Metric-Optimized Example Weights

Sen Zhao [*1] **Mahdi Milani Fard** [*1] **Harikrishna Narasimhan** [1] **Maya Gupta** [1]

Abstract

Real-world machine learning applications often have complex test metrics, and may have training and test data that are not identically distributed. Motivated by known connections between complex test metrics and cost-weighted learning, we propose addressing these issues by using a weighted loss function with a standard loss, where the weights on the training examples are learned to optimize the test metric on a validation set. These metric-optimized example weights can be learned for any test metric, including black box and customized ones for specific applications. We illustrate the performance of the proposed method on diverse public benchmark datasets and real-world applications. We also provide a generalization bound for the method.

1. Introduction

In machine learning, each example is usually weighted equally during training. Such uniform weighting delivers satisfactory performance when training and test examples are independent and identically distributed (IID), and the training loss matches the test metric. However, these requirements are often violated in real-world applications, as the real-world goals for a model are often quite complicated. If the training loss does not correlate sufficiently with the test metric, inferior test performance can result (Cortes & Mohri, 2004; Perlich et al., 2003; Davis & Goadrich, 2006).

If the test metric has a sufficiently nice structure, and the train and test distributions are IID, it may be possible to specify a useful example-weighted training loss, and train for it (Koyejo et al., 2014; Parambath et al., 2014; Narasimhan et al., 2015b;a). In this paper, we extend the strategy of using training example weights to arbitrary test metrics, by learning a function to weight the examples that best optimizes the test metric on a validation set. Our proposal, *metric-optimized example weights (MOEW)* is applicable for *any* test metric. MOEW can address non-IID test samples if there is available a small set of labeled validation examples that are IID with the test examples. By learning a weighting function, MOEW effectively rescales the loss on each training example, and reshapes the overall loss such that its optima better match the optima of the test metric.

As an illustrative example, Figure 1 shows a simulated dataset with non-IID training and test examples, and the learned MOEW example weighting. The goal is to maximize precision at 95% recall on the test distribution. At 95% recall, with uniform weighting the precision is 20.8%; with optimal importance weighting (Shimodaira, 2000) it is better at 21.8%, but with MOEW it can be improved to 23.2%, much closer to the Bayes optimal of 25%. Comparing Figure 1c to 1d, one can see that MOEW learns to upweight negative training examples compared to positive examples, and to upweight examples closer to the center.

2. Related Work

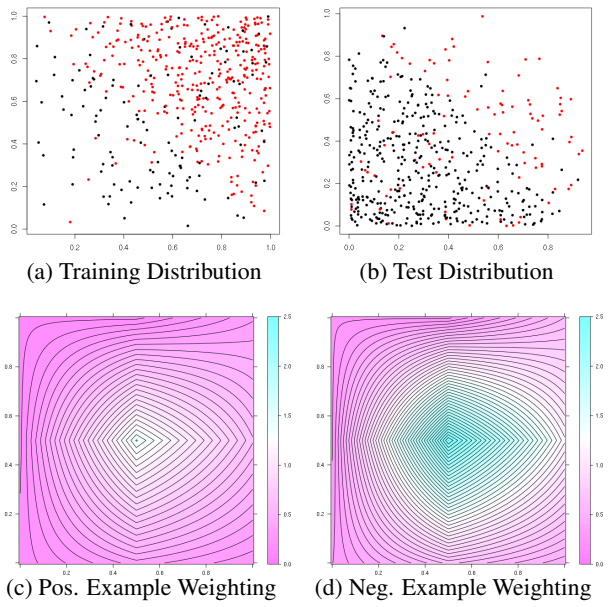
Our proposed approach simultaneously addresses two key issues: non-IID training and test sets, and mismatch between training loss and test metrics.

For the first issue, classical approaches include propensity matching (Lunceford & Davidian, 2004) and importance weighting (Shimodaira, 2000; Sugiyama et al., 2007; 2008; Kanamori et al., 2009). Bickel et al. (2009) also proposed a discriminative approach for learning under covariate shift . None of these approaches handle a general test metric.

For the second issue, a variety of approaches have been developed to directly optimize complex metrics on the training set. These include (i) *plug-in* approaches (Koyejo et al., 2014; Narasimhan et al., 2014), (ii) *surrogate loss optimization* for AUC (Ferri et al., 2002; Yan et al., 2003; Cortes & Mohri, 2004; Freund et al., 2003; Herschtal & Raskutti, 2004; Rudin & Schapire, 2009; Zhao et al., 2011), the F-measure (Joachims, 2005; Jansche, 2005) and other ranking metrics (Yue et al., 2007; Eban et al., 2017; Kar et al., 2014; 2015), and (iii) *sequential example-weighting* techniques for complex metrics (Parambath et al., 2014; Narasimhan et al., 2015b;a) and constraints (Agarwal et al., 2018; Kearns et al.,

[*]Equal contribution [1]Google AI, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA. Correspondence to: Sen Zhao <senzhao@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.



(a) Training Distribution (b) Test Distribution

(c) Pos. Example Weighting (d) Neg. Example Weighting

Figure 1: Figures 1a and 1b show the distribution of the training and validation/test data, where black dots represent negative examples and red dots represent positive examples. The two features were drawn from beta distributions: training $(x_1, x_2) \sim (\beta(2,1), \beta(2,1))$, but validation/test, $(x_1, x_2) \sim (\beta(1,2), \beta(1,2))$. Figures 1c and 1d show contour lines of the MOEW weighting function for positive and negative examples, respectively.

2018; Narasimhan, 2018). These methods crucially rely on the test metric having a specific closed-form structure and do not offer the generality of our approach.

Our approach generalizes the idea of sequential weighting techniques to arbitrary (possibly black-box) test metrics and to non-IID train and test distributions.

3. Metric Adaptive Weight Optimization

We now describe our proposed adaptive example weighting approach and provide theoretical justifications in Section 4.

3.1. Overview

We consider classification and regression problems where examples $x \in \mathbb{R}^D$ and labels $y \in \mathcal{Y} \subseteq \mathbb{R}$. We allow the training and test distribution to be non-IID, but sharing the same support.

We are interested in learning a model, $h(x;\theta) \in \mathcal{H}$, which is parameterized by $\theta \in \mathbb{R}^D$. The performance of the classifier or regressor is evaluated using a metric $M(\theta)$ defined on a test distribution (with higher values of M being better). Let $\mathcal{T}$ and $\mathcal{V}$ denote the sets of training and validation examples, respectively. We assume that $\mathcal{V}$ is drawn IID from the test distribution, but that $\mathcal{T}$ and $\mathcal{V}$ may not be IID. The goal is to use the training and validation sets to find model θ^* that maximizes the test metric M.

It is known that for a wide range of evaluation metrics that can be written as a function of simpler expected point-wise losses, the optimal parameters θ can be found by minimizing a particular example-weighted loss function (Parambath et al., 2014; Narasimhan et al., 2015a;b) (see Section 4 for details). Motivated by these results, we propose to directly learn an example weighting on the training examples that yields a model with optimal test metric.

In particular, we use an example weighting function $w : \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}_+$ parameterized by α that maps a training feature vector and label to a non-negative weight. For a predicted label $\hat{y}$ and true label y, define a loss function $L(\hat{y}, y)$. Find the optimal $\hat{\theta}$ that minimizes the corresponding weighted loss on the training set

$$\hat{\theta}(\alpha) = \arg\min_{\theta} \sum_{j \in \mathcal{T}} w\left(x_j, y_j; \alpha\right) L\left(h\left(x_j;\theta\right), y_j\right). \quad (1)$$

Let $\hat{M}(x_{\mathcal{V}}, y_{\mathcal{V}}; \theta)$ denote the estimate of the test metric on the validation dataset $\mathcal{V}$. Indeed one could consider directly optimizing $\hat{M}(x_{\mathcal{V}}, y_{\mathcal{V}}; \theta)$ over $\theta \in \mathbb{R}^D$. However, for a high-dimensional θ and a small $\mathcal{V}$, we may end up overfitting the validation set, and performing poorly on the test metric. Instead, we propose learning the example weighting function $w(x, y; \hat{\alpha})$, such that:

$$\hat{\alpha} = \arg\max_{\alpha} \hat{M}(x_{\mathcal{V}}, y_{\mathcal{V}}; \hat{\theta}(\alpha)). \quad (2)$$

In other words, we propose finding the optimal parameters $\hat{\alpha}$ for the example weighting function $w(x, y; \hat{\alpha})$ such that the model that optimizes the example-weighted loss on the training set achieves the best validation score. The benefit of this approach is that α can be constructed to be low-dimensional, which can be optimized to maximize M on the validation set, whereas θ, being high-dimensional, is optimized on the larger training set.

To simplify the notation, where possible, we denote $\theta(\alpha)$ by θ and $\hat{M}(x_{\mathcal{V}}, y_{\mathcal{V}}; \theta)$ by $\hat{M}(\theta)$. The validation metric $\hat{M}(\theta)$ as a function of θ and α is likely non-convex and non-differentiable, which makes it hard to be directly optimized through, e.g., SGD. Instead, we adopt an iterative algorithm to optimize for θ^* and $\hat{\alpha}$, which is detailed in Algorithm 1.

We start with a random sample of K weighting parameters, $\alpha^0 = \{\alpha_1^0, \ldots, \alpha_K^0\}$. For each of α_l^0, $1 \le l \le K$, we solve equation 1 to obtain K corresponding model parameters $\hat{\theta}_l^0$, and use those to compute K validation metrics, $\hat{M}(\hat{\theta}_l^0)$. Then, based on the batch of K weighting parameters and validation metrics, we determine a new set of K weight parameter candidates. This step is performed by a call to the

Algorithm 1 Get optimal $\hat{\alpha}$ and $\hat{\theta}(\hat{\alpha})$

$\mathcal{T} \leftarrow$ training data
$\mathcal{V} \leftarrow$ validation data
$B \in \mathbb{N}_+ \leftarrow$ number of batches of weight parameters
$\alpha^0 = \{\alpha_1^0, \ldots, \alpha_K^0\} \leftarrow$ initial weight parameters
for $i = 0, 1, \ldots, B-1$ **do**
  for all $l \in \{1, \ldots, K\}$ **do**
    $\hat{\theta}_l^i \leftarrow \arg\min_\theta \sum_{j \in \mathcal{T}} w(x_j, y_j; \alpha_l^i) L(h(x_j; \theta), y_j)$
  end for
  $s^i \leftarrow \{(\alpha_1^i, \hat{M}(\hat{\theta}_1^i)), \ldots, (\alpha_K^i, \hat{M}(\hat{\theta}_K^i))\}$
  $\alpha^{i+1} \leftarrow$ *GetCandidateAlphas*$(s^0, \ldots, s^i)$
end for
$\hat{b}, \hat{k} \leftarrow \arg\max_{b \in \{0,\ldots,B-1\}, k \in \{1,\ldots,K\}} \hat{M}(\theta_k^b)$
$\hat{\alpha}, \hat{\theta} \leftarrow \alpha_{\hat{k}}^{\hat{b}}, \hat{\theta}_{\hat{k}}^{\hat{b}}$

subroutine *GetCandidateAlphas*. The process is repeated for B iterations. At the end, we choose the candidate α that produced the best validation metric.

In the following subsections, we describe the function class $\mathcal{W}$ of the weighting model, and one possible instantiation of the *GetCandidateAlphas* subroutine used in Algorithm 1.

3.2. Function Class for the Example Weighting Model

Recall that the final optimal $\hat{\alpha}$ is taken to be the best out of $B \times K$ samples of α's. To ensure a sufficient coverage of the weight parameter space for a small number of $B \times K$ validation samples, we found it best to use a function class $\mathcal{W}$ with a small number of parameters.

There are many reasonable strategies for defining $\mathcal{W}$. In this paper, we chose the functional form:

$$w(x, y; \alpha) = c\, \pi(y)\, \sigma\left(z(x, y)^\top \alpha\right), \tag{3}$$

where $c \in \mathbb{R}_+$ is a constant that normalizes the weights over (a batch from) the training set $\mathcal{T}$, and $\sigma(z(x,y)^\top \alpha)$ is a sigmoid transformation of a linear function of a low-dimensional embedding z of (x, y). In the experiments discussed in Section 5, we used the standard importance function $\pi(y) = p_{\mathcal{V}}(y) / p_{\mathcal{T}}(y)$, where $p_{\mathcal{V}}(y)$ and $p_{\mathcal{T}}(y)$ denote the probability density function of y in the validation and training data, respectively. In practice, $\pi(y)$ can be substituted with any baseline weighting function, which can be considered as an initialization of MOEW.

While there are many ways to form a low-dimensional embedding z of (x, y), we choose to use an autoencoder. To train the autoencoder $z(x, y; \hat{\xi})$, we minimize the weighted sum of the reconstruction loss for x and y:

$$\sum_{(x,y) \in \mathcal{T}} \left\{\lambda\, L_x(z(x, y; \xi), x) + (1 - \lambda)\, L_y(z(x, y; \xi), y)\right\},$$

where L_x is an appropriate loss for the feature vector x and L_y is an appropriate loss for the label y. The hyperparameter λ is used to adjust the relative importance of features and the label in the embedding. For all the experiments in this paper discussed in Section 5, we used a fixed $\lambda = 0.5$.

3.3. Global Optimization of Weight Function Parameters

The validation metric $\hat{M}(\hat{\theta}(\alpha))$ may have multiple optima as a function of α. In order to find the maximum validation score, the sampled candidate α's should achieve two goals. First, α should sufficiently cover the weighting parameter space. In addition, we also need a large number of candidate α's sampled near the most promising local optima. In other words, there is an exploration (spread α's more evenly) and exploitation (make α's closer to the optima) trade-off when choosing candidate α's.

One can treat this as a global optimization problem and sample candidate α's with a derivative free optimization algorithm (Conn et al., 2009), such as simulated annealing (van Laarhoven & Aarts, 1987), particle swarm optimization (Kennedy & Eberhart, 1995) and differential evolution (Storn & Price, 1997). For a low dimensional space, one can do an exhaustive search for α's on a grids. We derive a generalization bound for such a search in Theorem 2.

For the experiments in this paper, we chose to base our algorithm on Gaussian process regression (GPR), specifically on the Gaussian Process Upper-Confidence-Bound (GP-UCB) (Auer, 2002; Auer et al., 2002) adapted to batched sampling of parameters, i.e., GP-BUCB. Desautels et al. (2014) shows that GP-BUCB could achieve the same cumulative regret as GP-UCB up to a constant factor.

As detailed in Algorithm 2, after getting the i-th batch of candidate α's and their corresponding validation metrics $\hat{M}(\hat{\theta})$, we build a GPR model $g(\alpha)$ to fit the validation metrics on α for all previous observations. The next batch of candidate α's is then selected sequentially: we first sample an α_1^{i+1} based on the upper bound of the $p\%$ prediction interval of $g(\alpha)$, i.e., $\alpha_1^{i+1} = \arg\max_\alpha Q_{(50+p/2)\%}[g(\alpha)]$. A larger value of hyperparameter p encourages exploration, whereas a smaller value encourages exploitation. After α_1^{i+1} is sampled, we refit a GPR model with an added observation for α_1^{i+1}, as if we have observed a validation metric $Q_{(50-q/2)\%}[g(\alpha_1^{i+1})]$, which is the lower bound of the $q\%$ prediction interval of $g(\alpha)$. Hyperparameter q controls how much the refitted GPR model trusts the old GPR model and a larger q encourages wider exploration within each batch. We then use the refitted GPR model to generate another candidate α_2^{i+1}, and continue this process until all the candidate α's in the $(i+1)$-th batch are generated. Note that to ensure convergence, in practice, we usually generate candidate α's within a bounded domain $\mathbb{B}$.

Algorithm 2 Get Candidate α^{i+1}

$\mathcal{S} \leftarrow$ set of α and validation metrics in prior batches
$K \in \mathbb{N}_+ \leftarrow$ number of candidates in a batch
$\mathbb{B} \leftarrow$ convex domain in which to sample α
$p, q \in [0, 100] \leftarrow$ explore-exploit hyperparameters
for $j = 1, 2, \ldots, K$ **do**
  $g(\alpha) \leftarrow \text{GPR}(\mathcal{S})$
  $\alpha_j^{i+1} \leftarrow \arg\max_{\alpha \in \mathbb{B}} \{Q_{(50+p/2)\%}[g(\alpha)]\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\alpha_j^{i+1}, Q_{(50-q/2)\%}[g(\alpha_j^{i+1})])\}$
end for
$\alpha^{i+1} \leftarrow \{\alpha_1^{i+1}, \ldots, \alpha_K^{i+1}\}$

Figure 2 shows 200 sampled candidate α's in $B = 10$ batches in the example in Section 5.4 with $p = q = 68.3$. It shows that at the beginning of the candidate α sampling process, GPR explores more evenly across the domain $\mathbb{B}$. As the sampling process continues, GPR begins to exploit more heavily near the optimal $\hat{\alpha}$ in the lower right corner.

4. Theoretical Analysis

In this section, we first show that example-weighting can indeed find the optimal model for a large family of evaluation metrics. We then provide a generalization bound for the MOEW approach that learns an example-weighting function for a general test metric. All proofs are in the appendix.

Let $\mathcal{D}$ and $\mathcal{D}'$ denote the test and train distributions respectively. We will present our results for a fairly broad family of evaluation metrics $M(\theta)$ that can be written as a general function $\Psi : \mathbb{R}^K \to \mathbb{R}$ of K simpler evaluation metrics defined on the *test* distribution:

$$M(\theta) \quad := \quad \Psi(\phi_1(\theta), \ldots, \phi_K(\theta)), \tag{4}$$

where:

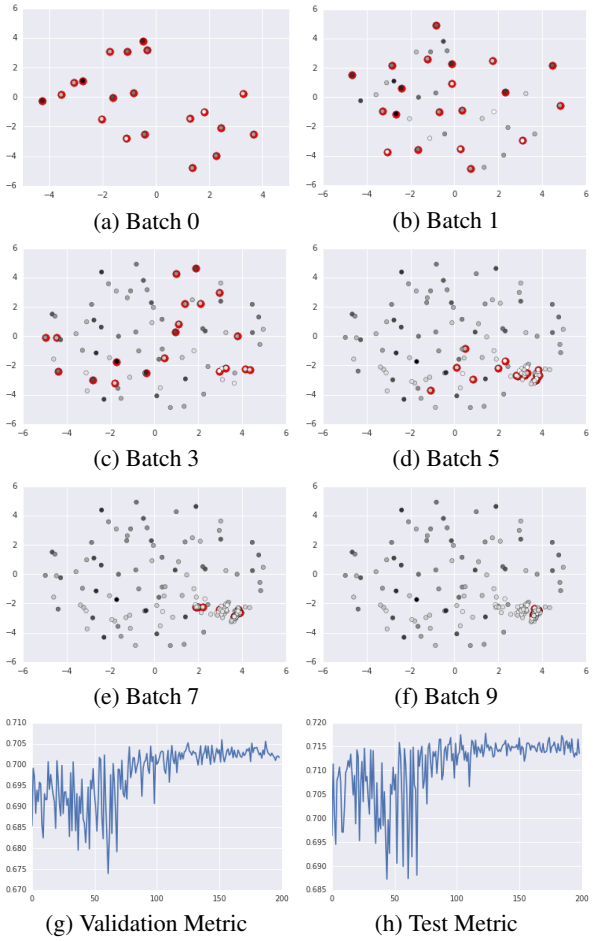
$$\phi_k(\theta) \quad := \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell_k\big(x, y; \theta\big)\big],$$

with each $\ell_k : \mathbb{R}^D \times \mathcal{Y} \to \mathbb{R}_+$ being a loss on individual examples. This class of metrics includes the F-measure, various fairness metrics and general functions of the confusion matrix of a classifier. In fact, similar functional forms have been used previously to model arbitrary functions on a set of points (Zaheer et al., 2017).

As a simple example, to write the *precision* of a classifier in this form, we can set:

$$\begin{aligned}
\ell_1(x, y; \theta) \quad &:= \quad \mathbf{1}(h(x;\theta) = 1, y = 0) \\
\ell_2(x, y; \theta) \quad &:= \quad \mathbf{1}(h(x;\theta) = 0, y = 1) \\
\Psi(\phi_1, \phi_2) \quad &:= \quad (\mathbb{E}[y=1] - \phi_2)/(\mathbb{E}[y=1] + \phi_1 - \phi_2).
\end{aligned}$$

We first generalize results from (Narasimhan et al., 2015a;b) to show that under specific assumptions on ℓ_k the optimal model for the above test metrics can be found by minimizing an example-weighted loss on the *train distribution*.



(a) Batch 0 (b) Batch 1 (c) Batch 3 (d) Batch 5 (e) Batch 7 (f) Batch 9 (g) Validation Metric (h) Test Metric

Figure 2: In Figures 2a-2f, red dots show the first two embedding dimensions of candidate α's throughout the $B = 10$ batches GPR sampling process in the example studied in Section 5.4. Figures 2g and 2h show the validation and test metrics with the 200 sampled candidate α. The correlation between validation (g) and test (h) metrics is 0.97.

Theorem 1 (Connection between complex metrics and example-weighting). *Let θ^* be the optimal model parameters for $M(\theta)$. Let the train and test distributions $\mathcal{D}$ and $\mathcal{D}'$ have the same support. Let each ϕ_k be strictly convex in θ and let each $\ell_k\big(x, y; \theta\big) = \big(\varphi_k(x, y)\big)^\top L\big(h(x;\theta), y\big)$ for weight functions $\varphi_1, \ldots, \varphi_K : \mathbb{R}^D \times \mathcal{Y} \to \mathbb{R}_+^n$ and a loss function $L : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}_+^n$. Let Ψ belong to **one** of the following two families:*

1. *$\Psi(z)$ is concave in z, and is strictly decreasing in z_i.*
2. *$\Psi(z) = \Psi'(z)/\Psi''(z)$ where $\Psi' : \mathbb{R}^K \to \mathbb{R}_+$ is concave, $\Psi'' : \mathbb{R}^K \to \mathbb{R}_+$ is convex, $\Psi''(z) > 0\ \forall z \in \mathbb{R}^K$, and $\Psi'(z) - M(\theta^*)\Psi''(z)$ is strictly decreasing in each z_i.*

Then there exists a weight function $w : \mathbb{R}^D \times \mathcal{Y} \to \mathbb{R}^n_+$ such that θ^* is the unique solution for the following weighted loss minimization on the train distribution:

$$\min_{\theta \in \mathbb{R}^D} \mathbb{E}_{(x,y)\sim\mathcal{D}'} \Big[\big(w(x,y)\big)^\top L\big(h(x;\theta), y\big) \Big].$$

The above theorem shows an equivalence between *maximizing* a metric M of the prescribed form, and *minimizing* an example-weighted loss. The monotonicity assumption on Ψ ensures that that the lower the individual losses ϕ_k, the higher is the metric.

To better understand the effect of the embedding dimension and the size of the validation set on the generalization error of the MOEW method, we consider a theoretical analysis of an instance of MOEW where the *GetCandidates* algorithm does an exhaustive search over a fixed candidate set $\mathcal{A}$ that covers the unit ball.

Theorem 2 (Generalization bound). *Let $\alpha \in \mathbb{B}^d$ be the MOEW coefficient vector in a d-dimensional unit ball. Let $\mathcal{V} \sim \mathcal{D}^N$ be a validation set of size N. Denote the empirical versions of the test metric in equation 4 as:*

$$\begin{aligned} \hat{M}(\theta) &\coloneqq \Psi(\hat{\phi}_1(\theta), \ldots, \hat{\phi}_K(\theta)), \\ \hat{\phi}_k(\theta) &\coloneqq \frac{1}{N} \sum_{(x,y)\in\mathcal{V}} \ell_k(x, y; \theta). \end{aligned}$$

Let $\alpha^* \coloneqq \arg\max_{\alpha\in\mathbb{B}^d} M(\hat{\theta}(\alpha))$ be the optimal coefficient vector in the unit ball, and $\hat{\alpha} \coloneqq \arg\max_{\alpha\in\mathcal{A}} \hat{M}(\hat{\theta}(\alpha))$ be the empirically optimal among a candidate set $\mathcal{A}$. Assume:

(A) For all k, ℓ_k is sub-Gaussian with parameter σ.
(B) Ψ is L_Ψ-Lipschitz continuous in ϕ's w.r.t. the L_2 norm.
(C) For all k, $\phi_k(\hat{\theta}(\alpha))$ is L_ϕ-Lipschitz continuous in α w.r.t. the L_2 norm.

For $N \geq 9\sigma^2 K$, there is a candidate set $\mathcal{A}$ such that with probability $1-\delta$, $M(\hat{\theta}(\alpha^*)) - M(\hat{\theta}(\hat{\alpha}))$ is bounded by:

$$\frac{\sigma L_\Psi \sqrt{K}}{\sqrt{N}} \left(\sqrt{4d \ln \frac{N}{\sigma^2} + 8 \ln \frac{2K}{\delta}} + 3L_\phi \right).$$

Theorem 2 establishes that an exhaustive search in the unit ball finds a solution that approaches the optimal test metric in order $\tilde{O}(\sqrt{d \log N / N})$. Note that the Lipschitz constant L_ϕ depends on the metric function, model hypothesis space, training loss function, training algorithm and the size of the training set. A more concrete bound can possibly be established with convexity assumption on the loss function and well-behaved hypothesis spaces.

5. Experimental Results

In this section, we illustrate the value of our proposal by comparing it to common strategies on a diverse set of example problems. Unless otherwise noted, for our proposal, we first create a d-dimensional embedding of training pairs $\mathcal{T}$ by training an autoencoder that has d nodes in the middle layer. We sample $B \times K$ candidate α's in a d-dimensional ball of radius R using GP-BUCB with $p = q = 68.3$ and an RBF kernel, whose kernel width was set to be equal to R. The noise level of the GP was determined based on the metric noise level of uniform weighting models. For a fair comparison, for competing methods, we also train the same number of models (with random initialization), and pick the one with the best validation metric. Both the autoencoder and the main models were trained for 10k steps using Adam optimizer (Kingma & Ba, 2015) with learning rate 0.001. We used squared loss for numeric, hinge loss for binary, and cross-entropy loss for multiclass label/features. To mitigate the randomness in the result, we repeat the whole process 100 times and report the average and error margin[1].

Our experiments are designed to test whether MOEW can help, irregardless of the model structure. Therefore, for simplicity, model architecture and hyperparameters were optimized for each experiment without any weights, then fixed for the weighting experiments. In practice, one might get some additional gains by tuning MOEW and model hyperparameters jointly.

The ability to optimize *any testing metric* is a unique benefit of MOEW over other metric-specific methods. For that reason we focus on demonstrating it does help optimize non-standard metrics. How MOEW compares to customized losses for optimizing specific testing metrics, such as AUC or F-score, is not the focus of this paper, but would be interesting follow-on work.

5.1. MOEW Performance Versus Model Complexity

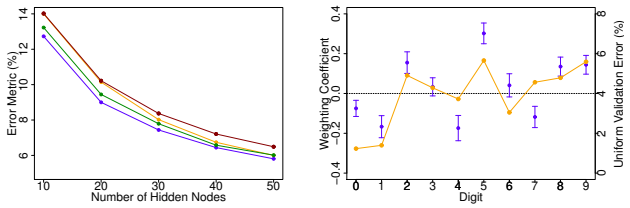
In this section we empirically examine the performance of MOEW in relation to the complexity of the main model and the weighting model.

5.1.1. Main Model Complexity

If the model is flexible enough, and the training data is clean and sufficient to fit a perfectly accurate model for all parts of the feature space, then MOEW is not needed. MOEW will be most valuable when the model must take some trade-off, and we can learn an example weighting in a way that best benefits the test metric. We illustrate this effect by studying the efficacy of MOEW as a function of the complexity of the main model. For this experiment, we used MNIST handwritten digit database (LeCun & Cortes, 2010), and train on it with classifiers of varying complexity.

We used training/validation/test split of sizes 55k/5k/10k

[1]The code on public datasets is available at the following GitHub address: https://github.com/google-research/google-research/tree/master/moew.



(a) Number of Hidden Nodes (b) Analysis of Digits

Figure 3: (a) MNIST test metric with varying number of hidden nodes in the main model. Methods include uniform weighting (orange), purely random weighting (red), MOEW with random α's (green) and with GP-BUCB selected α's (blue). The 95% error margin for any point is less than 0.1%. (b) With $h = 50$ hidden nodes in the main model, average α of each digit and their 95% error margin for GP-BUCB (blue, axis on the left), as well as the validation error of uniform weighting model (orange, axis on the right).

respectively. For the MOEW model, to simplify analysis, instead of using an autoencoder to learn the embedding, we directly used the class label as the 10-dimensional embedding. The main classifier was a sigmoid-activation network with $784 \to h \to 10$ nodes in each layer, with $h \in \{10, 20, 30, 40, 50\}$. We used $B = 10$ batches of $K = 20$ α's, which were either generated randomly or based on GP-BUCB, and compared against the best-of-200 uniform and purely random weighted models. The error metric was taken to be the maximum of the error rates for each digit: $\max_{i\in\{0,\ldots,9\}} \Pr\{\hat{y} \neq i | y = i\}$.

Figure 3a shows the test error metric calculated with various main model complexities, each averaged over 100 runs. We observe that our proposal (blue and green) clearly outperforms uniform weighting (orange) and purely random weighting (red) for models of limited complexity. The benefit is smaller for models that are more flexible. In most real-world situations, the model complexity is limited either by practical constraints on computation, or otherwise to avoid overfitting to smaller training datasets. In such cases there might be an inherent trade-off in the learning process, and we expect MOEW to deliver the most improvement. In addition to the above finding, we observe that candidate α's generated based on GP-BUCB delivers better performance than those generated randomly.

Figure 3b shows the average weighting coefficients (i.e., α in equation 3) of the ten digits generated by GP-BUCB with $h = 50$ hidden nodes in the main model (blue). It indicates that MOEW learns to upweight digits 2, 5, 8 and 9, and downweight 0, 1, 4 and 7. Such a result is consistent with the validation error in the uniform weighting model (orange), where digits 2, 5, 8 and 9 have the highest error.

5.1.2. MOEW Model Complexity and Risk of Overfitting

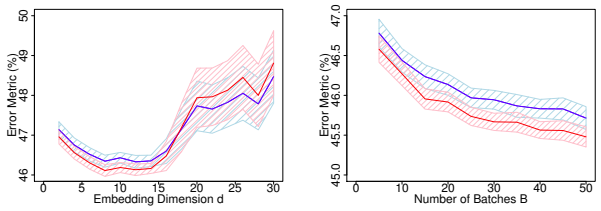
In this experiment, we study the effect of the embedding dimension d, the number of batches B and the exploration-exploitation parameters p and q on the MOEW performance. Specifically, we are interested in the difficulty for GP-BUCB to find the optimal solution, as well as the risk of overfitting to the validation dataset. We used the wine reviews dataset from Kaggle (www.kaggle.com/zynicide/wine-reviews). The task is to predict the price of the wine using 39 Boolean features describing characteristic of the wine and the quality score (points), for a total of 40 features. We calculate the error in percentage of the correct price, and want the model to have good accuracy across all price ranges. To that end, we set the test metric to be the worst of the errors among 4 quartiles $\{q_i\}$ of the price (thresholds are 17, 25 and 42 dollars): $\max_{i\in\{0,\ldots,3\}} E_{\text{price}(x)\in q_i}[|\hat{y}/y - 1|]$.

We used training/validation/test split of sizes 85k/12k/24k respectively. We applied a log transformation to the label (price) and used mean squared error as the training loss on the log-transformed prices (for all weightings). For MOEW, we used a d-dimensional embedding, created by training a sigmoid-activation autoencoder network on the (x, y) pair, with $41 \to 100 \to d \to 100 \to 41$ nodes in each layer. The main regressor was a sigmoid-activation network with $40 \to 20 \to 10 \to 1$ nodes in each layer. We used B batches of $K = 20$ α's in our proposed method.

In the first study, shown in figure 4a, we fixed the number of batches $B = 10$ and varied the embedding dimension $d \in \{2, 4, \ldots, 30\}$. The result indicates that the test performance of MOEW improves as we increase the embedding dimension up to around $d = 8$, at which point the test error metric is 46.33 ± 0.16. As a comparison, the best-of-200 uniform weighted models achieves an average test error metric of 52.00 ± 0.31. On the other hand, the result also suggests that the validation performance begins to drop when $d > 14$, which indicates that GP-BUCB was unable to converge to good solutions in such high-dimensional spaces with 200 candidate α's. In addition, the test-metric-to-validation-metric-ratio is slightly larger when d is small, which suggests that there might be overfitting to the validation set with *small* embedding dimension d. This is in fact intuitive: in a high-dimensional space with a limited number of α's, GP-BUCB acts similarly to pure exploration. Because there is less exploitation, there is also less overfitting.

In the second study, shown in figure 4b, we fixed the embedding dimension $d = 10$ and investigated the performance of MOEW with number of batches $B \in \{5, 10, \ldots, 50\}$. The figure indicates that both the validation and test performance improves as we sample more batches of candidate weighting parameters. In addition, the gap between validation and test metrics does not widen, suggesting that the risk of overfit-

ting is small in the range of values we experimented with. As a comparison with the $B = 50$ case, the best-of-1000 uniform model achieves test performance 48.27 ± 0.41.



(a) Embedding Dimension d (b) Number of Batches B

Figure 4: Metrics for validation (red) and test (blue) on wine data with (a) embedding dimension $d \in \{2, 4, \ldots, 30\}$ and number of batches $B = 10$, and (b) $d = 10$ and $B \in \{5, 10, \ldots, 50\}$. The pink and light blue shaded areas indicate the 95% confidence bands.

In the third study, we fixed the embedding dimension $d = 10$ and number of batches $B = 10$, and study the effect of p and q on MOEW performance. When $p = q = 0$, 75, 95, 99.99 and 99.999, MOEW test errors are 47.07 ± 0.19, 46.32 ± 0.15, 46.28 ± 0.15, 46.28 ± 0.16 and 46.46 ± 0.17, respectively. MOEW is insensitive to p or q if exploration is reasonably large.

5.2. MOEW Performance on Small Data

In this example, we examine MOEW on a small dataset, namely the Communities and Crime dataset from the UCI Machine Learning Repository (Dheeru & Karra Taniskidou, 2017), which contains the violent crime rate of 1994 communities. The goal is to predict whether a community has violent crime rate per 100k population above 0.28.

In addition to obtaining an accurate classifier, we also aim to improve its *fairness*. To this end, we divided the communities into 4 groups based on the quartiles of white population percentage in each community (thresholds are 63%, 84% and 94%). We seek a classifier with high accuracy, but that has similar false positive rates (FPR) across racial groups. Therefore, we evaluate classifiers based on two metrics: overall accuracy across all communities and the difference between the highest and lowest FPR across four racial groups (fairness violation).

We used a linear classifier with 95 features, including the percentage of African American, Asian, Hispanic and white population. For MOEW, those features and the label were projected to a 4-dimensional space using an autoencoder with $96 \to 10 \to 4 \to 10 \to 96$ nodes. We sampled candidate α's in $B = 10$ batches of size $K = 5$.

In practice, there is usually a trade-off between accuracy and fairness (see, e.g., Hardt et al., 2016; Goh et al., 2016). To explore this trade-off, we consider two approaches: with the vanilla approach, we used MOEW to minimize the difference between the highest and lowest FPR across the 4 groups (i.e., minimize fairness violation), with identical decision thresholds for all groups. With the post-shift approach (Hardt et al., 2016), after training the model, we used MOEW to maximize accuracy with separate decision thresholds for each group to achieve the same FPR on the training data while maintaining the same overall coverage.

Table 1: Average accuracy and fairness violation with their 95% error margin with IID training/test split. We consider (Vanilla) minimizing fairness violation with identical decision threshold across racial groups, and (Post-Shift) maximizing accuracy with racial-group-specific thresholds for equal FPR on the training data.

	Vanilla Approach	
	Uniform	MOEW
Accuracy (%)	86.82 ± 0.07	86.84 ± 0.12
Fairness Violation (%)	79.23 ± 2.35	57.74 ± 4.64
	Post-Shift Approach	
	Uniform	MOEW
Accuracy (%)	79.12 ± 0.12	79.95 ± 0.17
Fairness Violation (%)	12.26 ± 0.30	10.06 ± 0.54

In the first study, we sample 994/500/500 training/validation/testing examples purely randomly, and compare MOEW with the best-of-50 uniform weighting model. The results are summarized in Table 1. With the vanilla approach, MOEW reduces fairness violation by over 20% and yet achieves the same accuracy compared to uniform weighting. With the post-shift approach, MOEW improves accuracy and reduces fairness violation at the same time.

In the second study, we sample 994/500/500 training/validation/testing examples such that in the training data, 76% of the communities have above median population, whereas in the validation/testing set, 40% of the communities have above median population. We compare MOEW with the best-of-50 optimal domain adaptation training weights provided by Shimodaira (2000). The results are summarized in Table 2. MOEW performed similarly to Shimodaira (2000) with the vanilla approach, and better in fairness violation with post-shift.

5.3. Spam Blocking

For this problem from a large internet services company, the goal is to classify whether a result is spam, and this decision affects whether the result provider receives ads revenue from the company. Thus, it is more important to block more expensive spam results, but it is also important not to block any results that are not spam, especially results

Table 2: Average accuracy and fairness violation with their 95% error margin with non-IID training/test split.

	Vanilla Approach	
	Shimodaira	MOEW
Accuracy (%)	85.59 ± 0.05	85.59 ± 0.06
Fairness Violation (%)	37.53 ± 0.40	37.78 ± 0.39
	Post-Shift Approach	
	Shimodaira	MOEW
Accuracy (%)	72.46 ± 0.11	72.33 ± 0.12
Fairness Violation (%)	6.63 ± 0.44	4.61 ± 0.41

Table 3: Average test metric and 95% error margin for Spam Blocking with models trained with uniform weighting, optimal domain adaptation weighting (Shimodaira) and MOEW. Larger test metric is better.

	Study 1	Study 2
Uniform	1.000 ± 0.040	1.000 ± 0.124
Shimodaira	1.210 ± 0.063	1.010 ± 0.137
MOEW	1.849 ± 0.133	8.057 ± 0.923

with many impressions. We used a simplified test metric that captures these different goals (the actual metric is more complex and proprietary). Specifically, for each method we set the classifier decision threshold so that 5% of the validation set is labelled as spam. We then sum the costs saved by blocking correctly identified spams and divide it by the total number of blocked impressions of incorrectly-identified spams.

The datasets contain 12 features. We trained an autoencoder on the 12 features plus label, with layers of $13 \rightarrow 100 \rightarrow 3 \rightarrow 100 \rightarrow 13$ nodes, and used the middle layer as an embedding. For each weighting method, we built a sigmoid-activation network classifier with architecture $12 \rightarrow 20 \rightarrow 10 \rightarrow 1$. Candidate α's were sampled $K = 20$ at a time in $B = 10$ rounds of sampling.

In the first study, we divide the dataset such that the 180k training dataset is 25% spam, and is not IID with the validation/test datasets, which are IID and have 10k/30k examples with 5% spam. In the second study, we divide the dataset such that half of examples are from large result providers in the 180k training dataset, and 20% of examples are from large result providers in the 10k/30k validation/test datasets.

Table 3 compares MOEW with two common choices in practice: uniform weighting and optimal domain adaptation weighting (Shimodaira, 2000). We have normalized the reported scores so that the average uniformly weighted test metric is 1.0. Our proposed method clearly outperforms both uniform and optimal domain adaptation weighting.

5.4. Web Page Quality

This binary classifier example is from a large internet services company. The goal is to identify high quality webpages, thresholded such that 40% of examples are classified as high quality. The company performed several rounds of human ratings of example web pages. In the early rounds, the label was binary (high/low quality). Later, human raters provided finer-grained labels, scoring the quality in $[0, 1]$. The test metric for this problem is the mean fine-grained score of the positively classified test examples.

We trained a six-feature sigmoid-activation classifier with $6 \rightarrow 20 \rightarrow 10 \rightarrow 1$ nodes on the 62k examples with binary labels. To fit MOEW, we trained an autoencoder that mapped the six features plus label onto a 3-dimensional space: $7 \rightarrow 100 \rightarrow 3 \rightarrow 100 \rightarrow 7$. We sampled $K = 20$ candidate α's for $B = 10$ rounds.

The validation data and test data each have 10k web pages labeled with a fine-grained score in $[0, 1]$. The datasets are not IID: the training data is the oldest data, the test data is the newest data, with validation data in-between. The average quality score of 40% selected web pages is 0.7113 ± 0.0004 with uniform weighting, and 0.7176 ± 0.0004 with MOEW. Note that in this example, domain adaptation weighting results in uniform weighting.

6. Conclusions

We proposed learning example weights to sculpt a standard loss function into one that better optimizes a test metric for the test distribution. We demonstrated substantial benefits on public benchmark datasets and real-world applications, for problems with non-IID train/test distributions and custom metrics that incorporated multiple objectives.

To limit the amount of validation data needed, we define the example weighting model over a low-dimensional space. Our experiments used the low-dimensional embedding of (x, y) produced by an autoencoder, but we hypothesize that a discriminatively-trained embedding could be more optimal, or that using a small subset of semantically-meaningful features could be more interpretable.

We hypothesize that the MEOW could also be useful for other purposes. For example, they may be useful for guiding active sampling, suggesting one should sample more examples in feature regions with high training weights. And we hypothesize one could downsample areas of low-weight to reduce the size of training data for faster training and iterations, without sacrificing test performance.

References

Agarwal, A., Beygelzimer, A., Dudik, M., Langford, J., and Wallach, H. A reductions approach to fair classification. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 60–69, 2018.

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Bickel, S., Brückner, M., and Scheffer, T. Discriminative learning under covariate shift. *Journal of Machine Learning Research*, 10:2137–2155, 2009.

Conn, A. R., Scheinberg, K., and Vicente, L. N. *Introduction to Derivative-Free Optimization*. SIAM, 2009.

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

Davis, J. and Goadrich, M. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233–240, 2006.

Desautels, T., Krause, A., and Burdick, J. W. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923, 2014.

Dheeru, D. and Karra Taniskidou, E. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Eban, E., Schain, M., Mackey, A., Gordon, A., Rifkin, R., and Elidan, G. Scalable learning of non-decomposable objectives. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pp. 832–840, 2017.

Ferri, C., Flach, P. A., and Hernández-Orallo, J. Learning decision trees using the area under the ROC curve. In *Proceedings of the 19th International Conference on Machine Learning*, pp. 139–146, 2002.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

Goh, G., Cotter, A., Gupta, M., and Friedlander, M. P. Satisfying real-world goals with dataset constraints. In *Advances in Neural Information Processing Systems*, pp. 2415–2423, 2016.

Hardt, M., Price, E., and Srebro, N. Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems*, pp. 3315–3323, 2016.

Herschtal, A. and Raskutti, B. Optimising area under the roc curve using gradient descent. In *Proceedings of the 21st International Conference on Machine Learning*, pp. 49–56, 2004.

Jansche, M. Maximum expected F-measure training of logistic regression models. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 692–699, 2005.

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

Kanamori, T., Hido, S., and Sugiyama, M. A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10:1391–1445, 2009.

Kar, P., Narasimhan, H., and Jain, P. Online and stochastic gradient methods for non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, pp. 694–702, 2014.

Kar, P., Narasimhan, H., and Jain, P. Surrogate functions for maximizing precision at the top. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 189–198, 2015.

Kearns, M., Neel, S., Roth, A., and Wu, Z. S. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 2564–2572, 2018.

Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Koyejo, O. O., Natarajan, N., Ravikumar, P. K., and Dhillon, I. S. Consistent binary classification with generalized performance metrics. In *Advances in Neural Information Processing Systems*, pp. 2744–2752, 2014.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Lunceford, J. K. and Davidian, M. Stratification and weighting via the propensity score in estimation of causal treatment effects: a comparative study. *Statistics in Medicine*, 23(19):2937–2960, 2004.

Narasimhan, H. Learning with complex loss functions and constraints. In *International Conference on Artificial Intelligence and Statistics*, pp. 1646–1654, 2018.

Narasimhan, H., Vaish, R., and Agarwal, S. On the statistical consistency of plug-in classifiers for nondecomposable performance measures. In *Advances in Neural Information Processing Systems*, pp. 1493–1501, 2014.

Narasimhan, H., Kar, P., and Jain, P. Optimizing nondecomposable performance measures: A tale of two classes. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 199–208, 2015a.

Narasimhan, H., Ramaswamy, H., Saha, A., and Agarwal, S. Consistent multiclass algorithms for complex performance measures. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2398–2407, 2015b.

Parambath, S. P., Usunier, N., and Grandvalet, Y. Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems*, pp. 2123–2131. 2014.

Perlich, C., Provost, F., and Simonoff, J. S. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4:211–255, 2003.

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.

Shimodaira, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

Storn, R. and Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

Sugiyama, M., Krauledat, M., and Müller, K.-R. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8:985–1005, 2007.

Sugiyama, M., Nakajima, S., Kashima, H., Büenau, P. V., and Kawanabe, M. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in Neural Information Processing Systems*, pp. 1433–1440, 2008.

van Laarhoven, P. J. M. and Aarts, E. H. L. *Simulated Annealing: Theory and Applications*. Springer, 1987.

Yan, L., Dodier, R., Mozer, M. C., and Wolniewicz, R. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *Proceedings of the 20th International Conference on Machine Learning*, pp. 848–855, 2003.

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 271–278, 2007.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in Neural Information Processing Systems*, pp. 3391–3401, 2017.

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