Online AUC Optimization for High-dimensional Sparse Datasets

***Abstract*—The Area Under the ROC Curve (AUC) is a widely used performance measure for imbalanced classification arising from many application domains where high-dimensional sparse data is abundant. In such cases, each d dimensional sample has only k non-zero features with $k \ll d$, and data arrives sequentially in a streaming form. Current online AUC optimization algorithms have high per-iteration cost $\mathcal{O}(d)$ and usually produce non-sparse solutions in general, and hence are not suitable for handling the data challenge mentioned above.**

In this paper, we aim to directly optimize the AUC score for high-dimensional sparse datasets under online learning setting and propose a new algorithm, FTRL-AUC. Our proposed algorithm can process data in an online fashion with a much cheaper per-iteration cost $\mathcal{O}(k)$, making it amenable for high-dimensional sparse streaming data analysis. Our new algorithmic design critically depends on a novel reformulation of the U-statistics AUC objective function as the empirical saddle point reformulation, and the innovative introduction of the "lazy update" rule so that the per-iteration complexity is dramatically reduced from $\mathcal{O}(d)$ to $\mathcal{O}(k)$. Furthermore, FTRL-AUC can inherently capture sparsity more effectively by applying a generalized Follow-The-Regularized-Leader (FTRL) framework.

Experiments on real-world datasets demonstrate that FTRL-AUC significantly improves both run time and model sparsity while achieving competitive AUC scores compared with the state-of-the-art methods. Comparison with the online learning method for logistic loss demonstrates that FTRL-AUC achieves higher AUC scores especially when datasets are imbalanced.

***Index Terms*—online learning, Follow-The-Regularized-Leader, sparsity, AUC optimization**

I. Introduction

The Area Under the ROC Curve (AUC) score [1, 2, 3] is a widely used performance metric to measure the quality of classifiers, particularly in the problem of imbalanced classification where the size of one class is much larger than the other class. In such problems as online spam filtering [4], ad click prediction [5], and identifying malicious URLs [6], the datasets are not only imbalanced but also *high-dimensional* and *sparse*. Specifically, such datasets are of very high dimension d, but the number of nonzero features k in each training sample is far less than the total number of features d, i.e., $k \ll d$. We consider this type of online data, which arrives in a streaming fashion, requiring real-time training and predictions. Hence, it is of critical importance to develop efficient online AUC optimization algorithms which can make prediction in a real time manner upon receiving new high dimensional sparse data.

Online machine learning (*online learning*) [7, 8, 9] is a natural choice to deal with data in a real time manner as it can update the model sequentially. Most of the existing online learning algorithms [10, 11, 7, 8] focus on the error rate (accuracy) where the objective function is the sum of pointwise losses over individual examples. Thus, they are not suitable for the problem of AUC maximization because the AUC objective function is the sum of pairwise losses over pairs of examples in the form of U-statistics [12]. Recently, considerable work has been done to develop variants of stochastic (online) gradient descent algorithms for AUC maximization. Specifically, the work of [13, 14, 15, 16, 17, 18, 19] proposes a variant of stochastic gradient descent (SGD) algorithms and the particular work [17, 18, 19] uses stochastic (online) proximal gradient algorithms to handle the sparse ℓ^1-regularization.

However, such online AUC optimization algorithms do not explore the structure of high-dimensional sparse data, and the per-iteration cost of at least $O(d)$ is expensive when d is very large. Moreover, the produced AUC maximization models [17, 18, 19] updated by using ℓ^1 regularization (constraints) do not produce sparse solutions. As such, the existing online algorithms cannot apply to high-dimensional sparse data where the response prediction time is critically important [20, 5].

Inspired by a generalized Follow-The-Regularized-Leader (FTRL) framework [21, 22, 5], in this paper, we propose an online AUC optimization algorithm, namely FTRL-AUC, for high-dimensional sparse datasets. Our new algorithm FTRL-AUC has three novel improvements:

- Our proposed algorithm FTRL-AUC can handle the streaming data in an online manner, i.e. updating the model parameter upon receiving each individual data point without the need of pairing it with previous ones. Motivated by [16, 19], we achieve this by reformulating the original objective function of AUC maximization as an empirical saddle point formulation.
- The per-iteration cost of FTRL-AUC, making full use of inherent sparsity of datasets, is $\mathcal{O}(k)$, which is much less than $\mathcal{O}(d)$ of the existing methods. It is challenging to directly obtain $\mathcal{O}(k)$ complexity because the gradient of the original loss at each iteration is not sparse. To overcome this obstacle, our key idea is to introduce a new surrogate loss and a novel "lazy update" rule to update current positive score and negative score, and then apply the generalized FTRL framework.
- Finally, our experimental results demonstrate that FTRL-AUC significantly improves both run time and model sparsity compared to the state-of-the-art methods. We also compare our algorithm with other online learning algorithms for logistic loss and demonstrate that it achieves higher AUC scores especially when datasets are imbalanced.

The rest of the paper is organized as follows. We first discuss

the related work in Section II. The problem formulation of online AUC optimization is given in Section III. In Section IV, we present the new algorithm FTRL-AUC with time complexity and regret bound analysis. We evaluate our method in Section V. Section VI concludes the paper. For the reproducibility purpose, source code of FTRL-AUC including all baseline methods and datasets can be accessed at: https://anonymous.4open.science/r/25f48408-b148-4651-adcf-82dcee2c9c75/ and will be made available on publish.

II. Related Work

Online AUC optimization. AUC optimization algorithms have been developed under batch learning setting [23, 24, 25] where the predictor is generated based on the entire training samples. The reason why designing online AUC maximization methods is challenging is twofold: 1) in contrast to accuracy-based classification approaches where the loss is based on one individual example, the loss function of AUC optimization involves a pair of examples; 2) in practice, individual examples are arriving sequentially rather than pairs of examples.

The first work of online AUC optimization has been proposed in [15, 14, 13]. The key idea is to use the gradient of a local empirical error which compares the current training example with all previous ones. However, these methods need to save all (or part of) previous training samples and need to compare the current example with previous ones, which leads to high per-iteration $\mathcal{O}(td)$ at time t. The appealing work by Gao et. al [26] follows the same spirit but observes, in the case of the least square loss, that updates of such algorithms only rely on covariance matrix. This leads to a per-iteration cost $\mathcal{O}(d^2)$. Similarly, Ding et. al [27] use the same formulation but with an adaptive learning rate with similar per-iteration cost.

More recent studies [16, 18, 19] develop online AUC optimization with per-iteration cost $\mathcal{O}(d)$ with competitive convergence results. The key idea is to introduce a new equivalent saddle-point formulation of AUC optimization. In particular, it is shown that maximizing AUC score is equivalent to solving a min-max problem, and hence stochastic primal-dual gradient-based algorithms can be applied. The work [18, 19] develops fast algorithms for online AUC maximization with sparse regularization. However, all the methods do not take into account the inherent sparsity of data and the per-iteration cost is of $\mathcal{O}(d)$ which is still expensive if d is very large. Moreover, such sparse online AUC optimization algorithms are essentially variants of stochastic proximal gradient algorithms which, as shown in [11, 20], do not produce desired sparse solutions.

Online learning algorithms with sparsity. A natural approach to obtain sparsity under batch learning setting is to add the ℓ^1 regularization. However, this is not the case for online setting where only one training sample is available at each time. Indeed, this is shown in [20] that simply applying ℓ^1 regularization fails to work since the gradient of each sample does not induce sparsity. Many important works [28, 20, 11, 10, 29] have been proposed to capture problem sparsity. For example, the regularized dual-averaging method proposed in [11] captures sparsity more effectively. The idea of online dual averaging is based on [30] as all gradients used are equally important.

As demonstrated in these works [21, 22, 5], the regularized dual averaging can be regarded as a special case of the follow-the-regularized-leader. This type of algorithms is originally proposed in [31] which is based on the follow-the-leader [32]. It has been shown that both Regularized Dual Averaging (RDA) [11] and Follow The Proximally Regularized Leader (FTPRL) [21] can capture the sparsity effectively. However, the FTRL and regularized dual averaging methods are developed for accuracy-based loss depending on individual examples, while the objective function of AUC maximization is of U-Statistics based on pairs of examples. It remains unclear that how we can incorporate the FTRL framework with the AUC setting to design efficient online AUC optimization methods exploring inherent sparsity of high-dimensional sparse data.

We aim to design online AUC maximization algorithms which not only achieve much cheaper $O(k)$ per-iteration cost but also generate much sparser solutions than existing methods. By this, we mean the following two requirements. The first one is that the gradient of loss of each training sample should be sparse. The other one is that the loss needs to be convex with respect to current model variable in order to guarantee a sublinear or logarithmic regret bound. We will illustrate our new developments in the subsections to achieve this goal.

III. Problem Formulation

In this section, we give the definition of AUC score and then define our problem. To this end, let us introduce some notations.

The training sample at time t is denoted as $\boldsymbol{x}_t \in \mathbb{R}^d$ and the corresponding label is $y_t \in \{+1, -1\}$. The model of a specific algorithm at time t is denoted as $\boldsymbol{w}_t \in \mathbb{R}^d$. Define $\boldsymbol{x}_t^+ := \boldsymbol{x}_t$ if $y_t = 1$ and $\boldsymbol{x}_t^- := \boldsymbol{x}_t$ if $y_t = -1$. The i-th entries of $\boldsymbol{x}_t$ is denoted by $x_{t,i}$. The indicator function is defined as $\mathbb{1}_{[A]}$ where it takes value 1 if A is true and 0 otherwise. We assume each training sample is k-sparse, i.e. $\|\boldsymbol{x}_t\|_0 = \mathcal{O}(k)$ and $k \ll d$.[1] Matrices are denoted by bold capitals such as $\boldsymbol{Q}$. The average of all t positive and t negative data samples are denoted respectively by $\overline{\boldsymbol{x}}_t^+ = \frac{\sum_{i=1}^t \boldsymbol{x}_i^+}{t}$ and $\overline{\boldsymbol{x}}_t^- = \frac{\sum_{j=1}^t \boldsymbol{x}_j^-}{t}$.

A. Definition of AUC Optimization

Given a set of training samples $\mathcal{D} := \{\boldsymbol{x}_i, y_i\}_{i=1}^T$ where $T = T_+ + T_-$, T_+ is the number of positive samples and T_- is the number of negative samples in $\mathcal{D}$, the AUC score [3] of a specific linear classifier $\boldsymbol{w}$ is defined as

$$\mathrm{AUC}(\boldsymbol{w}) := \frac{1}{T_+ T_-} \sum_{i=1}^{T} \sum_{j=1}^{T} \mathbb{1}_{[\boldsymbol{w}^\top \boldsymbol{x}_i \geq \boldsymbol{w}^\top \boldsymbol{x}_j]} \mathbb{1}_{[y_i = 1]} \mathbb{1}_{[y_j = -1]}. \quad (1)$$

Intuitively, $\mathrm{AUC}(\boldsymbol{w})$ defined in (1) is equivalent to the probability of a positive sample ranked by $\boldsymbol{w}$ higher than negative samples. As illustrated on the left of Figure 1, the diagonal line is the linear classifier $\boldsymbol{w} = [-1, 1]^\top$, which scores

[1] $\|\boldsymbol{x}\|_0 := |\{x_i : x_i \neq 0\}|$, the number of non-zeros in $\boldsymbol{x}$. $k \ll d$ is not a hard condition. As long as $k < d$ on average over all training samples, our proposed method can benefit from it accordingly.

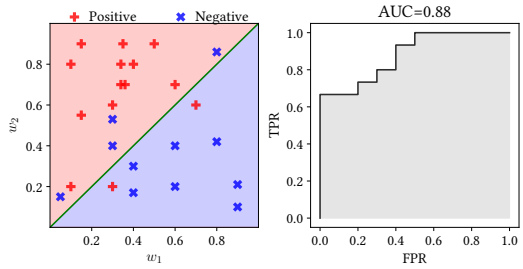

Fig. 1: A toy example of calculating AUC score

all points in red region positive and all points in blue region negative. By using the true labels of these data samples and these corresponding scores, we can draw the Area Under the ROC curve accordingly (as shown on the right of Figure 1), where FPR and TPR are the false positive rate and true positive rate respectively.

To simplify (1), denote positive samples as $\boldsymbol{x}_1^+, \boldsymbol{x}_2^+, \ldots, \boldsymbol{x}_{T_+}^+$ and negative samples as $\boldsymbol{x}_1^-, \boldsymbol{x}_2^-, \ldots, \boldsymbol{x}_{T_-}^-$ respectively. To maximize (1), it is equivalent to minimize $1 - AUC(\boldsymbol{w})$, and hence we have the following minimization problem

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{T_+T_-} \sum_{i=1}^{T_+}\sum_{j=1}^{T_-} \mathbb{1}_{[\boldsymbol{w}^\top \boldsymbol{x}_i^+ < \boldsymbol{w}^\top \boldsymbol{x}_j^-]}. \tag{2}$$

The inherently combinatorial of indicator function $\mathbb{1}_{[A]}$ makes it difficult to optimize (2) directly. Thus, we replace the indicator function by a convex loss such as least square [26] or hinge loss [15]. We choose the least square loss[2] as our surrogate as the following

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} F(\boldsymbol{w}) := \frac{1}{T_+T_-} \sum_{i=1}^{T_+}\sum_{j=1}^{T_-} \left(1 - \boldsymbol{w}^\top \left(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-\right)\right)^2, \tag{3}$$

where the objective function $F(\boldsymbol{w})$ is in the form of U-Statistics [9] depending on pairs of individual examples.

B. Online AUC optimization

The standard online learning algorithm is as follows: at each iteration t, the learner receives a question $\boldsymbol{x}_t$ and then the learner makes a prediction and a corresponding loss could occur after the true label y_t is available, i.e. $f_t(\boldsymbol{w}_t; \{\boldsymbol{x}_t, y_t\})$. For example, a least square loss can be written as $f_t(\boldsymbol{w}_t; \{\boldsymbol{x}_t, y_t\}) := \left(\boldsymbol{w}_t^\top \boldsymbol{x}_t - y_t\right)^2$. The goal of the online learning is to design an algorithm which aims to minimize the regret as defined in the following

$$\text{Regret}_T := \sum_{t=1}^{T} f_t(\boldsymbol{w}_t; \{\boldsymbol{x}_t, y_t\}) - \min_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{t=1}^{T} f_t(\boldsymbol{w}; \{\boldsymbol{x}_t, y_t\}) \tag{4}$$

The main difficulty of applying online learning algorithms is that $F(\boldsymbol{w})$ cannot be seamlessly decomposed into T separable loss functions. Some theoretical results show that when the loss is pairwise loss functions [13, 14], it is still possible to do online learning, but existing strategies need to save previous training samples, which could be memory costly and the per-iteration cost is very high.

[2]It has been proved in [26, 33], least square is consistent with AUC but hinge loss not.

To design an efficient online AUC optimization algorithm for high-dimensional sparse data with per iteration cost $\mathcal{O}(k)$, we reformulate the problem (3) as an empirical saddle point (min-max) formulation which is inspired by [34]. The proof in [34] uses the concepts of conditional expectations and we provide a much simpler and straightforward proof.

Lemma 1 (Empirical Saddle Point Reformulation)**.** *Minimization problem (3) is equivalent to the following saddle point problem*

$$\min_{\substack{\boldsymbol{w}\in\mathbb{R}^d,\\ (a,b)\in\mathbb{R}^2}} \max_{\alpha\in\mathbb{R}} \left\{ \sum_{t=1}^{T} f_t(\boldsymbol{w}, a, b, \alpha; \{\boldsymbol{x}_t, y_t\}) \right\}, \tag{5}$$

where each f_t is defined as the following

$$\begin{aligned} f_t(\boldsymbol{w}, a, b, \alpha; \{\boldsymbol{x}_t, y_t\}) &= (1-p_T)\left(\boldsymbol{w}^\top \boldsymbol{x}_t - a\right)^2 \mathbb{1}_{[y_t=1]} \\ &+ p_T\left(\boldsymbol{w}^\top \boldsymbol{x}_t - b\right)^2 \mathbb{1}_{[y_t=-1]} - p_T(1-p_T)\alpha^2 \\ &+ 2(1+\alpha)\boldsymbol{w}^\top \boldsymbol{x}_t \Big(p_T \mathbb{1}_{[y_t=-1]} - (1-p_T)\mathbb{1}_{[y_t=1]}\Big), \end{aligned} \tag{6}$$

where $p_T = \sum_{i=1}^{T} \mathbb{1}_{[y_i=1]}/T$ estimates the distribution of positive training samples and $a, b, \alpha \in \mathbb{R}$.

Proof. The original minimization objective is the following

$$\begin{aligned} F(\boldsymbol{w}) &= \frac{1}{T_+T_-} \sum_{i=1}^{T_+}\sum_{j=1}^{T_-} \left(1 - \boldsymbol{w}^\top \left(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-\right)\right)^2 \\ &= 1 - \frac{2}{T_+}\sum_{i=1}^{T_+} \boldsymbol{w}^\top \boldsymbol{x}_i^+ + \frac{2}{T_-}\sum_{j=1}^{T_-} \boldsymbol{w}^\top \boldsymbol{x}_j^- + \frac{1}{T_+}\sum_{i=1}^{T_+}\left(\boldsymbol{w}^\top \boldsymbol{x}_i^+\right)^2 \\ &\quad + \frac{1}{T_-}\sum_{j=1}^{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_j^-\right)^2 - \frac{2}{T_+T_-}\sum_{i=1}^{T_+}\sum_{j=1}^{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_i^+\right)\left(\boldsymbol{w}^\top \boldsymbol{x}_j^-\right) \\ &= 1 - \frac{2}{T_+}\sum_{i=1}^{T_+} \boldsymbol{w}^\top \boldsymbol{x}_i^+ + \frac{2}{T_-}\sum_{j=1}^{T_-} \boldsymbol{w}^\top \boldsymbol{x}_j^- \\ &\quad + \underbrace{\left\{\frac{1}{T_+}\sum_{i=1}^{T_+}\left(\boldsymbol{w}^\top \boldsymbol{x}_i^+\right)^2 - \left(\frac{1}{T_+}\sum_{i=1}^{T_+}\boldsymbol{w}^\top \boldsymbol{x}_i^+\right)^2\right\}}_{A} \\ &\quad + \underbrace{\left\{\frac{1}{T_-}\sum_{j=1}^{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_j^-\right)^2 - \left(\frac{1}{T_-}\sum_{j=1}^{T_-}\boldsymbol{w}^\top \boldsymbol{x}_j^-\right)^2\right\}}_{B} \\ &\quad + \underbrace{\left(\frac{1}{T_+}\sum_{i=1}^{T_+}\boldsymbol{w}^\top \boldsymbol{x}_i^+ - \frac{1}{T_-}\sum_{j=1}^{T_-}\boldsymbol{w}^\top \boldsymbol{x}_j^-\right)^2}_{C}. \end{aligned}$$

For items $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$, we can reformulate them as

$$\boldsymbol{A} = \min_{a\in\mathbb{R}} \frac{1}{T_+}\sum_{i=1}^{T_+}\left(\boldsymbol{w}^\top \boldsymbol{x}_i^+ - a\right)^2, \boldsymbol{B} = \min_{b\in\mathbb{R}} \frac{1}{T_-}\sum_{j=1}^{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_j^- - b\right)^2,$$

$$\boldsymbol{C} = \max_{\alpha\in\mathbb{R}} \left\{2\alpha\left(\frac{1}{T_-}\sum_{j=1}^{T_-}\boldsymbol{w}^\top \boldsymbol{x}_j^- - \frac{1}{T_+}\sum_{i=1}^{T_+}\boldsymbol{w}^\top \boldsymbol{x}_i^+\right) - \alpha^2\right\}.$$

Hence, we can rewrite $F(\boldsymbol{w})$ as the following

$$\begin{aligned} F(\boldsymbol{w}) &= 1 - \frac{2}{T_+}\sum_{i=1}^{T_+} \boldsymbol{w}^\top \boldsymbol{x}_i^+ + \frac{2}{T_-}\sum_{j=1}^{T_-} \boldsymbol{w}^\top \boldsymbol{x}_j^- \\ &\quad + \min_{(a,b)\in\mathbb{R}^2}\left\{\frac{1}{T_+}\sum_{i=1}^{T_+}\left(\boldsymbol{w}^\top \boldsymbol{x}_i^+ - a\right)^2 + \frac{1}{T_-}\sum_{j=1}^{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_j^- - b\right)^2\right\} \\ &\quad + \max_{\alpha\in\mathbb{R}}\left\{2\alpha\left(\frac{1}{T_-}\sum_{j=1}^{T_-}\boldsymbol{w}^\top \boldsymbol{x}_j^- - \frac{1}{T_+}\sum_{i=1}^{T_+}\boldsymbol{w}^\top \boldsymbol{x}_i^+\right) - \alpha^2\right\}. \end{aligned}$$

Finally, the minimization of $F(\boldsymbol{w})$ can be expressed as the sum of separable loss functions depending on each single training sample pair as the following

$$\begin{aligned}
\min_{\boldsymbol{w}} F(\boldsymbol{w}) = & \min_{\substack{\boldsymbol{w}\in\mathbb{R}^d,\\(a,b)\in\mathbb{R}^2}} \max_{\alpha\in\mathbb{R}} \Bigg\{ 1 - \frac{2}{T_+}\sum_{i=1}^{T_+} \boldsymbol{w}^\top \boldsymbol{x}_i^+ + \frac{2}{T_-}\sum_{j=1}^{T_-} \boldsymbol{w}^\top \boldsymbol{x}_j^- \\
& + \frac{1}{T_+}\sum_{i=1}^{T_+} \left(\boldsymbol{w}^\top \boldsymbol{x}_i^+ - a\right)^2 + \frac{1}{T_-}\sum_{j=1}^{T_-} \left(\boldsymbol{w}^\top \boldsymbol{x}_j^- - b\right)^2 \\
& + 2\alpha \left(\frac{1}{T_-}\sum_{j=1}^{T_-} \boldsymbol{w}^\top \boldsymbol{x}_j^- - \frac{1}{T_+}\sum_{i=1}^{T_+} \boldsymbol{w}^\top \boldsymbol{x}_i^+ \right) - \alpha^2 \Bigg\} \\
= & \min_{\substack{\boldsymbol{w}\in\mathbb{R}^d,\\(a,b)\in\mathbb{R}^2}} \max_{\alpha\in\mathbb{R}} \sum_{t=1}^{T} \Bigg\{ \frac{1}{T} - \frac{2}{T_+}\boldsymbol{w}^\top \boldsymbol{x}_t \mathbb{1}_{[y_t=1]} + \frac{2}{T_-}\boldsymbol{w}^\top \boldsymbol{x}_t \mathbb{1}_{[y_t=-1]} \\
& + \frac{1}{T_+}\left(\boldsymbol{w}^\top \boldsymbol{x}_t - a\right)^2 \mathbb{1}_{[y_t=1]} + \frac{1}{T_-}\left(\boldsymbol{w}^\top \boldsymbol{x}_t - b\right)^2 \mathbb{1}_{[y_t=-1]} \\
& + 2\alpha \left(\frac{1}{T_-}\boldsymbol{w}^\top \boldsymbol{x}_t \mathbb{1}_{[y_t=-1]} - \frac{1}{T_+}\boldsymbol{w}^\top \boldsymbol{x}_t \mathbb{1}_{[y_t=1]} \right) - \frac{\alpha^2}{T} \Bigg\}
\end{aligned}$$

Normalize the right hand side above by $\frac{T_+T_-}{T}$ and notice that $p_T = T_+/T$ and $1 - p_T = T_-/T$, we finally arrive at the following equivalent optimization problem

$$\min_{\substack{\boldsymbol{w}\in\mathbb{R}^d,\\(a,b)\in\mathbb{R}^2}} \max_{\alpha\in\mathbb{R}} \sum_{t=1}^{T} \Big\{ f_t(\boldsymbol{w}, a, b, \alpha; x_t, y_t) \Big\}.$$

□

Lemma 1 decomposes the original AUC objective function into the sum of separable loss by introducing three auxiliary variables (a, b, α). However, the algorithm in [16] needs to update $\boldsymbol{w}, a$, and b by using gradient descent and to update α by using gradient ascent. More importantly, the $\boldsymbol{w}_t$ at iteration t needs to project back to a ℓ^2-norm ball, which requires per-iteration cost $\mathcal{O}(d)$. A recent work [19] shows that, it is not necessary to update a, b, and α and optimization problem (5) can be further reformulated as

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \Big\{ \sum_{t=1}^{T} f_t(\boldsymbol{w}, a(\boldsymbol{w}), b(\boldsymbol{w}), \alpha(\boldsymbol{w}); \{\boldsymbol{x}_t, y_t\}) \Big\}, \tag{7}$$

where $a(\boldsymbol{w}) = \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_+}^+$, $b(\boldsymbol{w}) = \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_-}^-$, and $\alpha(\boldsymbol{w}) = b(\boldsymbol{w}) - a(\boldsymbol{w})$. From now on, we denote $f_t(\boldsymbol{w}, a(\boldsymbol{w}), b(\boldsymbol{w}), \alpha(\boldsymbol{w}); \{\boldsymbol{x}_t, y_t\})$ as $f_t(\boldsymbol{w})$. The gradient of each $f_t(\boldsymbol{w})$ above is

$$\begin{aligned}
\nabla f_t(\boldsymbol{w}) = & 2(1-p_T)\left(\boldsymbol{w}^\top \boldsymbol{x}_t - \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_+}^+\right)\left(\boldsymbol{x}_t - \overline{\boldsymbol{x}}_{T_+}^+\right)\mathbb{1}_{[y=1]} \\
& + 2p_T(\boldsymbol{w}^\top \boldsymbol{x}_t - \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_-}^-)(\boldsymbol{x}_t - \overline{\boldsymbol{x}}_{T_-}^-)\mathbb{1}_{[y=-1]} \\
& - 2p_T(1-p_T)\left(\boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_-}^- - \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_+}^+\right)\left(\overline{\boldsymbol{x}}_{T_-}^- - \overline{\boldsymbol{x}}_{T_+}^+\right) \\
& + 2(1 + \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_-}^- - \boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_+}^+)(p_T \mathbb{1}_{[y_t=-1]} - (1-p_T)\mathbb{1}_{[y_t=1]})\boldsymbol{x}_t \\
& + 2\boldsymbol{w}^\top \boldsymbol{x}_t (p_T \mathbb{1}_{[y_t=-1]} - (1-p_T)\mathbb{1}_{[y_t=1]})(\overline{\boldsymbol{x}}_{T_-}^- - \overline{\boldsymbol{x}}_{T_+}^+).
\end{aligned}$$

However, there are two disadvantages to apply (7) to the online learning setting: 1) the estimators $p_T, \overline{\boldsymbol{x}}_{T_+}^+$ and $\overline{\boldsymbol{x}}_{T_-}^-$ need to be known, which is unrealistic in the online learning setting; 2) $f_t(\boldsymbol{w})$ is not convex as proved in [17] and the gradient of $f_t(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ is not sparse in general. The gradient is non-sparse because it needs to calculate both $\boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_+}^+$ and $\boldsymbol{w}^\top \overline{\boldsymbol{x}}_{T_-}^-$. These vectors are not sparse and need $\mathcal{O}(d)$ cost in general. In the subsequent section, we will introduce several novel techniques to resolve these issues.

IV. Proposed Algorithm: FTRL-AUC

A. Problem reformulation

The difficultly of getting a sparse gradient motivates us to find an alternative way. Here is the main idea to overcome this obstacle. At each iteration t, instead of using $f_t(\boldsymbol{w}_t)$, we design a new convex loss $\widehat{f}_t(\boldsymbol{w}_t)$ by replacing the term $a(\boldsymbol{w}_t) = \boldsymbol{w}_t^T \overline{\boldsymbol{x}}_{T_+}^+$ by $a_t(\boldsymbol{w}_{t-1}) = \boldsymbol{w}_{t-1}^T \overline{\boldsymbol{x}}_{t_+}^+$, $b(\boldsymbol{w}_t) = \boldsymbol{w}_t^T \overline{\boldsymbol{x}}_{T_-}^-$ by $b_t(\boldsymbol{w}_{t-1}) = \boldsymbol{w}_{t-1}^T \overline{\boldsymbol{x}}_{t_-}^-$, and p_T by p_t, where $\overline{\boldsymbol{x}}_{t_+}^+$ $\overline{\boldsymbol{x}}_{t_-}^-$, and p_t are the current available estimators at time t. The precise form of this new function $\widehat{f}_t(\boldsymbol{w}_t)$ is described in the following theorem.

Theorem 1. *Define the loss of sample $\{\boldsymbol{x}_t, y_t\}$ as the following*

$$\begin{aligned}
\widehat{f}_t(\boldsymbol{w}_t; \{\boldsymbol{x}_t, y_t\}) := & (1-p_t)\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t - a_t(\boldsymbol{w}_{t-1})\right)^2 \mathbb{1}_{[y_t=1]} \\
& + p_t\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t - b_t(\boldsymbol{w}_{t-1})\right)^2 \mathbb{1}_{[y_t=-1]} - p_t(1-p_t)(\alpha_t(\boldsymbol{w}_{t-1}))^2 \\
& + 2(1 + \alpha_t(\boldsymbol{w}_{t-1}))\boldsymbol{w}_t^\top \boldsymbol{x}_t \Big(p_t \mathbb{1}_{[y_t=-1]} - (1-p_t)\mathbb{1}_{[y_t=1]} \Big), \quad (8)
\end{aligned}$$

where $p_t = \frac{\sum_{i=1}^t \mathbb{1}_{[y_i=1]}}{t}, \alpha_t(\boldsymbol{w}_{t-1}) = b_t(\boldsymbol{w}_{t-1}) - a_t(\boldsymbol{w}_{t-1})$

$$\begin{aligned}
a_t(\boldsymbol{w}_{t-1}) &= \boldsymbol{w}_{t-1}^\top \frac{\sum_{i=1}^t \boldsymbol{x}_i \mathbb{1}_{[y_i=1]}}{t}, \text{ and} \\
b_t(\boldsymbol{w}_{t-1}) &= \boldsymbol{w}_{t-1}^\top \frac{\sum_{i=1}^t \boldsymbol{x}_i \mathbb{1}_{[y_i=-1]}}{t}.
\end{aligned}$$

We denote the above definition as $\widehat{f}_t(\boldsymbol{w}_t)$ and it is convex with respect to $\boldsymbol{w}_t$. If $\|\boldsymbol{x}_t\|_0 = \mathcal{O}(k)$, then gradient of $\widehat{f}_t$ with respect to $\boldsymbol{w}_t$ is also a k sparse vector, i.e., $\|\nabla \widehat{f}_t(\boldsymbol{w}_t)\|_0 = \mathcal{O}(k)$.

Proof. Clearly, the convexity of $\widehat{f}_t(\boldsymbol{w}_t)$ can be checked by the Hessian matrix of $\widehat{f}_t(\boldsymbol{w}_t)$ with respect to $\boldsymbol{w}_t$, i.e,

$$H_{\widehat{f}_t}(\boldsymbol{w}_t) := \left(2(1-p_t)\mathbb{1}_{[y_t=1]} + 2p_t \mathbb{1}_{[y_t=-1]}\right)\boldsymbol{x}_t \boldsymbol{x}_t^\top.$$

Notice that the Hessian matrix $H_{\widehat{f}_t}(\boldsymbol{w}_t)$ is a rank-one matrix with a nonnegative coefficient, and hence it is a positive semi-definite matrix. Therefore, $\widehat{f}_t$ is convex. The gradient of $\widehat{f}_t(\boldsymbol{w}_t)$ with respect to $\boldsymbol{w}_t$ can be calculated as the following

$$\nabla \widehat{f}_t(\boldsymbol{w}_t) = \begin{cases} 2(1-p_t)\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t - b_t(\boldsymbol{w}_{t-1}) - 1\right)\boldsymbol{x}_t & y_t = 1 \\ 2p_t\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t - a_t(\boldsymbol{w}_{t-1}) + 1\right)\boldsymbol{x}_t & y_t = -1 \end{cases} \tag{9}$$

Notice that $\nabla \widehat{f}_t(\boldsymbol{w}_t)$ is a scaling of $\boldsymbol{x}_t$, hence, if $\|\boldsymbol{x}_t\|_0 = \mathcal{O}(k)$, then $\nabla \widehat{f}(\boldsymbol{w}_t)$ is also k sparse. □

Although the gradient of $\widehat{f}_t(\boldsymbol{w}_t)$ is sparse, it still needs to have $\mathcal{O}(d)$ cost by observing that (9) needs to calculate $a_t(\boldsymbol{w}_{t-1})$ and $b_t(\boldsymbol{w}_{t-1})$. In Section IV-B, we show that it is possible to obtain a sparse gradient of $\widehat{f}_t(\boldsymbol{w}_t)$ by using a "lazy update" trick in $\mathcal{O}(k)$ time.

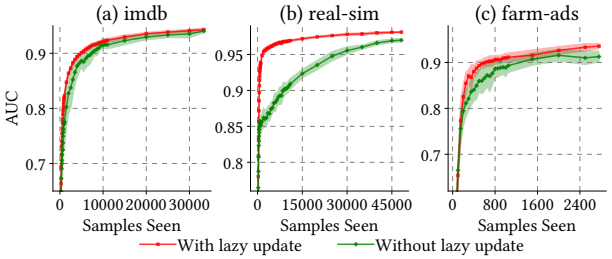

Fig. 2: Comparison between with and without lazy update

B. Pursuing $\mathcal{O}(k)$ per-iteration cost

Theorem 1 demonstrates that it is possible to calculate the gradient in $\mathcal{O}(k)$ per-iteration. However, the major difficulty is that we need to estimate the expectation of empirical score for positive and negative sample with respect to $\boldsymbol{w}$ up to t as the following

$$a_t(\boldsymbol{w}) = \boldsymbol{w}^\top \frac{\sum_{i=1}^t \boldsymbol{x}_i \mathbb{1}_{[y_i=1]}}{t}, b_t(\boldsymbol{w}) = \boldsymbol{w}^\top \frac{\sum_{i=1}^t \boldsymbol{x}_i \mathbb{1}_{[y_i=-1]}}{t}.$$

Given any $\boldsymbol{w}_{t-1}$, we can rewrite the above equations $a_t(\boldsymbol{w}_{t-1})$ and $b_t(\boldsymbol{w}_{t-1})$ as the following

$$a_t(\boldsymbol{w}_{t-1}) = \boldsymbol{w}_{t-1}^\top \frac{\sum_{i=1}^{t_+} \boldsymbol{x}_i^+}{t_+}, \quad b_t(\boldsymbol{w}_{t-1}) = \boldsymbol{w}_{t-1}^\top \frac{\sum_{j=1}^{t_-} \boldsymbol{x}_j^-}{t_-},$$

where $t = t_+ + t_-$. Take $a_t(\boldsymbol{w}_{t-1})$ as an example. We need to obtain the estimation of $a_t(\boldsymbol{w}_{t-1})$ where we denote it as $\widehat{a}_t(\boldsymbol{w}_{t-1})$. We reformulate the above equation and rewrite it as the following

$$\begin{aligned}
\widehat{a}_t(\boldsymbol{w}_{t-1}) &:= \boldsymbol{w}_{t-1}^\top \frac{\sum_{i=1}^{t_+-1} \boldsymbol{x}_i + \boldsymbol{x}_{t_+}^+}{t_+} \\
&= \frac{t_+ - 1}{t_+} \boldsymbol{w}_{t-1}^\top \frac{1}{t_+ - 1} \sum_{i=1}^{t_+-1} \boldsymbol{x}_i^+ + \frac{\boldsymbol{w}_{t-1}^\top \boldsymbol{x}_{t_+}^+}{t_+} \\
&\approx \frac{t_+ - 1}{t_+} \boldsymbol{w}_{t-2}^\top \frac{1}{t_+ - 1} \sum_{i=1}^{t_+-1} \boldsymbol{x}_i^+ + \frac{\boldsymbol{w}_{t-1}^\top \boldsymbol{x}_{t_+}^+}{t_+} \quad \textbf{(Lazy update)} \\
&= \frac{t_+ - 1}{t_+} \widehat{a}_{t-1}(\boldsymbol{w}_{t-2}) + \frac{\boldsymbol{w}_{t-1}^\top \boldsymbol{x}_{t_+}^+}{t_+}.
\end{aligned}$$

Similarly, $\widehat{b}_t(\boldsymbol{w}_{t-1})$ can be done in the same way above. We propose to update $a_t(\boldsymbol{w}_{t-1}), b_t(\boldsymbol{w}_{t-1})$ by their estimations $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ as the following

$$\begin{aligned}
\widehat{a}_t(\boldsymbol{w}_{t-1}) &= \frac{t_+ - 1}{t_+} \widehat{a}_{t-1}(\boldsymbol{w}_{t-2}) + \frac{\boldsymbol{w}_{t-1}^\top \boldsymbol{x}_{t_+}^+}{t_+}, \\
\widehat{b}_t(\boldsymbol{w}_{t-1}) &= \frac{t_- - 1}{t_-} \widehat{b}_{t-1}(\boldsymbol{w}_{t-2}) + \frac{\boldsymbol{w}_{t-1}^\top \boldsymbol{x}_{t_-}^-}{t_-}.
\end{aligned}$$

Clearly, per-iteration costs of $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ are $\mathcal{O}(k)$.

One may argue, algorithms without lazy update, as usually expected and taken-for-granted, will outperform those with lazy update. However, in our proposed algorithm, things are just the opposite. As a preliminary study, we apply both the gradient with and without lazy update step to our algorithm and test the two methods on three collected real-world datasets, imdb, real-sim, and farm-ads. As shown in Figure 2, the results by using the lazy update rule are surprisingly better and more stable than those without using it. It means that algorithm with lazy update rule could converge faster than the one without lazy update rule, especially at the early stage of the online learning processing. This could be due to the fact that, although $\boldsymbol{w}_t$ does not well converge to the true model $\boldsymbol{w}^*$ at the early stage, the estimator $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ which take the average of all previous models, i.e., $\sum_{i=1}^t \boldsymbol{w}_i/t$, will lead to a more stable behavior.

C. Sparsity-pursuing model

In this sub-section, we assume the gradient $\nabla \widehat{f}_i(\boldsymbol{w}_i)$ is the gradient by replacing $a_t(\boldsymbol{w}_{t-1})$ and $b_t(\boldsymbol{w}_{t-1})$ with $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ respectively. To make our model sparse, we apply the generalized Follow-The-Regularized-Leader framework proposed in a series of works [21, 22, 35]. For each time t, we propose to update the model $\boldsymbol{w}_t$ as

$$\begin{aligned}
\boldsymbol{w}_{t+1} = \arg\min_{\boldsymbol{w}} &\left\langle \sum_{i=1}^t \nabla \widehat{f}_i(\boldsymbol{w}_i), \boldsymbol{w} \right\rangle + t\lambda \|\boldsymbol{w}\|_1 \\
&+ \frac{1}{2} \sum_{i=1}^t \left\| \boldsymbol{Q}_t^{1/2} (\boldsymbol{w} - \boldsymbol{w}_i) \right\|_2^2,
\end{aligned}$$

where $\boldsymbol{Q}_t := \mathrm{diag}(q_{t,1}, q_{t,2}, \ldots, q_{t,d})$ is a positive definite matrix to control the adaptivity of regularizers and corresponding learning rate. This regularization term makes the updated model more stable [7]. We adopt the learning rate strategy suggested in [22] where $\eta_{t,i} = \frac{1}{\sum_{j=1}^t q_{j,i}}$. To pursue sparsity, we add ℓ^1-norm with parameter λ. The above minimization problem has the following closed-form solution

$$\boldsymbol{w}_{t+1} = -\eta_t \mathrm{sign}(\boldsymbol{z}_t)[|\boldsymbol{z}_t| - \lambda]_+,$$

where all operations are element-wise. More specifically, the i-th coordinate, $\boldsymbol{z}_t$ can be calculated recursively as the following

$$\boldsymbol{z}_{t,i} = \boldsymbol{z}_{t-1,i} + \boldsymbol{\nabla} \widehat{f}_{t,i}(\boldsymbol{w}_{t,i}) + \left(\frac{1}{\eta_{t,i}} - \frac{1}{\eta_{t-1,i}} \right) \boldsymbol{w}_{t,i},$$

where the per-coordinate learning rate is $\eta_{t,i} = \frac{\gamma}{1 + \sqrt{\sum_{j=1}^t \nabla \widehat{f}_{j,i}(\boldsymbol{w}_{j,i})^2}}$. Again, according to the previous analysis in Section IV-A and IV-B, the gradient is k sparse and can be updated by using $\mathcal{O}(k)$ time per-iteration. Interestingly, by using the above learning rate schedule, we do not need to update $\boldsymbol{w}_{t+1,i}$ of the i-th coordinate whenever $\boldsymbol{x}_{t,i} = 0$. Notice that $\nabla \widehat{f}_{t,i}(\boldsymbol{w}_{t,i}) = 0$ if $\boldsymbol{x}_{t,i} = 0$ because the gradient is the scaling vector of $\boldsymbol{x}_t$. Furthermore, we do not need to update $\boldsymbol{z}_{t,i}$ either if i-th entry of $\boldsymbol{x}_t$ is 0 by noticing the following equation

$$\begin{aligned}
\frac{1}{\eta_{t,i}} - \frac{1}{\eta_{t-1,i}} &= \frac{\sqrt{\sum_{j=1}^t \nabla \widehat{f}_{j,i}(\boldsymbol{w}_{j,i})^2} - \sqrt{\sum_{j=1}^{t-1} \nabla \widehat{f}_{j,i}(\boldsymbol{w}_{j,i})^2}}{\gamma} \\
&= \frac{\sqrt{\sum_{j=1}^{t-1} \nabla \widehat{f}_{j,i}(\boldsymbol{w}_{j,i})^2 + \nabla \widehat{f}_{t,i}(\boldsymbol{w}_{t,i})^2}}{\gamma} \\
&\quad - \frac{\sqrt{\sum_{j=1}^{t-1} \nabla \widehat{f}_{j,i}(\boldsymbol{w}_{j,i})^2}}{\gamma} = 0,
\end{aligned}$$

where the last equation due to $\nabla \widehat{f}_{t,i}(\boldsymbol{w}_{t,i}) = 0$. It means that $\boldsymbol{z}_t$ is also k-sparse, which could update $\boldsymbol{w}_{t+1}$ in $\mathcal{O}(k)$.

To summarize, the proposed algorithm FTRL-AUC is presented in Algorithm 1. It has two input parameters, the initial learning rate γ and sparse regularization (ℓ^1-norm) parameter λ. At the beginning, it initializes $p_0 = 0, \boldsymbol{w}_0 = \mathbf{0}, \widehat{a}_0(\boldsymbol{w}_{-1}) = 0, \widehat{b}_0(\boldsymbol{w}_{-1}) = 0$. At each time t, executes the following two main steps to update $\boldsymbol{w}_t$:

- At each time t, it receives a sample $\boldsymbol{x}_t$ in Line 5. The gradient of $\widehat{f}_t(\boldsymbol{w}_t)$ is $\boldsymbol{g}_t := \nabla \widehat{f}_t(\boldsymbol{w}_t)$ as shown in (9) which is calculated in Line 8 if y_t is positive; otherwise in Line 13. We use the lazy update steps to update $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ in Line 11 and 16 respectively. The proportion of positive samples p_t is updated in Line 9 and Line 14. Overall, it costs $\mathcal{O}(2k)$ from Line 5 to Line 17.
- After taking the gradient of current training sample $\boldsymbol{x}_t$. The $\boldsymbol{w}_t$ will be updated accordingly in Line 19. All previous gradients are accumulated in vector $\boldsymbol{z}$ in Line 20. From Line 18 to Line 22, it only needs $\mathcal{O}(k)$.

Algorithm 1 Follow-The-Regularized-Leader AUC (FTRL-AUC)

1: **Input**: γ, λ
2: $t_+ = 0, t_- = 0, p_0 = 0, \boldsymbol{w}_0 = \mathbf{0}, \widehat{a}_0(\boldsymbol{w}_{-1}) = 0, \widehat{b}_0(\boldsymbol{w}_{-1}) = 0$
3: $\boldsymbol{z} = \mathbf{0}, \boldsymbol{v} = \mathbf{0}, \boldsymbol{w}_0 = \mathbf{0}$
4: **for** $t = 0, 1, 2, \ldots$ **do**
5: receive $\{\boldsymbol{x}_t, y_t\}$
6: predict the score $\boldsymbol{w}_t^\top \boldsymbol{x}_t$
7: **if** $y_t = 1$ **then**
8: $\boldsymbol{g}_t = 2(1 - p_t)(\boldsymbol{w}_t^\top \boldsymbol{x}_t - \widehat{b}_t(\boldsymbol{w}_{t-1}) - 1) \cdot \boldsymbol{x}_t$
9: $p_{t+1} = \frac{t}{t+1} p_t + \frac{1}{t+1}$
10: $t_+ = t_+ + 1$
11: $\widehat{a}_{t+1}(\boldsymbol{w}_t) = \frac{t_+ - 1}{t_+} \widehat{a}_t(\boldsymbol{w}_{t-1}) + \frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{t_+}$
12: **else**
13: $\boldsymbol{g}_t = 2p_t(\boldsymbol{w}^\top \boldsymbol{x}_t - \widehat{a}_t(\boldsymbol{w}_{t-1}) + 1)\boldsymbol{x}_t$
14: $p_{t+1} = \frac{t}{t+1} p_t$
15: $t_- = t_- + 1$
16: $\widehat{b}_{t+1}(\boldsymbol{w}_t) = \frac{t_- - 1}{t_-} \widehat{b}_t(\boldsymbol{w}_{t-1}) + \frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{t_-}$
17: **end if**
18: **for** $i = \{j : x_{t,j} \neq 0, j \in \{1, 2, \ldots, d\}\}$ **do**
19: $w_{t+1,i} = -\frac{\gamma}{1 + \sqrt{v_i}} \text{sign}(z_i) \left[|z_i| - \lambda\right]_+$
20: $z_i = z_i + g_{t,i} - \frac{1}{\gamma}(\sqrt{v_i + g_{t,i}^2} - \sqrt{v_i}) w_{t,i}$
21: $v_i = v_i + g_{t,i}^2$
22: **end for**
23: **end for**

D. Complexity and Regret Analysis

Time and space complexity. The time complexity of the per-iteration as claimed is $\mathcal{O}(k)$. It needs $\mathcal{O}(k)$ operations to calculate the gradient from Line 7 to Line 18 and to update model $\boldsymbol{w}_t$. Other parameters such as $\boldsymbol{v}, \boldsymbol{z}$ and $\boldsymbol{g}_t$ also need $O(k)$ operations. Hence, the total is $\mathcal{O}(k)$. Furthermore, FTRL-AUC is also space efficient. During the learning process, it only keeps track of four vectors, $\boldsymbol{w}_t, \boldsymbol{z}, \boldsymbol{v}$, and $\boldsymbol{g}_t$. Hence, the memory requirement is $\mathcal{O}(d)$.

Regret analysis. Before closing this section, we briefly discuss the regret bound of our algorithm. Although it is unable to provide a sublinear regret bound for the original formulation as defined in (7), we can still provide a sublinear regret for the new loss as defined in (8) by directly applying Theorem 2 provided in [35]. However, one should notice that, to avoid defining new $\widehat{f}_t(\boldsymbol{w}_t)$ involving the estimators $\widehat{a}_t$ and $\widehat{b}_t$, just like we assumed in previous section, $\widehat{f}_t(\boldsymbol{w}_t)$ defined in (8) is actually the function replacing $a_t(\boldsymbol{w}_{t-1})$ and $b_t(\boldsymbol{w}_{t-1})$ with $\widehat{a}_t(\boldsymbol{w}_{t-1})$ and $\widehat{b}_t(\boldsymbol{w}_{t-1})$ respectively. Yet, this will not affect the convexity and corresponding analysis. More specifically, the regret with respect to $\widehat{f}_t(\boldsymbol{w}_t)$ can be defined as

$$\text{Regret}_T(\boldsymbol{w}) := \sum_{t=1}^{T} \widehat{f}_t(\boldsymbol{w}_t; \{\boldsymbol{x}_t, y_t\}) - \sum_{t=1}^{T} \widehat{f}_t(\boldsymbol{w}; \{\boldsymbol{x}_t, y_t\}).$$

If we further assume $\|\boldsymbol{w}\|_2 \leq R$ and $\|\nabla \widehat{f}_t(\boldsymbol{w})\| \leq L$ where R and L are two constants, by using (15) in Section 3.4 of [35], the regret bound of Algorithm 1 can be upper bounded as the following

$$Regret_T(\boldsymbol{w}) \leq 2\sqrt{2}R \sqrt{\sum_{t=1}^{T} \|\nabla \widehat{f}_t(\boldsymbol{w})\|_2^2} = \mathcal{O}(\sqrt{T}).$$

Therefore, FTRL-AUC has a sublinear regret $\mathcal{O}(\sqrt{T})$ for the new designed loss.

V. Experiments

To verify FTRL-AUC in experiments, we aim to answer the following three questions:

- **Q1**: Compared with the state-of-the-art online AUC optimization methods, can FTRL-AUC significantly shorten the run time?
- **Q2**: Can FTRL-AUC capture sparsity more effectively without significant loss of performance on AUC score?
- **Q3**: Does FTRL-AUC have any advantages over the online learning method for logistic loss when the dataset is imbalanced?

A. Datasets and baseline methods

Datasets. We consider the high-dimensional sparse datasets with the task of binary classification. All datasets are summarized in Table I. More specifically, n is the total number of samples, n_+ and n_- is the total number of positive and negative samples respectively. Datasets of real-sim, rcv1b, news20b, and avazu can be downloaded from [36]. The dataset of farm-ads [37] can be downloaded from [38]. There are two sentiment classification datasets, reviews [39] and imdb [40]. We also consider a click-through rate prediction dataset avazu [41] which has 1 million features and about 14 millions of training samples.

Baseline methods. We consider two types of method. The first type is for online AUC optimization. It has six methods, including SOLAM, a stochastic online AUC Maximization method proposed in [16], SPAM, a stochastic proximal AUC

TABLE I: High-dimensional sparse datasets

datasets	n	n_+	n_-	d	$\mathcal{O}(k)$[1]
real-sim	72,309	22,238	50,071	20,958	52
rcv1b	697,641	365,951	331,690	46,674	74
farm-ads	4,143	2,210	1,933	54,876	198
imdb	50,000	25,000	25,000	89,527	136
reviews	8,000	4,000	4,000	473,856	190
news20b	19,996	9,999	9,997	1,355,191	455
avazu	14,596,137	1,734,407	12,861,730	1,000,000	15

[1] $\mathcal{O}(k)$ of each dataset is calculated by $\lceil \sum_{i=1}^{n} \|\boldsymbol{x}_i\|_0 / n \rceil$. $\mathcal{O}(k)$ tells the number of nonzero entries on $\boldsymbol{x}_i$ on average.

maximization algorithm put forward in [19] (SPAM-ℓ^1, SPAM-ℓ^2, and SPAM-ℓ^1/ℓ^2 based on the different regularization strategy), FSAUC, a fast stochastic algorithm for true AUC maximization designed in [18], and SPAUC proposed in [17].[3] The second type is the online learning methods that minimize the logistic loss. We mainly consider the sparse-pursuing ones, including RDA-ℓ^1 [11], AdaGrad [10], and FTRL-Pro [5] which is an essentially Follow-The-Regularized-Leader approach proposed in [21].

Experimental setup. All methods are implemented in C language with a Python2.7 wrapper. We split all datasets in the following way: 4/6 samples are for training and the rest two 1/6 samples for validation and testing respectively. We repeat this procedure 10 times and report the results over these 10 trials. Parameter tuning and other detailed experimental setup can be found in the supplementary material provided in https://anonymous.4open.science/r/25f48408-b148-4651-adcf-82dcee2c9c75/. All methods stop after seeing all training samples. That is, all methods pass training samples once. AUC scores of all convergence curves in our experiments are calculated by using testing datasets.

B. Run time performance

To answer **Q1**, as we have claimed, one of the main improvements of our method over baselines is the time complexity. We test all methods on the six high-dimensional datasets and the run time has been shown in the top section of Table II. Clearly, the run time of FTRL-AUC significantly outperforms all the other methods by a large margin. For example, in news20b dataset, it is about 887 times faster than the existing fastest baseline, i.e., SPAM-ℓ^1 and about 1981 times faster than the slowest, i.e., FSAUC. The run time of FSAUC is worse than the others. This is because it needs to projection w_t on ℓ^1-ball and it is time consuming. However, our method only needs to have $\mathcal{O}(k)$ multiplication per-iteration, thus time-efficient. For example, in rcv1b dataset, FTRL-AUC only uses 1.609 seconds in average to process about 465,094 training samples with dimension $d = 46,674$.

C. Model sparsity and AUC performance

Q2 is affirmatively addressed in the following two parts.

[3]There are other online AUC optimization methods such as OAM [15], OPAUC [26] and AdaOAM [27]. However, these algorithms either need to have $\mathcal{O}(d \times d)$ memory or $\mathcal{O}(T \times d)$ run time, which makes them hard to be applied to high-dimensional sparse datasets.

Model sparsity. As discussed, the second improvement is that our algorithm can capture sparsity more effectively. To answer the first part of **Q2**, we measure the sparsity of final model $\boldsymbol{w}_t$ by using the sparse ratio which is defined as the following

$$\text{Sparse Ratio} := \frac{\|\boldsymbol{w}_t\|_0}{d}.$$

Without sacrificing the AUC score, the sparser the model, the better. The middle section of Table II shows clearly that the sparse ratio of the models obtained by FTRL-AUC is much sparser compared with other methods. For example, in imdb dataset, FTRL-AUC needs only about 2,865 features on average to get AUC score 0.94614, while SPAUC, the method getting the best AUC score 0.94983 needs to have up to 77,539 features on average. Clearly, this makes the ℓ^1-regularization less meaningful for SPAUC. The sparse ratio of SOLAM is always 1.0 because at the beginning, it needs $\boldsymbol{w}_0$ on the ball $\{\boldsymbol{w} : \|\boldsymbol{w}\| \leq R\}$. The excellent sparsity pursuing ability of our method can also been seen via the model selection phase which will be discussed in the later part of this section.

AUC score. We compare the AUC score of $\boldsymbol{w}_t$ on testing data of all methods. The bottom section of Table II presents the AUC scores. The up-to-now best performance of AUC score is SPAUC [17] in most cases, a recently developed AUC optimization method. One of the key properties of SPAUC is that each objective function is convex compared with SPAM-based methods. Another advantage of SPAUC is based on a stochastic proximal update step developed in [28]. By contrast, our method is consistently the runner-up among all online methods. It means that our method is competitive with respect to AUC score. For example, in real-sim, rcv1b, and imdb, the AUC scores of our method are only, 0.169%, 0.065%, and 0.388% less than these of SPAUC respectively. In new20b dataset, our method is 2.167% less than SPAUC. This is because our method uses $\mathcal{O}(1/\sqrt{t})$ learning rate which is slower than $\mathcal{O}(1/t)$ updates of SPAUC. Furthermore, by pursuing sparse solution, our method threshold out some less frequent but important nonzero features. However, SPAUC hardly obtain sparse solution by using an ℓ^1 regularization and has much slower run time. This indicates that our method could be a good alternative if real-world application needs sparse solution due to memory or run time consideration.

The convergence with respect to the run time is illustrated in Figure 3. This figure clearly demonstrates that our method converges extremely fast over time. The convergence with respect to the number of training samples can be found in supplementary material provided in https://anonymous.4open.science/r/25f48408-b148-4651-adcf-82dcee2c9c75/.

Sparsity tuning. To have a fair comparison, we choose λ, the ℓ^1 regularization parameter, from a large range set $\{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1.0, 3.0, 5.0\}$ for FTRL-AUC, SPAM-ℓ^1, SPAM-ℓ^1/ℓ^2, and SPAUC. We want to see if there is any AUC gain to pursue sparsity. Surprisingly, all baselines of online AUC optimization fail to gain AUC score when pursuing the sparsity. The experimental results

TABLE II: The performance of AUC optimization methods with respect to run time, sparse ratio, and AUC score

Datasets[1]	FTRL-AUC	SPAM-ℓ^1	SPAM-ℓ^2	SPAM-ℓ^1/ℓ^2	SOLAM	SPAUC	FSAUC[2]
Run Time[3] (seconds)							
farm-ads	**0.015±0.007**	1.597±0.014	0.746±0.221	1.586±0.053	1.614±0.430	1.875±0.137	2.346±0.335
real-sim	**0.106±0.029**	10.513±0.284	3.562±0.855	11.521±0.696	9.741±3.307	14.200±0.543	17.623±2.704
rcv1b	**1.609±0.155**	230.769±7.121	103.251±25.46	234.379±0.491	178.989±31.705	334.820±28.554	424.636±73.22
news20b	**0.324±0.059**	287.676±18.94	355.556±30.83	397.424±1.688	404.911±151.445	347.866±8.039	642.008±105.9
reviews	**0.025±0.015**	28.334±6.604	26.529±10.369	35.035±0.607	15.054±5.854	22.944±5.437	21.740±7.169
imdb	**0.355±0.046**	30.717±0.984	11.603±1.271	58.007±4.895	26.014±6.423	39.728±0.905	53.630±4.183
Sparse Ratio ($\|\boldsymbol{w}_t\|_0/d$)							
farmads	**.0130±.0059**	.2903±.0541	.8099±.0203	.3072±.0101	1.0000±.0000	.1933±.0137	.8095±.0202
real-sim	**.3236±.0275**	1.0000±.0000	1.0000±.0000	.9666±.0301	1.0000±.0000	.9792±.0571	1.0000±.0000
rcv1b	**.2987±.0273**	.8398±.1175	.9571±.0010	.9054±.0629	1.0000±.0000	.7683±.0193	.9571±.0010
news20b	**.0016±.0004**	.9224±.0037	.9224±.0037	.9224±.0037	1.0000±.0000	.9220±.0039	.9208±.0039
reviews	**.0006±.0002**	.6622±.1976	.7310±.0051	.6618±.1977	1.0000±.0000	.5155±.2409	.7292±.0064
imdb	**.0320±.0081**	.8661±.0009	.8661±.0009	.6083±.3940	1.0000±.0000	.8661±.0009	.8661±.0009
AUC score							
farm-ads	.94290±.00669[4]	.92486±.00631	.92914±.01086	.92609±.00599	.94210±.00899	**.95402±.00511**	.95212±.00691
real-sim	.99226±.00081	.98541±.00115	.98741±.00111	.97911±.00532	.99141±.00079	**.99394±.00058**	.99331±.00059
rcv1b	.99488±.00017	.99258±.00074	.99365±.00031	.99257±.00081	.99358±.00016	.99553±.00013	**.99544±.0001**
news20b	.97434±.00193	.94420±.02221	.96008±.00796	.94906±.02245	.95411±.00228	**.99240±.00104**	.98260±.00247
reviews	.91320±.00840	.88822±.01426	.89825±.01426	.89407±.01304	.90799±.01220	**.93343±.00915**	.91826±.00592
imdb	.94614±.00275	.86767±.01199	.86911±.01198	.80066±.10940	.89735±.00541	**.94983±.00223**	.92644±.00520

[1] All reported values are averaged on 10 trials of outcomes by randomly shuffling all training samples.
[2] One should notice that FSAUC is not a true online learning algorithm. It needs to have the total number of input samples as a prior. We treat it as an offline algorithm as a reference and do not attend to compare it with other online methods.
[3] The run time of a specific method is the total running time of passing whole training samples once.
[4] Underline of the AUC score means that it is the runner-up among all online methods (excluding the offline method FSAUC).

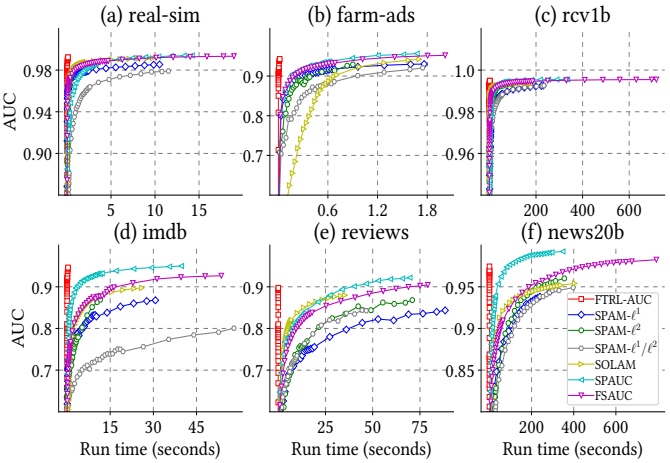

Fig. 3: Convergence Rate with respect to run time

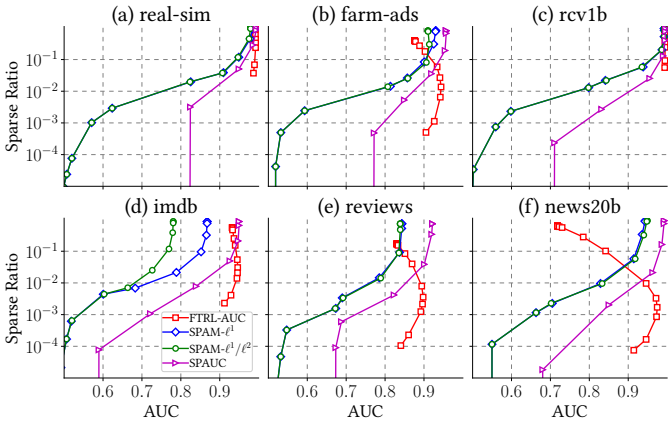

Fig. 4: Sparse Ratio as a function of the AUC score

are illustrated in Figure 4. Clearly, FTRL-AUC boosts the AUC score when it tries the λ from lowest 10^{-8} to high. However, for the other three methods, the AUC scores decrease dramatically when λ increases. One possible explanations is that the ability of FTRL tries to use all previous gradient information $\sum_{t=1}^{T} \nabla \widehat{f}_t(\boldsymbol{w}_t)$ while all the other methods only use current gradient information. To approximate the true gradient, the accumulated gradients are clearly more stable and effective than the single gradient.

D. Performance on imbalanced datasets

As we have mentioned, our method could better optimize the AUC score when the dataset is imbalanced. To answer **Q3**, we compare FTRL-AUC with the online learning algorithms for logistic loss including RDA-ℓ^1, AdaGrad, and FTRL-Pro. To make these six high-dimensional sparse datasets imbalanced, we use the following strategy: In our first experiment, we keep all negative training samples (T_- in total) and only keep the first $0.1 * T_-$ positive samples so that the imbalance ratio is low $T_+/T_- = 0.1$. Similarly, in our second experiment, we only keep the first $0.05 * T_-$ positive samples. The rest experimental setup remains unchanged.

The AUC scores of these two experiments are reported in Table III. Compared with FTRL-Pro[4], our method have much

[4]The only difference between our method and FTRL-Pro is that FTRL-Pro uses logistic loss but our method uses AUC loss defined in (3).

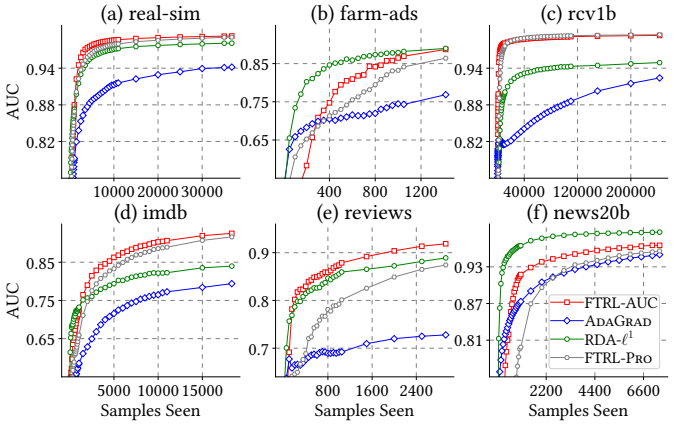

Fig. 5: AUC as function of the number of samples seen

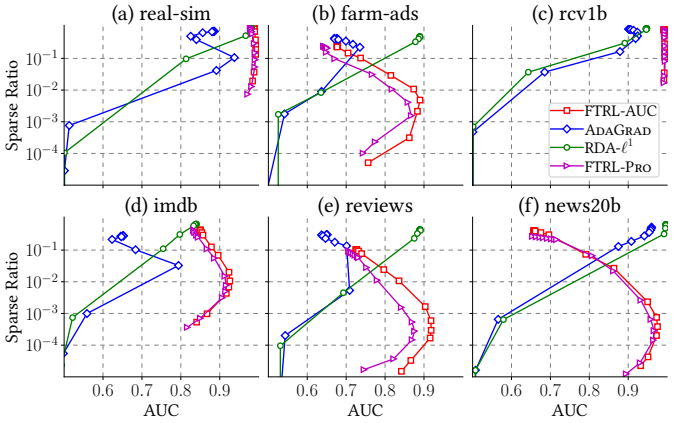

Fig. 6: Sparse Ratio as a function of AUC score

TABLE III: Comparison of AUC score with logistic loss methods

Datasets	FTRL-AUC	AdaGrad	RDA-ℓ^1	FTRL-Pro
AUC Score ($T_+/T_- = 0.1$)				
farm-ads	.8870±.0169	.7686±.0487	**.8897±.0335**	.8641±.0203
real-sim	**.9926±.0007**	.9417±.0046	.9809±.0018	.9907±.0012
rcv1b	.9935±.0004	.9240±.0023	.9491±.0018	**.9947±.0003**
news20b	.9752±.0054	.9599±.0061	**.9967±.0010**	.9660±.0062
reviews	**.9187±.0130**	.7278±.0341	.8891±.0161	.8742±.0088
imdb	**.9256±.0055**	.7939±.0164	.8399±.0178	.9163±.0050
AUC Score ($T_+/T_- = 0.05$)				
farm-ads	.8723±.0343	.6697±.1260	**.8784±.0331**	.8076±.0515
real-sim	**.9939±.0007**	.9413±.0066	.9789±.0031	.9899±.0014
rcv1b	.9925±.0003	.8997±.0039	.9439±.0019	**.9938±.0006**
news20b	.9638±.0120	.9490±.0173	**.9969±.0017**	.9541±.0140
reviews	**.8781±.0291**	.6584±.0505	.8746±.0236	.8260±.0297
imdb	**.9133±.0152**	.8159±.0262	.8412±.0175	.9021±.0163

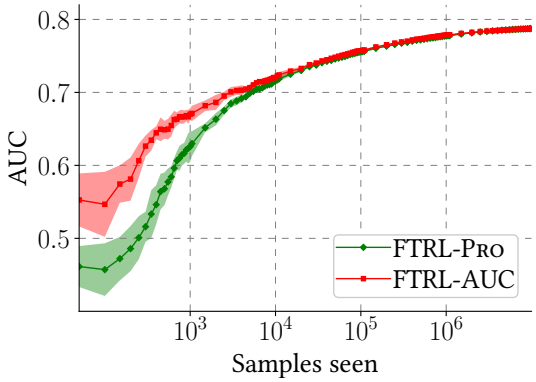

Fig. 7: The convergence curve with respect to the number of training samples seen on Avazu dataset

higher AUC score on farm-ads, reviews. The best performance of FTRL-Pro on rcv1b is 0.9947 while RDA-ℓ^1 obtains the best performance on farm-ads and news20b. For rcv1b dataset, our method is only 0.12% less than the best one, still competitive. From Figure 5 ($T_+/T = 0.1$), we can see that the convergence curve of FTRL-AUC than that of FTRL-Pro, indicating that advantage of our method. One may notice that RDA-ℓ^1 is much better than ours on news20b dataset. This may be because RDA-ℓ^1 but it performs much worse than ours on imdb, reviews, and rcv1b datasets. Figure 6 presents the parameter tuning of λ of these four methods.

Avazu: Click-through rate prediction. The task of online click-through rate prediction is to predict whether a user will click an advertisement or not. We compare our method with FTRL-Pro on avazu of the Kaggle dataset where the original dataset is released at https://www.kaggle.com/c/avazu-ctr-prediction, which has about 14 million samples. In the feature engineering step, we use the data preprocessing step, a field-aware factorization machines proposed in [41]. Each training sample has 1 million features after the factorization processing. Again, we split them as training, validation, and testing samples by 4:1:1. Both methods have the same parameter space. We randomly shuffle the dataset 10 times and run both methods and report the convergence curve in Figure 7. When the two algorithms receive enough training samples, the AUC scores do not have much difference. However, at the early stage of the learning process, FTRL-AUC achieves significantly higher AUC scores than FTRL-Pro.

VI. Conclusion and Future Work

To conclude, in this paper, we propose a faster online AUC optimization algorithm based on a generalized follow-the-regularized-leader framework. By using a new "lazy update" formula, we reduce the per-iteration time cost from $\mathcal{O}(d)$ to $\mathcal{O}(k)$. Our experimental results demonstrate that FTRL-AUC can significantly reduce the run time as well as obtain sparse models more effectively. This makes our method attractive to very high-dimensional sparse datasets. For the future work, an interesting research question is that what is the error between the proposed loss and the true loss and how to control it. Furthermore, we can try to improve the AUC score performance without loss of the run time and model sparsity advantages. One potential direction is to explore how to incorporate the second order information. Also, it remains interesting to see if our work can be generalized to multi-classification tasks.

References

[1] Andrew P Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, pages 1145–1159, 1997.

[2] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, pages 861–874, 2006.

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

[4] David Sculley and Gabriel M Wachman. Relaxed online svms for spam filtering. In *SIGIR*, page 415–422, 2007.

[5] H. Brendan McMahan, Gary Holt, and et al. Ad click prediction: A view from the trenches. In *KDD*, pages 1222–1230, 2013.

[6] Justin Ma, Lawrence K. Saul, and et al. Identifying suspicious urls: An application of large-scale online learning. In *ICML*, page 681–688, 2009.

[7] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[8] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[9] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, page 2121–2159, 2011.

[11] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *JMLR*, pages 2543–2596, 2010.

[12] Stéphan Clémençon, Gábor Lugosi, Nicolas Vayatis, et al. Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, 36(2):844–874, 2008.

[13] Yuyang Wang, Roni Khardon, and et al. Generalization bounds for online learning algorithms with pairwise loss functions. In *ALT*, pages 13.1–13.22. PMLR, 2012.

[14] Purushottam Kar, Bharath K. Sriperumbudur, and et al. On the generalization ability of online learning algorithms for pairwise loss functions. In *ICML*, page 441–449, 2013.

[15] Peilin Zhao, Steven C. H. Hoi, and et al. Online auc maximization. In *ICML*, page 233–240, 2011.

[16] Yiming Ying, Longyin Wen, and Siwei Lyu. Stochastic online auc maximization. In *NIPS*, page 451–459, 2016.

[17] Yunwen Lei and Yiming Ying. Stochastic proximal AUC maximization. *CoRR*, abs/1906.06053, 2019.

[18] Mingrui Liu, Xiaoxuan Zhang, and et al. Fast stochastic AUC maximization with $o(1/n)$-convergence rate. In *ICML*, pages 3189–3197, 2018.

[19] Michael Natole, Jr., Yiming Ying, and Siwei Lyu. Stochastic proximal algorithms for AUC maximization. In *ICML*, pages 3710–3719, 2018.

[20] John Langford and et al. Sparse online learning via truncated gradient. *JMLR*, page 777–801, 2009.

[21] H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *COLT*, pages 00–15, 2010.

[22] Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, pages 525–533, 2011.

[23] Thorsten Joachims. A support vector method for multivariate performance measures. In *ICML*, page 377–384, 2005.

[24] Alan Herschtal and Bhavani Raskutti. Optimising area under the ROC curve using gradient descent. In *ICML*, page 49, 2004.

[25] Xinhua Zhang and et al. Smoothing multivariate performance measures. *JMLR*, 13:3623–3680, 2012.

[26] Wei Gao, Rong Jin, and et al. One-pass AUC optimization. In *ICML*, pages 906–914, 2013.

[27] Yi Ding, Peilin Zhao, and et al. An adaptive gradient method for online auc maximization. In *AAAI*, page 2568–2574, 2015.

[28] John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *JMLR*, pages 2899–2934, 2009.

[29] Haiqin Yang, Zenglin Xu, and et al. Online learning for group lasso. In *ICML*, page 1191–1198, 2010.

[30] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[31] Shai Shalev-Shwartz and Yoram Singer. A primal-dual perspective of online learning algorithms. *Machine Learning*, 69(2):115–142, Dec 2007.

[32] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957.

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

[34] Yiming Ying and Ding-Xuan Zhou. Online pairwise learning algorithms. *Neural Comput.*, page 743–777, 2016.

[35] H. Brendan McMahan. A survey of algorithms and analysis for adaptive online learning. *JMLR*, page 3117–3166, 2017.

[36] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[37] Chris Mesterharm and Michael J. Pazzani. Active learning using on-line algorithms. In *KDD*, page 850–858, 2011.

[38] C.L. Blake D.J. Newman and C.J. Merz. UCI repository of machine learning databases, 1998.

[39] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

[40] Andrew L. Maas, Raymond E. Daly, and et al. Learning word vectors for sentiment analysis. In *ACL*, page 142–150, 2011.

[41] Yuchin Juan, Yong Zhuang, and et al. Field-aware factorization machines for ctr prediction. In *RecSys*, page 43–50, 2016.

[42] Zhang Xianyi, Wang Qian, and Zhang Yunquan. Model-driven level 3 blas performance optimization on loongson 3a processor. In *18th international conference on parallel and distributed systems*, pages 684–691, Singapore, 2012. IEEE.

[43] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML*, page 272–279, 2008.

[44] Laurent Condat. Fast projection onto the simplex and the l1 ball. *Mathematical Programming*, pages 575–585, 2016.

APPENDIX

A. Reproducibility and detailed experimental Setup

Implementation Details. To reproduce results including results of baselines, we present implementation details as follows:

- All methods are implemented in C11, a C standard revision, language with Python2.7 as a wrapper. The experiments are executed in a cluster with 5 nodes. Each node has 28 CPUs and 250Gb memory. For each method, we only use 1 CPU at a time.
- The random seeds for all trials are np.random.seed(17), which makes results of AUC scores and sparse ratios reproducible.
- Critical operations of all baseline methods are scale product $c\cdot \boldsymbol{x}$ and the inner product $\langle \boldsymbol{x}, \boldsymbol{y}\rangle = \boldsymbol{x}^\top \boldsymbol{y}$, which are calculated by cblas_dscal() and cblas_ddot() respectively. These two functions are provided by OpenBLAS [42] [5], an optimized BLAS library.
- For SPAM-ℓ^1, SPAM-ℓ^2, and SPAM-ℓ^1/ℓ^2, since they need to estimate $\widehat{p}_T, a(\boldsymbol{w}_t), b(\boldsymbol{w}_t)$, and $\alpha(\boldsymbol{w}_t)$, in our experiments, we estimate them by using $p_t, a_t(\boldsymbol{w}_t), b_t(\boldsymbol{w}_t)$, and $\alpha_t(\boldsymbol{w}_t)$ defined in (8).
- For FSAUC, there exists a projection step onto a ℓ^1-norm ball. The projection used in the original implementation is the method proposed in [43]. However, there exists a much faster version of ℓ^1-ball projection [44] as claimed has $\mathcal{O}(d)$ run time in practice [6].

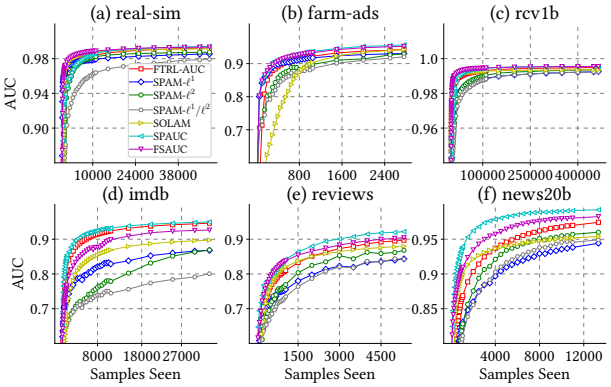

Fig. 8: Convergence rate with respect to the number of training samples seen

Parameter Tuning. We list parameter tuning of all methods including the baseline methods as follows:

- FTRL-AUC, it has two parameters. The ℓ^1-regularization parameter λ which is from a sufficient large range $\{10^{-8}, 10^{-7}, \ldots, 10^{-3}, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1.0, 3.0, 5.0\}$, and the initial learning rate γ is from $\{10^{-5}, 5\cdot 10^{-5}, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0\}$.
- SPAM-ℓ^1 has two parameters. The initial learning rate ξ is from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$. The ℓ^1-regularization parameter is the same as FTRL-AUC's.
- SPAM-ℓ^2 has two parameters. The initial learning rate ξ is from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$. The ℓ^2-regularization parameter is the same as λ.
- SPAM-ℓ^1/ℓ^2 has 3 main parameters. To avoid large cross-validation time, the parameter ξ and λ^2 is used by the parameter tuned from SPAM-ℓ^2. We only tune the ℓ^1 parameter λ_1 which is the same λ.
- FSAUC has 2 parameters. The ℓ^1-norm ball which is from $\{10^{-1}, 10^{0}, \ldots, 10^{5}\}$. The corresponding initial learning rate is from $\{2^{-10}, 2^{-9}, 2^{-8}, \ldots, 2^{8}, 2^{9}, 2^{10}\}$ as suggested in [18].
- SOLAM has two parameters. The ℓ^2-norm ball diameter which is from $\{10^{-1}, 10^{0}, \ldots, 10^{5}\}$ and the initial learning rate $\xi \in \{1.0, 10.0, 19.0, 28.0, \ldots, 100.0\}$ as suggested in [16].
- SPAUC has two parameters. The initial learning rate parameter is from $\{10^{-7.0}, 10^{-6.5}, 10^{-6.0}, \ldots, 10^{-2.5}\}$. Since we use the ℓ^1-regularization and it is the same λ.
- FTRL-PRO has the same parameter tuning strategy as FTRL-AUC.
- RDA-ℓ^1 has three parameters. It has an initial learning rate from the range $\{10.0, 50.0, 100.0, 500.0, 1000.0, 5000.0\}$. The sparsity-enhancing parameter is from $\{0.0, 0.005\}$, where 0.0 corresponding to non-enhancing sparsity. The λ is the same as FTRL-AUC's.
- ADAGRAD has three parameters. The ϵ is fixed to 10^{-8} to avoid the divided by zero error. The learning rate parameter is from $\{0.001, 0.01, 0.1, 1.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 5000.0\}$ while λ is the same as others.

B. More Results

We first provide the experimental details for Figure 2. We fix the initial learning rate $\gamma = 1.0$ and the sparsity parameter $\lambda = 0.5$. The convergence curves illustrate in Figure 2 and 9 are the AUC scores averaged on 10 trials.

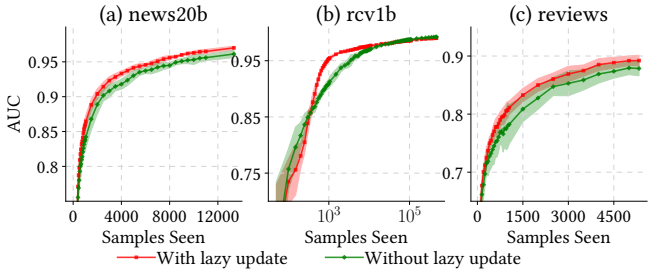

Fig. 9: Comparison of with and without "lazy update"

The convergence curve with respect to the number training samples seen are illustrated in Figure 8. In general, the performance of FTRL-AUC on the convergence is better than SPAM-based. Figure 10 illustrates the convergence curve as a function the number of training samples seen for the datasets

[5] https://github.com/xianyi/OpenBLAS with version 0.3.1 (Accessed in February 2020)

[6] The C version code can be download from https://lcondat.github.io/download/condat_l1ballproj.c(AccessedinFebruary2020).

of imbalance ratio $T_+/T = 0.05$. Figure 11 illustrate the sparse ratio and corresponding AUC scores as a function of the parameter λ for the datasets of imbalance ratio $T_+/T = 0.05$.

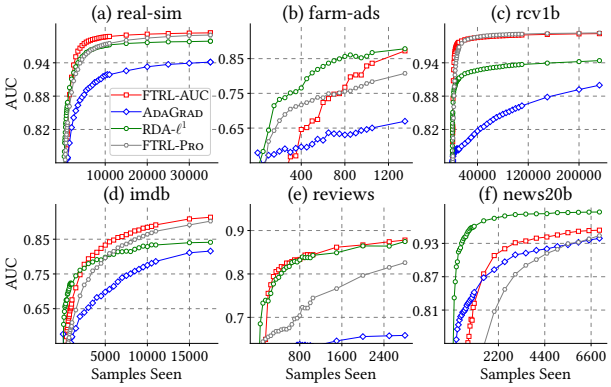

Fig. 10: Convergence curve with respect to the number of training samples seen

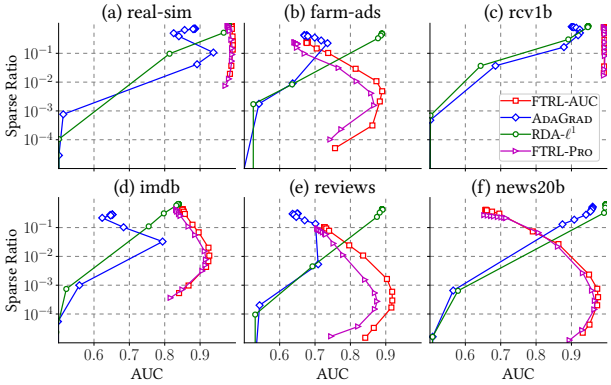

Fig. 11: Sparse Ratio as a function of the AUC score.