One-Pass AUC Optimization

Wei Gao GAOW@LAMDA.NJU.EDU.CN
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Rong Jin RONGJIN@CSE.MSU.EDU
Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, 48824, USA

Shenghuo Zhu ZSH@NEC-LABS.COM
NEC Laboratories America, CA, 95014, USA

Zhi-Hua Zhou ZHOUZH@LAMDA.NJU.EDU.CN
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Abstract

AUC is an important performance measure and many algorithms have been devoted to AUC optimization, mostly by minimizing a surrogate convex loss on a training data set. In this work, we focus on one-pass AUC optimization that requires going through the training data only once without storing the entire training dataset, where conventional online learning algorithms cannot be applied directly because AUC is measured by a sum of losses defined over pairs of instances from different classes. We develop a regression-based algorithm which only needs to maintain the first and second-order statistics of training data in memory, resulting a storage requirement independent from the size of training data. To efficiently handle high-dimensional data, we develop a randomized algorithm that approximates the covariance matrices by low-rank matrices. We verify, both theoretically and empirically, the effectiveness of the proposed algorithm.

1. Introduction

AUC (Area Under ROC curve) (Metz, 1978; Hanley & McNeil, 1983) is an important performance measure that has been widely used in many tasks (Provost et al., 1998; Cortes & Mohri, 2004; Liu et al., 2009; Flach et al., 2011). Many algorithms have been developed to optimize AUC based on surrogate losses (Herschtal & Raskutti, 2004; Joachims, 2006; Rudin & Schapire, 2009; Kotlowski et al., 2011; Zhao et al., 2011).

In this work, we focus on AUC optimization that requires only one pass of training examples. This is particularly important for applications involving big data or streaming data in which a large volume of data come in a short time period, making it infeasible to store the entire data set in memory before an optimization procedure is applied. Although many online learning algorithms have been developed to find the optimal solution of some performance measures by only scanning the training data once (Cesa-Bianchi & Lugosi, 2006), few effort addresses one-pass AUC optimization.

Unlike the classical classification and regression problems where the loss function can be calculated on a single training example, AUC is measured by the losses defined over pairs of instances from different classes, making it challenging to develop algorithms for one-pass optimization. An online AUC optimization algorithm was proposed very recently by Zhao et al. (2011). It is based on the idea of reservoir sampling, and achieves a solid regret bound by only storing $\sqrt{T}$ instances, where T is the number of training examples. Ideally, for one-pass approaches, it is crucial that the storage required by the learning process should be independent from the amount of training data, because it is often quite difficult to expect how many data will be received in those applications.

In this work, we propose a regression-based algorithm for one-pass AUC optimization in which a square loss

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

is used to measure the ranking error between two instances from different classes. The main advantage of using the square loss lies in the fact that it only needs to store the first and second-order statistics for the received training examples. Consequently, the storage requirement is reduced to $O(d^2)$, where d is the dimension of data, independent from the number of training examples. To deal with high-dimensional data, we develop a randomized algorithm that approximates the covariance matrix of $d \times d$ by a low-rank matrix. We show, both theoretically and empirically, the effectiveness of our proposal algorithm by comparing to state-of-the-art algorithms for AUC optimization.

Section 2 introduces some preliminaries. Sections 3 proposes the OPAUC (One Pass AUC) framework. Section 4 provides theoretical analysis. Section 5 presents our experimental results. Section 6 concludes with future work.

2. Preliminaries

We denote by $\mathcal{X} \in \mathbb{R}^d$ an instance space and $\mathcal{Y} = \{+1, -1\}$ the label set, and let $\mathcal{D}$ denote an unknown (underlying) distribution over $\mathcal{X} \times \mathcal{Y}$. A training sample of n_+ positive instances and n_- negative ones

$$\mathcal{S} = \{(\mathbf{x}_1^+, +1), (\mathbf{x}_2^+, +1), \ldots, (\mathbf{x}_{n_+}^+, +1), \\ (\mathbf{x}_1^-, -1), (\mathbf{x}_2^-, -1), \ldots, (\mathbf{x}_{n_-}^-, -1)\}$$

is drawn identically and independently according to distribution $\mathcal{D}$, where we do not fix n_+ and n_- before the training sample is chosen. Let $f\colon \mathcal{X} \to \mathbb{R}$ be a real-valued function. Then, the AUC of function f on the sample $\mathcal{S}$ is defined as

$$\sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \frac{\mathbb{I}[f(\mathbf{x}_i^+) > f(\mathbf{x}_j^-)] + \frac{1}{2}\mathbb{I}[f(\mathbf{x}_i^+) = f(\mathbf{x}_j^-)]}{n_+ n_-}$$

where $\mathbb{I}[\cdot]$ is the indicator function which returns 1 if the argument is true and 0 otherwise.

Direct optimization of AUC often leads to an NP-hard problem as it can be cast into a combinatorial optimization problem. In practice, it is approximated by a convex optimization problem that minimizes the following objective function

$$\mathcal{L}(\mathbf{w}) = \frac{\lambda}{2}|\mathbf{w}|^2 + \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \frac{\ell\left(\mathbf{w}^\top(\mathbf{x}_i^+ - \mathbf{x}_j^-)\right)}{2n_+ n_-} \quad (1)$$

where ℓ is a convex loss function and λ is the regularization parameter that controls the model complexity. Notice that each loss term $\ell(\mathbf{w}^\top(\mathbf{x}_i^+ - \mathbf{x}_j^-))$ involves two instances from different classes; therefore, it is difficult to extend online learning algorithms for one-pass AUC optimization without storing all the training instances. Zhao et al. (2011) addressed this challenge by exploiting the reservoir sampling technique.

3. The OPAUC Approach

To address the challenge of one-pass AUC optimization, we propose to use the square loss in Eq. (1), i.e.,

$$\mathcal{L}(\mathbf{w}) = \frac{\lambda}{2}|\mathbf{w}|^2 + \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \frac{\left(1 - \mathbf{w}^\top(\mathbf{x}_i^+ - \mathbf{x}_j^-)\right)^2}{2n_+ n_-}. \quad (2)$$

The main advantage of using the square loss lies in the fact that it is sufficient to store the first and second-order statistics of training examples for optimization, leading to a memory requirement of $O(d^2)$, which is independent from the number of training examples. Another advantage is that the square loss is consistent with AUC, as will be shown by Theorem 1 (Section 4). In contrast, loss functions such as hinge loss are proven to be inconsistent with AUC (Gao & Zhou, 2012).

As aforementioned, the classical online setting cannot be applied to one-pass AUC optimization because, even if the optimization problem of Eq. (2) has a closed form, it requires going through the training examples multiple times. To address this challenge, we modify the overall loss $\mathcal{L}(\mathbf{w})$ in Eq. (2) (with a little variation) as a sum of losses for individual training instance $\sum_{t=1}^{T} \mathcal{L}_t(\mathbf{w})$, where

$$\mathcal{L}_t(\mathbf{w}) = \frac{\lambda}{2}|\mathbf{w}|^2 + \frac{\sum_{i=1}^{t-1} \mathbb{I}[y_i \neq y_t](1 - y_t(\mathbf{x}_t - \mathbf{x}_i)^\top \mathbf{w})^2}{2|\{i \in [t-1] : y_i y_t = -1\}|}$$

for i.i.d. sequence $\mathcal{S}_t = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)\}$, and it is an unbiased estimation to $\mathcal{L}(\mathbf{w})$. For simplicity, we denote by X_t^+ and X_t^- the sets of positive and negative instances in sequence $\mathcal{S}_t$, respectively, and we further denote by T_t^+ and T_t^- their respective cardinalities. Also, we set $\mathcal{L}_t(\mathbf{w}) = 0$ for $T_t^+ T_t^- = 0$.

If $y_t = 1$, we calculate the gradient as

$$\nabla \mathcal{L}_t(\mathbf{w}) = \lambda \mathbf{w} + \mathbf{x}_t \mathbf{x}_t^\top \mathbf{w} - \mathbf{x}_t \\ + \sum_{i:\, y_i = -1} \left(\mathbf{x}_i + (\mathbf{x}_i \mathbf{x}_i^\top - \mathbf{x}_i \mathbf{x}_t^\top - \mathbf{x}_t \mathbf{x}_i^\top)\mathbf{w}\right) / T_t^-. \quad (3)$$

It is easy to observe that $c_t^- = \sum_{i:\, y_i = -1} \mathbf{x}_i / T_t^-$ and $S_t^- = \sum_{i:\, y_i = -1} (\mathbf{x}_i \mathbf{x}_i^\top - \mathbf{c}_t^- [\mathbf{c}_t^-]^\top) / T_t^-$ correspond to the mean and covariance matrix of negative class, respectively; thus, Eq. (3) can be further simplified as

$$\nabla \mathcal{L}_t(\mathbf{w}) = \lambda \mathbf{w} - \mathbf{x}_t + \mathbf{c}_t^- \\ + (\mathbf{x}_t - \mathbf{c}_t^-)(\mathbf{x}_t - \mathbf{c}_t^-)^\top \mathbf{w} + S_t^- \mathbf{w}. \quad (4)$$

Algorithm 1 The OPAUC Algorithm

Input: The regularization parameter $\lambda > 0$ and stepsizes $\{\eta_t\}_{t=1}^T$.
Initialization: Set $T_0^+ = T_0^- = 0$, $\mathbf{c}_0^+ = \mathbf{c}_0^- = \mathbf{0}$, $\mathbf{w}_0 = \mathbf{0}$ and $\Gamma_0^+ = \Gamma_0^- = [\mathbf{0}]_{d\times u}$ for some $u > 0$

1: **for** $t = 1, 2, \ldots, T$ **do**
2: Receive a training example $(\mathbf{x}_t, y_t)$
3: **if** $y_t = +1$ **then**
4: $T_t^+ = T_{t-1}^+ + 1$ and $T_t^- = T_{t-1}^-$;
5: $\mathbf{c}_t^+ = \mathbf{c}_{t-1}^+ + \frac{1}{T_t^+}(\mathbf{x}_t - \mathbf{c}_{t-1}^+)$ and $\mathbf{c}_t^- = \mathbf{c}_{t-1}^-$;
6: Update Γ_t^+ and $\Gamma_t^- = \Gamma_{t-1}^-$;
7: Calculate the gradient $\widehat{g}_t(\mathbf{w}_{t-1})$
8: **else**
9: $T_t^- = T_{t-1}^- + 1$ and $T_t^+ = T_{t-1}^+$;
10: $\mathbf{c}_t^- = \mathbf{c}_{t-1}^- + \frac{1}{T_t^-}(\mathbf{x}_t - \mathbf{c}_{t-1}^-)$ and $\mathbf{c}_t^+ = \mathbf{c}_{t-1}^+$;
11: Update Γ_t^- and $\Gamma_t^+ = \Gamma_{t-1}^+$;
12: Calculate the gradient $\widehat{g}_t(\mathbf{w}_{t-1})$
13: **end if**
14: $\mathbf{w}_t = \mathbf{w}_{t-1} - \eta_t \widehat{g}_t(\mathbf{w}_{t-1})$
15: **end for**

In a similar manner, we calculate the following gradient for $y_t = -1$:

$$\nabla \mathcal{L}_t(\mathbf{w}) = \lambda \mathbf{w} + \mathbf{x}_t - \mathbf{c}_t^+ + (\mathbf{x}_t - \mathbf{c}_t^+)(\mathbf{x}_t - \mathbf{c}_t^+)^\top \mathbf{w} + S_t^+ \mathbf{w} \quad (5)$$

where $S_t^+ = \sum_{i:\, y_i=1}(\mathbf{x}_i\mathbf{x}_i^\top - \mathbf{c}_t^+[\mathbf{c}_t^+]^\top)/T_t^+$ and $\mathbf{c}_t^+ = \sum_{i:\, y_i=1}\mathbf{x}_i/T_t^+$ are the covariance matrix and mean of positive class, respectively.

The storage cost for keeping the class means ($\mathbf{c}_t^+$ and $\mathbf{c}_t^-$) and covariance matrices (S_{t-1}^+ and S_{t-1}^-) is $O(d^2)$. Once we get the gradient $\nabla \mathcal{L}_t(\mathbf{w})$, by theory of stochastic gradient descent, the solution can be updated by $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \mathcal{L}_t(\mathbf{w}_t)$, where η_t is the stepsize for the t-th iteration.

Algorithm 1 highlights the key steps of the proposed algorithm. We initialize $\Gamma_0^- = \Gamma_0^+ = [\mathbf{0}]_{d\times d}$, where $u = d$. At each iteration, we set $\Gamma_t^+ = S_t^+$ and $\Gamma_t^- = S_t^-$, and update Γ_t^+ (Line 6) and Γ_t^- (Line 11), respectively, by using the following equations

$$\Gamma_t^+ = \Gamma_{t-1}^+ + \frac{\mathbf{x}_t\mathbf{x}_t^\top - \Gamma_{t-1}^+}{T_t^+} + \mathbf{c}_{t-1}^+[\mathbf{c}_{t-1}^+]^\top - \mathbf{c}_t^+[\mathbf{c}_t^+]^\top,$$
$$\Gamma_t^- = \Gamma_{t-1}^- + \frac{\mathbf{x}_t\mathbf{x}_t^\top - \Gamma_{t-1}^-}{T_t^-} + \mathbf{c}_{t-1}^-[\mathbf{c}_{t-1}^-]^\top - \mathbf{c}_t^-[\mathbf{c}_t^-]^\top.$$

Finally, the stochastic gradient $\widehat{g}_t(\mathbf{w}_{t-1})$ of Lines 7 and 12 in Algorithm 1 are given by $\nabla \mathcal{L}_t(\mathbf{w}_{t-1})$ that are calculated by Eqs. (4) and (5), respectively.

Dealing with High-Dimensional Data. One limitation of the approach in Algorithm 1 is that the storage cost of the two covariance matrices S_t^+ and S_t^- is $O(d^2)$, making it unsuitable for high-dimensional data. We tackle this by developing a randomized algorithm that approximates the covariance matrices by low-rank matrices. We are motivated by the observation that S_t^+ and S_t^- can be written, respectively, as

$$S_t^+ = \tfrac{1}{T_t^+}\left(X_t^+ - \mathbf{c}_t^+\mathbf{1}_{T_t^+}^\top\right) I_{T_t^+}\left(X_t^+ - \mathbf{c}_t^+\mathbf{1}_{T_t^+}\right)^\top,$$
$$S_t^- = \tfrac{1}{T_t^-}\left(X_t^- - \mathbf{c}_t^-\mathbf{1}_{T_t^-}^\top\right) I_{T_t^-}\left(X_t^- - \mathbf{c}_t^-\mathbf{1}_{T_t^-}\right)^\top,$$

where I_t is an identity matrix of size $t \times t$ and $\mathbf{1}_t$ is an all-one vector of size t. To approximate S_t^+ and S_t^-, we approximate the identify matrix I_t by a matrix of rank $\tau \ll d$. To this end, we randomly sample $\mathbf{r}_i \in \mathbb{R}^\tau, i = 1, \ldots, t$ from a Gaussian distribution $\mathcal{N}(0, I_\tau)$, and approximate I_t by $R_t R_t^\top$, where $R_t = \frac{1}{\tau}(\mathbf{r}_1, \ldots, \mathbf{r}_t)^\top \in \mathbb{R}^{t\times\tau}$. We further divide R_t into two matrices where $R_t^+ \in \mathbb{R}^{T_t^+\times\tau}$ and $R_t^- \in \mathbb{R}^{T_t^-\times\tau}$ that contain the subset of the rows in R_t corresponding to all the positive and negative instances received before the t-th iteration, respectively. Therefore, the covariance matrices S_t^+ and S_t^- can be approximated, respectively, by

$$\widehat{S}_t^+ = \tfrac{1}{T_t^+} Z_t^+ [Z_t^+]^\top - \widehat{\mathbf{c}}_{t-1}^+ [\widehat{\mathbf{c}}_{t-1}^+]^\top,$$
$$\widehat{S}_t^- = \tfrac{1}{T_t^-} Z_t^- [Z_t^-]^\top - \widehat{\mathbf{c}}_{t-1}^- [\widehat{\mathbf{c}}_{t-1}^-]^\top,$$

where $Z_t^+ = X_t^+ R_t^+$, $\widehat{\mathbf{c}}_t^+ = \mathbf{c}_t^+\mathbf{1}_{T_t^+}^\top R_t^+/T_t^+$, $Z_t^- = X_t^- R_t^-$ and $\widehat{\mathbf{c}}_t^- = \mathbf{c}_t^-\mathbf{1}_{T_t^-}^\top R_t^-/T_t^-$. Based on approximate covariance matrix $\widehat{S}_t^\pm$, the approximation algorithm essentially tries to minimize $\sum_{t=1}^T \widehat{\mathcal{L}}_t(\mathbf{w})$, where

$$\widehat{\mathcal{L}}_t(\mathbf{w}) = \mathbf{w}^\top(\mathbf{c}_{t-1}^- - \mathbf{x}_t) + (1 + \mathbf{w}^\top \widehat{S}_t^- \mathbf{w})/2 + \lambda|\mathbf{w}|^2/2 + \mathbf{w}^\top(\mathbf{x}_t - \mathbf{c}_{t-1}^-)(\mathbf{x}_t - \mathbf{c}_{t-1}^-)^\top \mathbf{w}/2 \quad (6)$$

if $y_t = 1$; otherwise,

$$\widehat{\mathcal{L}}_t(\mathbf{w}) = \mathbf{w}^\top(\mathbf{x}_t - \mathbf{c}_{t-1}^+) + (1 + \mathbf{w}^\top \widehat{S}_t^+ \mathbf{w})/2 + \lambda|\mathbf{w}|^2/2 + \mathbf{w}^\top(\mathbf{x}_t - \mathbf{c}_{t-1}^+)(\mathbf{x}_t - \mathbf{c}_{t-1}^+)^\top \mathbf{w}/2. \quad (7)$$

Further, we have the following recursive formulas:

$$Z_t^+ = Z_{t-1}^+ + \mathbf{x}_t\mathbf{r}_t^\top \mathbb{I}[y_t = +1]/\sqrt{m}, \quad (8)$$
$$Z_t^- = Z_{t-1}^- + \mathbf{x}_t\mathbf{r}_t^\top \mathbb{I}[y_t = -1]/\sqrt{m}. \quad (9)$$

It is important to notice that we do not need to calculate and store the approximate covariance matrices $\widehat{S}_t^+$ and $\widehat{S}_t^-$ explicitly. Instead, we only need to maintain

matrices Z_t^+ and Z_t^- in memory. This is because the stochastic gradient $\widehat{g}_t(\mathbf{w})$ based on the approximate covariance matrices can be computed directly from Z_t^+ and Z_t^-. More specifically, $\widehat{g}_t(\mathbf{w})$ is computed as

$$\widehat{g}_t(\mathbf{w}) = \mathbf{c}_{t-1}^- - \mathbf{x}_t + \lambda\mathbf{w} + (\mathbf{x}_t - \mathbf{c}_{t-1}^-)(\mathbf{x}_t - \mathbf{c}_{t-1}^-)^\top \mathbf{w} + \left(Z_t^-[Z_t^-]^\top/T_t^- - \widehat{\mathbf{c}}_{t-1}^-[\widehat{\mathbf{c}}_{t-1}^-]^\top\right)\mathbf{w} \quad (10)$$

for $y_t = 1$; otherwise

$$\widehat{g}_t(\mathbf{w}) = \mathbf{x}_t - \mathbf{c}_{t-1}^+ + \lambda\mathbf{w} + (\mathbf{x}_t - \mathbf{c}_{t-1}^+)(\mathbf{x}_t - \mathbf{c}_{t-1}^+)^\top \mathbf{w} + \left(Z_t^+[Z_t^+]^\top/T_t^+ - \widehat{\mathbf{c}}_{t-1}^+[\widehat{\mathbf{c}}_{t-1}^+]^\top\right)\mathbf{w}. \quad (11)$$

We require a memory of $O(\tau d)$ instead of $O(d^2)$ to calculate $\widehat{g}_t(\mathbf{w})$ by using the trick $A[A]^\top\mathbf{w} = A([A]^\top\mathbf{w})$, where $A \in \mathbb{R}^{d\times 1}$ or $\mathbb{R}^{d\times\tau}$.

To implement the approximate approach, we initialize $\Gamma_0^- = \Gamma_0^+ = [\mathbf{0}]_{d\times\tau}$ in Algorithm 1, where $u = \tau$. At each iteration, we set $\Gamma_t^+ = Z_t^+$ and $\Gamma_t^- = Z_t^-$, and compute the gradient $\widehat{g}_t(\mathbf{w}_{t-1})$ of Lines 7 and 12 in Algorithm 1 by Eqs. (10) and (11), respectively. Γ_t^+ and Γ_t^- are updated by Eqs. (8) and (9), respectively.

Remark. An alternative approach for the high-dimensional case is through the random projection (Johnstone, 2006; Hsu et al., 2012). Let $H \in \mathbb{R}^{d\times\tau}$ be a random Gaussian matrix, where $\tau \ll d$. By performing random projection using H, we compute a low-dimensional representation for each instance $\mathbf{x}_t$ as $\widehat{\mathbf{x}}_t = H^\top\mathbf{x}_t \in \mathbb{R}^\tau$ and will only maintain covariance matrices of size $\tau\times\tau$ in memory. Despite that it is computationally attractive, this approach performs significantly worse than the randomized low-rank approximation algorithm, according to our empirical study. This may owe to the fact that the random projection approach is equivalent to approximating $S_t^\pm = I_d S_t^\pm I_d$ by $HH^\top S_t^\pm HH^\top$, which replaces both the left and right identity matrices of $S_t^\pm$ with $HH^\top$. In contrast, our proposed approach only approximates one identity matrix in $S_t^\pm$, making it more reliable for tackling high-dimensional data.

4. Main Theoretical Results

This section presents our main theoretical results. Due to the page limit, we present the detailed proofs and analysis in a longer version (Gao et al., 2013). We first prove the consistency of square loss:

Theorem 1 *For square loss $\ell(t) = (1-t)^2$, the surrogate loss $\Psi(f, \boldsymbol{x}, \boldsymbol{x}') = \ell(f(\boldsymbol{x}) - f(\boldsymbol{x}'))$ is consistent with AUC.*

Proof Sketch: Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ with marginal probability $p_i > 0$ and conditional probability ξ_i, and we denote by the expected risk

$$R_\Psi(f) = \sum\nolimits_{i\neq j} p_i p_j \big(\xi_i(1-\xi_j)\ell(f(\mathbf{x}_i) - f(\mathbf{x}_j)) + \xi_j(1-\xi_i)\ell(f(\mathbf{x}_j) - f(\mathbf{x}_i))\big) + C_0$$

where $\ell(t) = (1-t)^2$ and C_0 is a constant w.r.t. f. According to (Gao & Zhou, 2012), it suffices to prove that, for every solution f s.t. $R_\Psi(f) = \inf_{f'} R_\Psi(f')$, we have $f(\mathbf{x}_i) > f(\mathbf{x}_j)$ if $\xi_i > \xi_j$.

If $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2\}$, then minimizing $R_\phi(f)$ gives the optimal solution $f(\mathbf{x}_1) - f(\mathbf{x}_2) = \mathrm{sgn}(\xi_1 - \xi_2)$ for $\xi_1 \neq \xi_2$, and this shows the consistency.

If $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ for $n \geq 3$, and if $\xi_i(1-\xi_i) = 0$ for every $i \in [n]$, then minimizing $R_\phi(f)$ gives the optimal solution $f(\mathbf{x}_i) = f(\mathbf{x}_j) + 1$ for each pair of $\xi_i = 1$ and $\xi_j = 0$; this also shows the consistency.

If $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ for $n \geq 3$, and if $\xi_{i_0}(1-\xi_{i_0}) \neq 0$ for some $i_0 \in [n]$, then subgradient conditions give

$$\sum_{k\neq i} p_k(\xi_i + \xi_k - 2\xi_i\xi_k)(f(\mathbf{x}_i) - f(\mathbf{x}_k)) = \sum_{k\neq i} p_k(\xi_i - \xi_k)$$

for $i \in [n]$. Solving the above n linear equations yields

$$f(\mathbf{x}_i) - f(\mathbf{x}_j) = \frac{\xi_i - \xi_j}{\Delta} \prod_{k\neq i,j}^{n} \sum_{l=1} p_l(\xi_l + \xi_k - 2\xi_l\xi_k)$$

where $\Delta > 0$ is a polynomial in p_k and $\xi[k_1] + \xi[k_2] - 2\xi[k_1]\xi[k_2]$ for $k, k_1, k_2 \in [n]$. The theorem follows. ■

Define $\mathbf{w}_* = \arg\min_{\mathbf{w}} \sum_t \mathcal{L}_t(\mathbf{w})$. The following theorem shows the convergence rate for Algorithm 1 when the full covariance matrices are provided.

Theorem 2 *For $\|\mathbf{x}_t\| \leq 1$ $(t \in [T])$, $\|\mathbf{w}_*\| \leq B$ and $TL^* \geq \sum_{t=1}^T \mathcal{L}_t(\mathbf{w}_*)$, we have*

$$\sum\nolimits_t \mathcal{L}_t(\mathbf{w}_t) - \sum\nolimits_t \mathcal{L}_t(\mathbf{w}_*) \leq 2\kappa B^2 + B\sqrt{2\kappa TL^*},$$

where $\kappa = 4 + \lambda$ and $\eta_t = 1/(\kappa + \sqrt{(\kappa^2 + \kappa TL^*/B^2)})$.

This theorem presents an $O(1/T)$ convergence rate for the OPAUC algorithm if the distribution is separable, i.e., $L^* = 0$, and an $O(1/\sqrt{T})$ convergence rate for general case. Compared to the online AUC optimization algorithm (Zhao et al., 2011), which achieves at most $O(1/\sqrt{T})$ convergence rate, our proposed algorithm clearly reduce the regret. The faster convergence rate of our proposed algorithm owes to the smoothness of the square loss, an important property that has been explored by some studies of online learning (Rakhlin et al., 2012) and generalization error bound analysis (Srebro et al., 2010).

Remark: The bound in Theorem 2 does not explicitly explore the strongly convexity of $\mathcal{L}_t(\mathbf{w})$, which can lead to an $O(1/T)$ convergence rate. Instead, we focus on exploiting the smoothness of the loss function, since we did not introduce a bounded domain for $\mathbf{w}$. Due to the regularizer $\lambda|\mathbf{w}|^2/2$, we have $|\mathbf{w}_*| \leq 1/\lambda$, and it is reasonable to restrict $\mathbf{w}_t$ by $|\mathbf{w}_t| \leq 1/\lambda$, leading to a regret bound of $O(\ln T/[\lambda^3 T])$ by applying the standard stochastic gradient descent with $\eta_t = 1/[\lambda t]$. This bound is preferred only when $\lambda = \Omega(T^{-1/6})$, a scenario which rarely occurs in empirical study. This problem may also be addressable by exploiting the epoch gradient method (Nocedal & Wright, 1999), a subject of future study.

We now consider the case when covariance matrices are approximated by low-rank matrices. Note that the low-rank approximation is accurate only if the eigenvalues of covariance matrices follow a skewed distribution. To capture the skewed eigenvalue distribution, we introduce the concept of *effective numerical rank* (Hansen, 1987) that generalizes the rank of matrix:

Definition 1 *For a positive constant $\mu > 0$ and semi-positive definite matrix $M \in \mathbb{R}^{d\times d}$ of eigenvalues $\{\nu_i\}$, the effective numerical rank w.r.t. μ is defined to be $r(M,\mu) = \sum_{i=1}^{d} \nu_i/(\mu+\nu_i)$.*

It is evident that the effective numerical rank is upper bounded by the true rank, i.e., $r(M,\mu) \leq rank(M)$. To further see how the concept of effective numerical rank captures the skewed eigenvalue distribution, consider a PSD matrix M of full rank with $\sum_{i=k}^{d} \nu_i \leq \mu$ for small k. It is easy to verify that $r(M,\mu) \leq k$, i.e., M can be well approximated by a matrix of rank k.

Define the effective numerical rank for a set of matrices $\{M_t\}_{t=1}^T$ as $r\left(\{M_t\}_{t=1}^T,\mu\right) = \max_{1\leq t\leq T} r(M_t,\mu)$. Under the assumption that the effective numerical rank for the set of covariance matrices $\{S_t^{\pm}\}_{t=1}^T$ is small (i.e., $S_t^{\pm}$ can be well approximated by low-rank matrices), the following theorem gives the convergence rate for $|\sum_t \widehat{\mathcal{L}}_t(\mathbf{w}_t) - \sum_t \mathcal{L}_t(\mathbf{w}_*)|$, where $\widehat{\mathcal{L}}_t(\mathbf{w}_t)$ are given by Eqs. (6) and (7).

Theorem 3 *Let $r = r(\{S_t^{\pm}\}_{t=1}^T,\lambda)$ be the effective numerical rank for the sequence of covariance matrices $\{S_t^{\pm}\}_{t=1}^T$. For $0<\delta<1$, $0<\epsilon\leq 1/2$, $\|\mathbf{w}_*\| \leq B$, $\|\mathbf{x}_t\| \leq 1$ $(t\in[T])$ and $TL^* \geq \sum_{t=1}^T \mathcal{L}_t(\mathbf{w}_*)$, we have with probability at least $1-\delta$,*

$$\left|\sum_t \left(\widehat{\mathcal{L}}_t(\mathbf{w}_t) - \mathcal{L}_t(\mathbf{w}_*)\right)\right| \leq 2\epsilon TL^* + 2\kappa B^2 + B\sqrt{2\kappa TL^*}$$

provided $\tau \geq 32r\lambda(\log 2dT/\delta)/\epsilon^2$, where $\kappa = 4+\lambda$ and $\eta_t = 1/(\kappa+\sqrt{(\kappa^2+\kappa TL^*/B^2)})$.

Table 1. Benchmark datasets

datasets	#inst	#feat	datasets	#inst	#feat
diabetes	768	8	w8a	49,749	300
fourclass	862	2	kddcup04	50,000	65
german	1,000	24	mnist	60,000	780
splice	3,175	60	connect-4	67,557	126
usps	9,298	256	acoustic	78,823	50
letter	15,000	16	ijcnn1	141,691	22
magic04	19,020	10	epsilon	400,000	2,000
a9a	32,561	123	covtype	581,012	54

For the separable distribution $L^* = 0$, we also obtain an $O(1/T)$ convergence rate when the covariance matrices are approximated by low-rank matrices. Compared with Theorem 2, Theorem 3 introduces an additional term $2\epsilon L^*$ in the bound when using the approximate covariance matrices, and it is noteworthy that the approximation does not significantly increase the bound of Theorem 2 if $2\epsilon TL^* \leq B\sqrt{2(4+\lambda)TL^*}$, i.e., $\epsilon \leq B\sqrt{2(\lambda+4)/TL^*}$. This implies that the approximate algorithm will achieve similar performance as the one using the full covariance matrices provided $\tau = \Omega(r\lambda T(\log d + \log T)/(\lambda+4))$. When $\lambda = O(1/T)$, this requirement is reduced to $\tau = \Omega(r[\log d + \log T])$, a logarithmic dependence on dimension d.

5. Experiments

We evaluate the performance of OPAUC on benchmark datasets and high-dimensional datasets in Sections 5.1 and 5.2, respectively. Then, we study the parameter influence in Section 5.3.

5.1. Comparison on Benchmark Data

We conduct our experiments on sixteen benchmark datasets[1,2,3] as summarized in Table 1. Some datasets have been used in previous studies on AUC optimization, whereas the other are large ones requiring one-pass procedure. The features have been scaled to $[-1,1]$ for all datasets. Multi-class datasets have been transformed into binary ones by randomly partitioning classes into two groups, where each group contains the same number of classes.

In addition to state-of-the-art online AUC approaches $\mathbf{OAM_{seq}}$ and $\mathbf{OAM_{gra}}$ (Zhao et al., 2011), we also compare with:

- **online Uni-Exp**: An online learning algorithm which optimizes the (weighted) univariate exponential loss (Kotlowski et al., 2011);

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

Table 2. Comparison of the testing AUC values (mean±std.) on benchmark datasets. •/◦ indicates that OPAUC is significantly better/worse than the corresponding method (pairwise t-tests at 95% significance level).

datasets	OPAUC	OAM_{seq}	OAM_{gra}	online Uni-Exp	batch Uni-Log	batch SVM-OR	batch LS-SVM
diabetes	.8309±.0350	.8264±.0367	.8262± .0338	.8215±.0309•	.8330±.0322	.8326±.0328	.8325±.0329
fourclass	.8310±.0251	.8306±.0247	.8295±.0251	.8281±.0305	.8288±.0307	.8305±.0311	.8309±.0309
german	.7978±.0347	.7747±.0411•	.7723±.0358•	.7908±.0367	.7995±.0344	.7935±.0348	.7994±.0343
splice	.9232±.0099	.8594±.0194•	.8864±.0166•	.8931±.0213•	.9208±.0107•	.9239±.0089	.9245±.0092◦
usps	.9620±.0040	.9310±.0159•	.9348±.0122•	.9538±.0045•	.9637±.0041◦	.9630±.0047◦	.9634±.0045◦
letter	.8114±.0065	.7549±.0344•	.7603±.0346•	.8113±.0074	.8121±.0061	.8144±.0064◦	.8124±.0065◦
magic04	.8383±.0077	.8238±.0146•	.8259±.0169•	.8354±.0099•	.8378±.0073	.8426±.0074◦	.8379±0.0078
a9a	.9002±.0047	.8420±.0174•	.8571±.0173•	.9005±.0024	.9033±.0025◦	.9009±.0036	.8982±.0028•
w8a	.9633±.0035	.9304±.0074•	.9418±.0070•	.7693±.0986•	.9421±.0062•	.9495±.0082•	.9495±.0092•
kddcup04	.7912±.0039	.6918±.0412•	.7097±.0420•	.7851±.0050•	.7900±.0039•	.7903±.0039•	.7898±.0039•
mnist	.9242±.0021	.8615±.0087•	.8643±.0112•	.7932±.0245•	.9334±.0021◦	.9340±.0020◦	.9336±.0025◦
connect-4	.8760±.0023	.7807±.0258•	.8128±.0230•	.8702±.0025•	.8784±.0026◦	.8749±.0025•	.8739±.0026•
acoustic	.8192±.0032	.7113±.0590•	.7711±.0217•	.8171±.0034•	.8253±.0032◦	.8262±.0032◦	.8210±.0033◦
ijcnn1	.9269±.0021	.9209±.0079•	.9100±.0092•	.9264±.0035	.9282±.0023◦	.9337±.0024◦	.9320±.0037◦
epsilon	.9550±.0007	.8816±.0042•	.8659±.0176•	.9488±.0012•	.8647±.0150•	.8643±.0053•	.8644±.0050•
covtype	.8244±.0014	.7361±.0317•	.7403±.0289•	.8236±.0017	.8246±.0010	.8248±.0013	.8222±.0014•
win/tie/loss		**14/2/0**	**14/2/0**	**10/6/0**	**4/6/6**	**4/6/6**	**6/4/6**

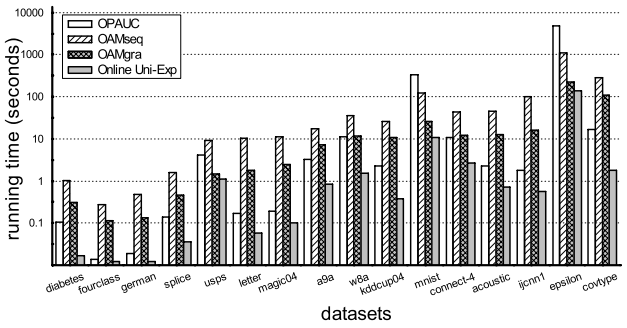

Figure 1. Comparison of the running time (in seconds) of OPAUC and online learning algorithms on benchmark data sets. Notice that the y-axis is in log-scale.

- **batch Uni-Log**: A batch learning algorithm which optimizes the (weighted) univariate logistic loss (Kotlowski et al., 2011);
- **batch SVM-OR**: A batch learning algorithm which optimizes the pairwise hinge loss (Joachims, 2006);
- **batch LS-SVM**: A batch learning algorithm which optimizes the pairwise square loss.

All experiments are performed with Matlab 7 on a node of computational cluster with 16 CPUs (Intel Xeon Due Core 3.0GHz) running RedHat Linux Enterprise 5 with 48GB main memory. For batch algorithms, due to memory limit, 8,000 training examples are randomly chosen if training data size exceeds 8,000, whereas only 2,000 training examples are used for the epsilon dataset because of its high dimension.

Table 3. High-dimensional datasets

datasets	#inst	#feat	datasets	#inst	#feat
sector	9,619	55,197	news20.binary	19,996	1,355,191
sector.lvr	9,619	55,197	rcv1v2	23,149	47,236
news20	15,935	62,061	ecml2012	456,886	98,519

Five-fold cross-validation is executed on training sets to decide the learning rate $\eta_t \in 2^{[-12:10]}$ for online algorithms, the regularized parameter $\lambda \in 2^{[-10:2]}$ for OPAUC and $\lambda \in 2^{[-10:10]}$ for batch algorithms. For OAM_{seq} and OAM_{gra}, the buffer sizes are fixed to be 100 as recommended in (Zhao et al., 2011). For univariate approaches, the class ratios are chosen as done in (Kotlowski et al., 2011).

The performances of the compared methods are evaluated by five trials of 5-fold cross validation, where the AUC values are obtained by averaging over these 25 runs, as summarized in Table 2. It is evident that OPAUC is better than the other three online algorithms OAM_{seq}, OAM_{gra} and online Uni-Exp, particularly for large datasets. The win/tie/loss counts show that OPAUC is clearly superior to these online algorithms, as it wins for most times and never loses. It is also observable that OPAUC is highly competitive to the three batch learning algorithms; this is impressive because these batch algorithms require storing the whole training dataset whereas OPAUC does not store training data. Additionally, batch LS-SVM which optimizes the square loss is comparable to the other batch algorithms, verifying our argument that square loss is effective for AUC optimization. We have also compared with SVM-perf (Joachims, 2005), on-

Table 4. Comparison of the testing AUC values (mean±std.) on high-dimensional datasets. •/◦ indicates that OPAUCr is significantly better/worse than the corresponding method (pairwise t-tests at 95% significance level). 'N/A' means that no results were obtained after running out 10^6 seconds (about 11.6 days).

datasets	OPAUCr	OAM_{seq}	OAM_{gra}	online Uni-Exp	OPAUC^{f}	OPAUC^{rp}	$\text{OPAUC}^{\text{pca}}$
sector	.9292±.0081	.9163±.0087•	.9043±.0100•	.9215±.0034•	.6228±.0145•	.7286±.0619•	.8853±.0114•
sector.lvr	.9962±.0011	.9965±.0064	.9955±.0059	.9969±.0093	.6813±.0444•	.9863±.0258•	.9893±.0288•
news20	.8871±.0083	.8543±.0099•	.8346±.0094•	.8880±.0047	.5958±.0118•	.7885±.0079•	.8878±.0115
news20.binary	.6389±.0136	.6314±.0131•	.6351±.0135•	.6347±.0092•	.5068±.0086•	.6212±.0072•	N/A
rcv1v2	.9686±.0029	.9686±.0026	.9604±.0025•	.9822±.0042◦	.6875±.0101•	.9353±.0053•	.9752±.0020◦
ecml2012	.9828±.0008	N/A	.9657±.0055•	.9820±.0016•	.6601±.0036•	.9355±.0047•	N/A

line and batch univariate square loss, and our results show that OPAUC is is significantly better than online and batch univariate square loss, and highly competitive to SVM-perf. Due to page limit, we present these results in a longer version (Gao et al., 2013).

We also compare the running time of OPAUC and the online algorithms OAM_{seq}, OAM_{gra} and online Uni-Exp, and the average CPU time (in seconds) are shown in Figure 1. As expected, online Uni-Exp takes the least time cost because it optimizes on single-instance (univariate) loss, whereas the other algorithms work by optimizing pairwise loss. On most datasets, the running time of OPAUC is competitive to OAM_{seq} and OAM_{gra}, except on the mnist and epsilon datasets which have the highest dimension in Table 1.

5.2. Comparison on High-Dimensional Data

Next, we study the performance of using low-rank matrices to approximate the full covariance matrices, denoted by OPAUCr. Six datasets[4,5] with nearly or more than 50,000 features are used, as summarized in Table 3. The news20.binary dataset contains two classes, different from news20 dataset. The original news20 and sector are multi-class datesets; in our experiments, we randomly group the multiple classes into two meta-classes each containing the same number of classes, and we also use the sector.lvr dataset which regards the largest class as positive whereas the union of other classes as negative. The original ecml2012 and rcv1v2 are multi-label datasets; in our experiments, we only consider the label with the largest population, and remove the features in ecml2012 dataset that take zero values for all instances.

Besides the online algorithms OAM_{seq}, OAM_{gra} and online Uni-Exp, we also evaluate three variants of OPAUC to study the effectiveness of approximating full covariance matrices with low-rank matrices:

- **OPAUC^{f}**: Randomly selects $1,000$-dim features and then works with full covariance matrices;
- **OPAUC^{rp}**: Projects into a $1,000$-dim feature space by Random Projection, and then works with full covariance matrices;
- **$\text{OPAUC}^{\text{pca}}$**: Projects into a $1,000$-dim feature space obtained by Principle Component Analysis, and then works with full covariance matrices.

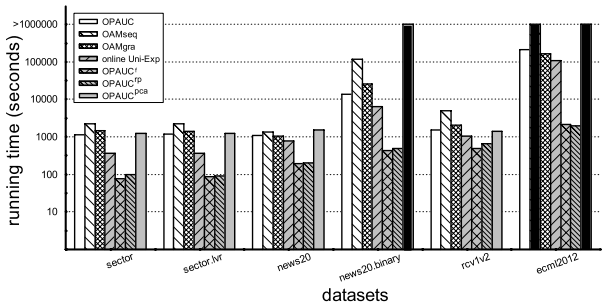

Figure 2. Comparison of the running time on high-dimensional datasets. Full black columns imply that no result was obtained after running out 10^6 seconds.

Similar to Section 5.1, five-fold cross validation is executed on training sets to decide the learning rate $\eta_t \in 2^{[-12:10]}$ and the regularization parameter $\lambda \in 2^{[-10:2]}$. Due to memory and computational limit, the buffer sizes are set to 50 for OAM_{seq} and OAM_{gra}, and the rank τ of OPAUCr is also set to 50. The performances of the compared methods are evaluated by five trials of 5-fold cross validation, where the AUC values are obtained by averaging over these 25 runs.

The comparison results are summarized in Table 4 and the average running time is shown in Figure 2. These results clearly show that our approximate OPAUCr approach is superior to the other compared methods. Compared with OAM_{seq} and OAM_{gra}, the running time costs are comparable whereas the performance of OPAUCr is better. Online Uni-Exp is more efficient than OPAUCr because it optimizes univariate loss, but the performance of OPAUCr is highly competitive or better, except on rcv1v2, the only dataset

[4] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/
[5] http://www.ecmlpkdd2012.net/discovery-challenge

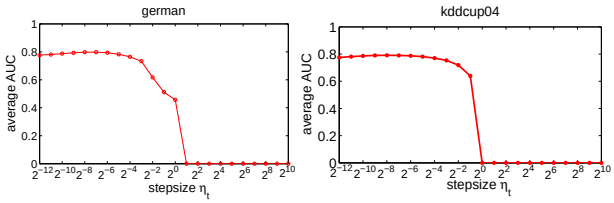

Figure 3. Influence of stepsize η_t

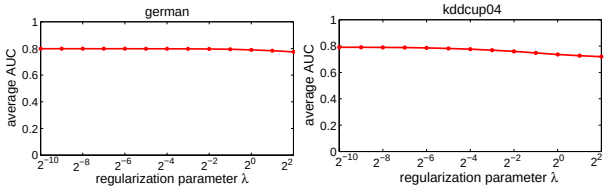

Figure 4. Influence of regularization parameter λ

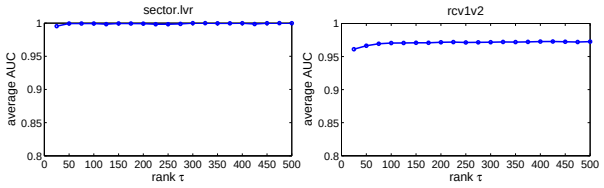

Figure 5. Influence of rank τ

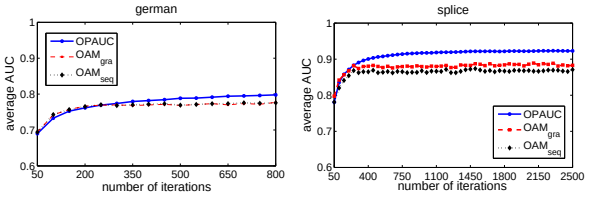

Figure 6. Influence of iterations

with less than 50,000 features. Compared with the three variants, OPAUC$^{\text{f}}$ and OPAUC$^{\text{rp}}$ are more efficient, but with much worse performances. OPAUC$^{\text{pca}}$ achieves a better performance on rcv1v2, but it is worse on datasets with more features; particularly, on the two datasets with the largest number of features, OPAUC$^{\text{pca}}$ cannot return results even after running out 10^6 seconds (almost 11.6 days). Our approximate OPAUCr approach is significantly better than all the other methods (if they return results) on the two datasets with the largest number of features: news.binary with more than 1 million features, and ecml2012 with nearby 100 thousands features. These observations validate the effectiveness of the low-rank approximation used by OPAUCr for handling high-dimensional data.

5.3. Parameter Influence

We study the influence of parameters in this section. Figure 3 shows that stepsize η_t should not be set to values bigger than 1, whereas there is a relatively big range between $[2^{-12}, 2^{-4}]$ where OPAUC achieves good results. Figures 4 shows that OPAUC is not sensitive to the value of regularization parameter λ given that it is not set with a big value. Figure 5 shows that OPAUCr is not sensitive to the values of rank τ, and it works well even when $\tau = 50$; this verifies Theorem 3 that a relatively small τ value suffices to lead to a good approximation performance. Figure 6 compares studies the influence of the iterations for OPAUC, OAM$_{\text{seq}}$ and OAM$_{\text{gra}}$, and it is observable that OPAUC convergence faster than the other two algorithms, which verifies our theoretical argument in Section 4.

Due to page limit, we only present the results of two datasets for the study of each parameter, but the trends are similar on other datasets, and more results can be found in our longer version (Gao et al., 2013).

6. Conclusion

In this paper, we study one-pass AUC optimization that requires going through the training data only once, without storing the entire dataset. Here, a big challenge lies in the fact that AUC is measured by a sum of losses defined over pairs of instances from different classes. We propose the OPAUC approach, which employs a square loss and requires the storing of only the first and second-statistics for the received training examples. A nice property of OPAUC is that its storage requirement is $\mathrm{O}(d^2)$, where d is the dimension of data, independent from the number of training examples. To handle high-dimensional data, we develop an approximate strategy by using low-rank matrices. The effectiveness of our proposed approach is verified both theoretically and empirically. In particular, the performance of OPAUC is significantly better than state-of-the-art online AUC optimization approaches, even highly competitive to batch learning approaches; the approximate OPAUC is significantly better than all compared methods on large datasets with one hundred thousands or even more than one million features. An interesting future issue is to develop one-pass AUC optimization approaches not only with a performance comparable to batch approaches, but also with an efficiency comparable to univariate loss optimization approaches.

Acknowledgements: The authors want to thank the reviewers for helpful comments and suggestions. This research was partially supported by the NSFC (61073097, 61021062), 973 Program (2010CB327903), and ONR Award (N000141210431, N00014-09-1-0663).

References

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

Cortes, C. and Mohri, M. AUC optimization vs. error rate minimization. In *Advances in Neural Information Processing Systems 16*, pp. 313–320. MIT Press, Cambridge, MA, 2004.

Flach, P. A., Hernández-Orallo, J., and Ramirez, C. F. A coherent interpretation of AUC as a measure of aggregated classification performance. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 657–664, Bellevue, WA, 2011.

Gao, W. and Zhou, Z.-H. On the consistency of AUC optimization. *CoRR/abstract*, 1208.0645, 2012.

Gao, W., Jin, R., Zhu, S., and Zhou, Z.-H. One-pass AUC optimization. *CoRR/abstract*, 1305.1363, 2013.

Hanley, J. A. and McNeil, B. J. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148 (3):839–843, 1983.

Hansen, P. C. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*. SIAM, 1987.

Herschtal, A. and Raskutti, B. Optimising area under the ROC curve using gradient descent. In *Proceedings of the 21st International Conference on Machine Learning*, Alberta, Canada, 2004.

Hsu, D., Kakade, S., and Zhang, T. Random design analysis of ridge regression. In *Proceedings of the 25th Annual Conference on Learning Theory*, pp. 9.1–9.24, Edinburgh, Scotland, 2012.

Joachims, T. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 377–384, Bonn, Germany, 2005.

Joachims, T. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 217–226, Philadelphia, PA, 2006.

Johnstone, I. High dimensional statistical inference and random matrices. In *Proceedings of the International Congress of Mathematicians*, pp. 307–333, Madrid, Spain, 2006.

Kotlowski, W., Dembczynski, K., and Hüllermeier, E. Bipartite ranking through minimization of univariate loss. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 1113–1120, Bellevue, WA, 2011.

Liu, X.-Y., Wu, J., and Zhou, Z.-H. Exploratory undersampling for class-imbalance learning. *IEEE Trans. Systems, Man, and Cybernetics - B*, 39(2): 539–550, 2009.

Metz, C. E. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283–298, 1978.

Nocedal, J. and Wright, S. J. *Numerical optimization*. Springer, 1999.

Provost, F. J., Fawcett, T., and Kohavi, R. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 445–453, Madison, WI, 1998.

Rakhlin, A., Shamir, O., and Sridharan, K. Making gradient descent optimal for strongly convex stochastic optimization. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 449–456, Edinburgh, Scotland, 2012.

Rudin, C. and Schapire, R. E. Margin-based ranking and an equivalence between AdaBoost and RankBoost. *Journal of Machine Learning Research*, 10: 2193–2232, 2009.

Srebro, N., Sridharan, K., and Tewari, A. Smoothness, low noise and fast rates. In *Advances in Neural Information Processing Systems 24*, pp. 2199–2207. MIT Press, Cambridge, MA, 2010.

Zhao, P., Hoi, S., Jin, R., and Yang, T. Online AUC maximization. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 233–240, Bellevue, WA, 2011.