifically, let the projection be defined as $\boldsymbol{P}(\boldsymbol{x}) := \boldsymbol{x} - (\boldsymbol{x}^\top \cdot \boldsymbol{u}/\|\boldsymbol{u}\|^2) \cdot \boldsymbol{u}$ where $\boldsymbol{u}$ is the normal vector of $H(\boldsymbol{u})$. The critical property of $\boldsymbol{P}$ is that for any $\boldsymbol{x} \in H(\boldsymbol{u})$, $\boldsymbol{x}^\top \boldsymbol{P}(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) = \boldsymbol{x}^\top(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)$. Therefore, the inner products of training samples with $\boldsymbol{w}$ are the same as those of projected training samples.

Due to this inevitable recursion, the number of interesting hyperplanes exponentially depends upon d; hence time complexity of AUC-opt in $\mathbb{R}^d$ is $\mathcal{O}\left((n_pn_q)^{d-1}\log(n_pn_q)\right)$. We summarize this recursive procedure in Algo. 2. Due to the space limitation, detailed algorithm description is in the supplementary.

Algorithm 2: $[\mathrm{AUC}_{\mathrm{opt}}, \boldsymbol{w}]$ =AUC-opt($\mathcal{D}, d$)

```
K = {(x_i^+ − x_j^−) : i ∈ {1,...,n_+}, j ∈ {1,...,n_−}}
if d=2 then
    return AUC_cur, w' = AUC-opt(D)                    ▷ call Algo. 1
end if
for u ∈ K do
    P = {(x − (x^⊤·u)/‖u‖² · u, y) : (x, y) ∈ D}        ▷ project points of D onto the hyperplane defined by u.
    P' = change_coordinates(P)                          ▷ change coordinates so that points are presented in d − 1 coordinates.
    AUC_cur, w' = AUC-opt(P', d − 1)
    w = change_coordinates(w')                          ▷ change d − 1 coordinates of w' back to d coordinates.
    if AUC_opt < AUC_cur then
        w* = w, AUC_opt = AUC_cur
    end if
end for
```

Theorem 3. *Given the dataset $\mathcal{D} := \{(\boldsymbol{x}_i, y_i) : i \in \{1, 2, \ldots, n\}\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$ and a collection of linear separators $\mathcal{H} := \{\boldsymbol{w} : \boldsymbol{w} \in \mathbb{R}^d\}$. There exists an algorithm solves the LAO problem (1) exactly in $\mathcal{O}\left((n_+n_-)^{d-1}\log(n_+n_-)\right)$.*

Proof. The proof is in the supplementary. ■

5 Experiments

We evaluate AUC-opt on both $\mathbb{R}^2$ and $\mathbb{R}^3$ by using t-SNE datasets. To confirm that AUC-opt produces the best possible linear AUC classifiers, we compare it with 7 other classifiers on the binary classification task at the training and testing stage. Furthermore, we compare the approximate AUC methods with other standard methods for nonlinear classifiers. *More results and experimental details, including data collection, baseline description, and parameter tuning, are in the supplementary.*[2]

Datasets and Experimental Setup

Datasets. We collect 50 real-world datasets where the positive ratio (n_p/n) of most datasets are ≤ 0.1. These highly imbalanced datasets make optimizing AUC problem meaningful. To generate 2 and 3 dimensional samples, we project samples of these datasets onto $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively using t-SNE (Maaten and Hinton 2008) so that class patterns are conserved. Models use projected points as training samples while keep labels unchanged.

[2]Our code can be found in https://github.com/baojian/auc-opt

		SVM	B-SVM	LR	B-LR	SVM-Perf	SPAUC	SPAM	AUC-opt	SVM	B-SVM	LR	B-LR	SVM-Perf	SPAUC	SPAM	AUC-opt
		Significance t-test ($\alpha = 0.05$) on *training*								Significance t-test ($\alpha = 0.05$) on *testing*							
	SVM	-	0	1	1	3	0	1	0	-	0	0	0	3	0	1	1
	B-SVM	30	-	7	4	27	3	3	0	31	-	7	4	23	3	3	1
	LR	31	11	-	2	27	2	5	0	32	14	-	2	26	2	5	2
	B-LR	32	15	6	-	30	2	4	0	33	16	6	-	29	3	4	2
$d = 2$	SVM-Perf	16	2	0	0	-	0	1	0	16	2	0	1	-	1	2	2
	SPAUC	31	14	8	2	28	-	2	0	29	12	7	3	26	-	4	2
	SPAM	33	10	9	3	29	1	-	0	32	11	6	3	26	1	-	2
	AUC-opt	**40**	**29**	**20**	**17**	**40**	**21**	**26**	-	**34**	**18**	**15**	**11**	**31**	**13**	**17**	-

Table 1. The comparison of AUC scores over 200 trials of 50 t-SNE datasets on $\mathbb{R}^2$. Each cell (I,J) means the number of datasets where method I is significantly better than method J by using t-test with a significance level of 5%. Numbers in the red region are the results of AUC optimizers better than standard classifiers, while numbers in the blue region are the reverse.

Experimental setup. For each dataset, 50% samples are for training and the rest for testing. All parameters are tuned by 5-fold cross-validation. Each dataset is randomly shuffled 200 times, and the reported results are averaged on 200 trials. All methods have been tested on servers with Intel(R) Xeon(R) CPU (2.30GHz) 64 cores and 187G memory. For all methods that involve randomness, the random state has been fixed for the purpose of reproducibility.

Baselines. We consider the following baseline classifiers: 1) Logistic Regression (LR); 2) B-LR. Balanced Logistic Regression (B-LR). We adjust weights of samples inversely proportional to class frequencies so that it can have better performance on imbalanced datasets; 3) Support-vector Machine (SVM); 4) B-SVM. The balanced SVM (B-SVM) uses the same strategy as B-LR; 5) SVM-Perf. The SVM-Perf algorithm is proposed in Joachims (2005) where the goal is to minimize the AUC loss by using SVM-based method; 6) SPAUC. The Stochastic Proximal AUC (SPAUC) maximization algorithm is proposed in Lei and Ying (2019); 7) SPAM. The Stochastic Proximal AUC Maximization (SPAM) algorithm is proposed in Natole, Ying, and Lyu (2018).

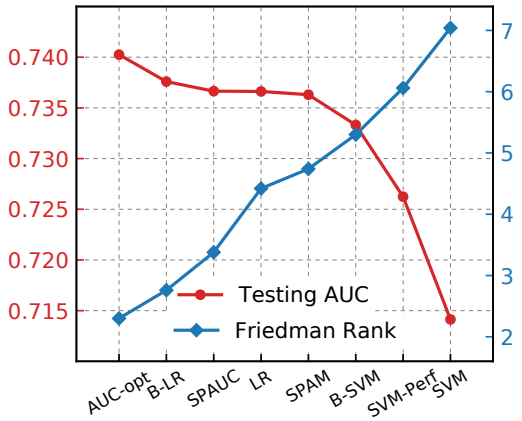

Figure 3: Mean of testing AUC

Results on T-SNE Datasets

Comparison of AUC scores. Table 1 presents the comparison of AUC scores calculated on both training and testing datasets in $\mathbb{R}^2$ and $\mathbb{R}^3$. To compare AUC scores from different methods, for the method I and J on a specific dataset, we calculate whether I is significantly better than J in a statistical sense. Important observations are: 1) AUC-opt achieves more significant gains over standard classifiers than the approximate AUC optimizers on the training stage. It confirms that gaps between AUC-opt and other approximate AUC optimizers are significant on some datasets; 2) Compared with SVM and LR, balanced versions of SVM (B-SVM) and LR (B-LR) have better performance. We see that the simple weighting strategy improves the performance by adjusting the weights on training samples;[3] and 3) Compared with the

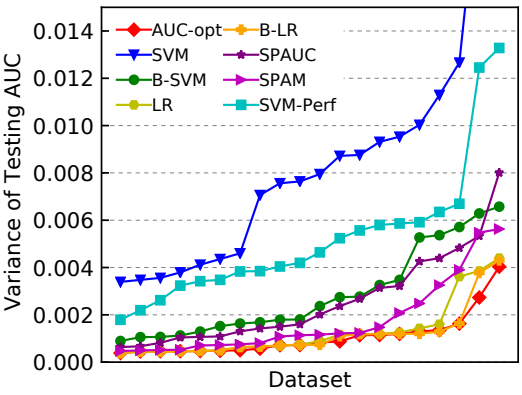

Figure 4: Variance of testing AUC

[3]The balanced version of a classifier is that a class weight has been added for each sample. Specifically, weight $n/(2 * n_p)$ is for

	SVM	B-SVM	LR	B-LR	RF	B-RF	GB	SVM-RBF	B-SVM-RBF	RankBoost	AdaBoost	SPAM	SPAUC	SVM-Perf	**Average**
SVM	-	4	2	3	10	9	15	19	8	15	17	29	30	3	12.62
B-SVM	22	-	6	4	15	12	18	21	12	15	19	36	38	20	18.31
LR	31	21	-	5	17	14	19	23	14	18	20	38	38	29	22.08
B-LR	31	23	8	-	17	14	19	23	14	19	21	37	37	28	22.38
RF	34	30	31	31	-	6	34	29	19	34	38	39	37	34	30.46
B-RF	36	33	31	31	19	-	33	32	19	35	37	38	36	35	31.92
GB	31	29	29	29	12	13	-	25	17	26	30	36	38	32	26.69
SVM-RBF	27	26	26	27	16	15	21	-	6	23	23	30	31	27	22.92
B-SVM-RBF	36	32	32	32	24	22	29	20	-	31	33	39	38	37	31.15
RankBoost	32	31	29	29	9	8	16	24	13	-	35	38	38	33	25.77
AdaBoost	31	29	26	27	7	8	9	22	12	3	-	37	35	31	21.31
SPAM	13	4	2	2	7	7	12	18	7	8	12	-	27	15	10.31
SPAUC	6	3	2	1	9	7	10	17	7	10	10	10	-	8	7.69
SVM-Perf	3	10	4	5	9	8	15	16	7	13	16	25	28	-	12.23

Table 2. The comparison of testing AUC scores of 50 real-world datasets. The settings are the same as in Table 1.

best standard classifier B-LR, AUC-opt produces significant gains on 17 training datasets while reduced to 11 on testing datasets. It is worse in $\mathbb{R}^3$. This degradation happens mainly because results of B-LR are tuned by adding regularization hence better generalization ability, while AUC-opt does not. With regularization, B-LR has better generalization performance on testing datasets, but regularization is not taken into consideration in our method. Under $\mathbb{R}^3$, AUC-opt only beats 4 datasets over B-LR, the best classifier. It means that the gain proves insignificant on most datasets.

We also report the mean with the Friedman Rank and variance of AUC scores in Fig. 3 and 4, respectively. Results indicate the superiority of AUC-opt in optimizing AUC scores. Variances of AUC shown in (c) and (d) indicate that 0-1 objective optimization (AUC-opt) is more robust, while SVMs are not. Yet, the performance of SVM-Perf is better than SVM but not B-SVM.

Results on Real-world Datasets

We test both the approximate AUC optimizers and the standard classification methods on 50 real-world datasets. The AUC scores on testing are reported in Table 2. First, Balanced Random Forest (B-RF) and Balanced SVM-RBF (B-SVM-RBF) prove the best overall on testing AUC scores. It wins 31.92 and 31.15 datasets on average respectively. This performance is consistent with findings shown in both Fernández-Delgado et al. (2014) and Couronné, Probst, and Boulesteix (2018). Furthermore, a relationship between RF and LR has been studied in Couronné, Probst, and Boulesteix (2018) where it has been shown that RF can obtain much higher AUC scores compared with LR. Boosting-based methods such as AdaBoost[4] and Gradient Boost (GB) also work well.

RankBoost is inferior to RF and B-RF, winning on only 9 and 8 datasets, respectively. The performance of AdaBoost and RankBoost prove competitive with each other. This has been theoretically justified in Rudin and Schapire (2009). However, interestingly, RankBoost still outperforms AdaBoost over 35 datasets in the testing stage. Gradient Boost (GB) wins more datasets than RankBoost. Generally speaking, two nonlinear AUC optimizers are inferior to these popular nonlinear standard classifiers. This clearly suggests that approximate AUC optimizers may not lead to the best AUC performance, hence having space to improve. All linear methods lose on average to non-linear ones.

6 Conclusion and Future Work

Our complexity results show linear AUC optimization is NP-complete via reduction to the open hemisphere problem. It remains interesting to prove the hardness results for other hypothesis classes mentioned in Ben-David, Eiron, and Long (2003). We then present an optimal method AUC-opt that is both time and space-efficient for optimizing AUC in $\mathbb{R}^2$. We demonstrate that it can be naturally extended to $\mathbb{R}^d$ with a total cost $\mathbb{O}((n_+n_-)^{d-1}\log(n_+n_-))$. Our empirical results suggest that to justify the objective to optimize AUC, more effort may be needed to improve the optimization quality of AUC optimizers.

AUC-opt is impractical for real-world datasets since the time complexity is exponentially getting worse with the dimension d. However, it remains interesting to see whether more efficient algorithms exist for higher dimensionality. One potential direction is to use branch and bound Nguyen and Sanner (2013). It is also interesting to compare our method with Rudin and Wang (2018), a recently proposed method that directly optimizes a rerank statistic.

each positive sample while weight $n/(2 * n_q)$ is for each negative sample.

[4]We treat AdaBoost as the AUC-based method because theoretical finding indicates that AdaBoost is equivalent to RankBoost.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. The work of Baojian Zhou is partially supported by startup funding from Fudan University. Steven Skiena was partially supported by NSF grants IIS-1926781, IIS-1927227, IIS-1546113, OAC-191952, and a New York State Empire Innovation grant.

References

Ben-David, S.; Eiron, N.; and Long, P. M. 2003. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3): 496–514.

Ben-Or, M. 1983. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 80–86.

Bradley, A. P. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7): 1145–1159.

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

Calders, T.; and Jaroszewicz, S. 2007. Efficient AUC optimization for classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*, 42–53. Springer.

Clémençon, S.; Lugosi, G.; Vayatis, N.; et al. 2008. Ranking and empirical minimization of U-statistics. *The Annals of Statistics*, 36(2): 844–874.

Cohen, W. W.; Schapire, R. E.; and Singer, Y. 1998. Learning to order things. In *NIPS*, 451–457.

Cortes, C.; and Mohri, M. 2004. AUC optimization vs. error rate minimization. In *NIPS*, 313–320.

Couronné, R.; Probst, P.; and Boulesteix, A.-L. 2018. Random forest versus logistic regression: a large-scale benchmark experiment. *BMC Bioinformatics*, 19(1): 270.

Eban, E.; Schain, M.; Mackey, A.; Gordon, A.; Rifkin, R.; and Elidan, G. 2017. Scalable Learning of Non-Decomposable Objectives. In *Artificial Intelligence and Statistics*, 832–840.

Edelsbrunner, H.; and Guibas, L. J. 1989. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38(1): 165–194.

Elizondo, D. 2006. The linear separability problem: Some testing methods. *IEEE Transactions on Neural Networks*, 17(2): 330–344.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8): 861–874.

Feldman, V.; Guruswami, V.; Raghavendra, P.; and Wu, Y. 2012. Agnostic learning of monomials by halfspaces is hard. *SIAM Journal on Computing*, 41(6): 1558–1590.

Fernández-Delgado, M.; Cernadas, E.; Barro, S.; and Amorim, D. 2014. Do we need hundreds of classifiers to solve real world classification problems? *JMLR*, 3133–3181.

Ferri, C.; Flach, P. A.; and Hernández-Orallo, J. 2002. Learning Decision Trees Using the Area Under the ROC Curve. In *ICML*, 139–146.

Freund, Y.; Iyer, R.; Schapire, R. E.; and Singer, Y. 2003. An efficient boosting algorithm for combining preferences. *JMLR*, 933–969.

Freund, Y.; and Schapire, R. E. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3): 277–296.

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

Gao, W.; and Zhou, Z.-H. 2015. On the consistency of AUC pairwise optimization. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 939–945.

Gultekin, S.; Saha, A.; Ratnaparkhi, A.; and Paisley, J. 2020. MBA: mini-batch AUC optimization. *IEEE Transactions on Neural Networks and Learning Systems*.

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.

Herschtal, A.; and Raskutti, B. 2004. Optimising area under the ROC curve using gradient descent. In *ICML*, 49.

Joachims, T. 2005. A support vector method for multivariate performance measures. In *ICML*, 377–384.

Johnson, D. S.; and Preparata, F. P. 1978. The densest hemisphere problem. *Theoretical Computer Science*, 6(1): 93–107.

Kallus, N.; and Zhou, A. 2019. The fairness of risk scores beyond classification: Bipartite ranking and the xAUC metric. In *NIPS*, 3433–3443.

Kotłowski, W.; Dembczyński, K.; and Hüllermeier, E. 2011. Bipartite ranking through minimization of univariate loss. In *ICML*, 1113–1120.

Le, Q. V.; Smola, A.; Chapelle, O.; and Teo, C. H. 2010. Optimization of ranking measures. *JMLR*, 1(2999): 1–48.

Lee, D.-T.; and Preparata, F. P. 1984. Computational geometry? a survey. *IEEE Transactions on Computers*, 1072–1101.

Lei, Y.; and Ying, Y. 2019. Stochastic Proximal AUC Maximization. *arXiv preprint arXiv:1906.06053*.

Liu, M.; Yuan, Z.; Ying, Y.; and Yang, T. 2020. Stochastic auc maximization with deep neural networks. In *International Conference on Learning Representations (ICLR)*.

Liu, M.; Zhang, X.; Chen, Z.; Wang, X.; and Yang, T. 2018. Fast Stochastic AUC Maximization with $O(1/n)$-Convergence Rate. In *ICML*, 3189–3197.

Maaten, L. v. d.; and Hinton, G. 2008. Visualizing data using t-SNE. *JMLR*, 2579–2605.

Menon, A. K.; and Williamson, R. C. 2016. Bipartite Ranking: a Risk-Theoretic Perspective. *JMLR*, 17(195): 1–102.

Natole, M.; Ying, Y.; and Lyu, S. 2018. Stochastic proximal algorithms for AUC maximization. In *ICML*, 3710–3719.

Nguyen, T. T.; and Sanner, S. 2013. Algorithms for direct 0-1 loss optimization in binary classification. In *ICML*, 1085–1093.

Provost, F.; and Fawcett, T. 2001. Robust classification for imprecise environments. *Machine Learning*, 42(3): 203–231.

Qin, T.; Liu, T.-Y.; and Li, H. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, 13(4): 375–397.

Rakotomamonjy, A. 2004. Optimizing AUC with support vector machine. In *European Conference on Artificial Intelligence Workshop on ROC Curve and AI*, 469–478.

Rudin, C.; and Schapire, R. E. 2009. Margin-based ranking and an equivalence between AdaBoost and RankBoost. *JMLR*, 2193–2232.

Rudin, C.; and Wang, Y. 2018. Direct Learning to Rank And Rerank. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, 775–783. PMLR.

Steck, H. 2007. Hinge rank loss and the area under the ROC curve. In *European Conference on Machine Learning*, 347–358. Springer.

Vogel, R.; Bellet, A.; Clémen, S.; et al. 2021. Learning Fair Scoring Functions: Bipartite Ranking under ROC-based Fairness Constraints. In *International Conference on Artificial Intelligence and Statistics*, 784–792. PMLR.

Yan, L.; Dodier, R. H.; Mozer, M.; and Wolniewicz, R. H. 2003. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *ICML*, 848–855.

Yang, T.; and Ying, Y. 2022. AUC maximization in the era of big data and AI: A survey. *ACM Computing Surveys (CSUR)*.

Ying, Y.; Wen, L.; and Lyu, S. 2016. Stochastic online AUC maximization. In *NIPS*, 451–459.

Yogananda, A.; Murty, M. N.; and Gopal, L. 2007. A fast linear separability test by projection of positive points on subspaces. In *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*, 713–720.

Zhao, P.; Hoi, S. C.; Jin, R.; and Yang, T. 2011. Online AUC maximization. In *ICML*, 233–240.