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., 2009).
We are interested in learning a model, We consider classification and regression problems where We now describe our proposed adaptive example weighting respectivly. We assume that \( T \) and \( V \) may not be IID. The goal is to use the training and validation sets to find model \( \theta^* \) 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 \( \theta \) 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 \( w : \mathbb{R}^D \times \mathbb{R} \rightarrow \mathbb{R}_+ \) parameterized by \( \alpha \) 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

\[
\hat{\theta}(\alpha) = \arg \min_{\theta} \sum_{j \in T} w(x_j, y_j; \alpha) L(h(x_j; \theta), y_j). \tag{1}
\]

Let \( \hat{M}(x_V, y_V; \theta) \) denote the estimate of the test metric on the validation dataset \( V \). Indeed one could consider directly optimizing \( \hat{M}(x_V, y_V; \hat{\theta}(\alpha)) \) over \( \theta \). However, for a high-dimensional \( \theta \) and a small \( V \), 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: 

\[
\hat{\alpha} = \arg \max_{\alpha} \hat{M}(x_V, y_V; \hat{\theta}(\alpha)). \tag{2}
\]

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 \( \alpha \) can be constructed to be low-dimensional, which can be optimized to maximize \( M \) on the validation set, whereas \( \theta \), being high-dimensional, is optimized on the larger training set.

To simplify the notation, where possible, we denote \( \theta(\alpha) \) by \( \theta \) and \( \hat{M}(x_V, y_V; \theta) \) by \( \hat{M}(\theta) \). The validation metric \( \hat{M}(\theta) \) as a function of \( \theta \) and \( \alpha \) 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 \( \theta^* \) 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 \( \alpha_l^0 \), \( 1 \leq l \leq K \), we solve equation 1 to obtain \( K \) corresponding model parameters \( \theta_l^0 \), and use those to compute \( K \) validation metrics, \( \hat{M}(\theta_l^0) \). 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
While there are many ways to form a low-dimensional em-
cx
x
z
z
π
denote the probability density function of
function
π
discussed in Section 5, we used the standard importance
z
T
over (a batch from) the training set
paper, we chose the functional form:

\[
\begin{align*}
&\text{Algorithm 1 Get optimal } \hat{\alpha} \text{ and } \hat{\theta}(\hat{\alpha}) \\
&T \leftarrow \text{training data} \\
&V \leftarrow \text{validation data} \\
&B \in \mathbb{N}_+ \leftarrow \text{number of batches of weight parameters} \\
&\alpha^0 = \{\alpha_1^0, \ldots, \alpha_K^0\} \leftarrow \text{initial weight parameters} \\
&\text{for } i = 0, 1, \ldots, B - 1 \text{ do} \\
&\quad \text{for all } l \in \{1, \ldots, K\} \text{ do} \\
&\quad\quad \theta_l^i \leftarrow \arg\min_{\theta} \sum_{j \in T} w(x_j, y_j; \theta) L(h(x_j; \theta), y_j) \\
&\quad \text{end for} \\
&\quad s_l^i \leftarrow \{\alpha_1^i, \hat{M}(\hat{\theta}_1^i), \ldots, \alpha_K^i, \hat{M}(\hat{\theta}_K^i)\} \\
&\quad \alpha^{i+1} \leftarrow \text{GetCandidateAlphas}(s_0^i, \ldots, s_l^i) \\
&\quad \text{end for} \\
&\quad \hat{b}, \hat{k} \leftarrow \arg\max_{\hat{b}, \hat{k}} \{0, \ldots, B-1\} \{1, \ldots, K\} \hat{M}(\hat{\theta}_K^i) \\
&\quad \hat{\alpha}, \hat{\theta} \leftarrow \alpha_{\hat{b}}^i, \hat{\theta}_{\hat{k}}^i \\
&\text{end for} \\
&\text{end for} \\
&\text{subroutine GetCandidateAlphas. The process is repeated for } B \text{ iterations. At the end, we choose the candidate } \alpha \text{ that produced the best validation metric.}
\end{align*}
\]

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

\subsection{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 \( \alpha \)'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
W with a small number of parameters.

There are many reasonable strategies for defining \( W \). In this paper, we chose the functional form:

\[
w(x, y; \alpha) = c \pi(y) \sigma(z(x, y)^\top \alpha),
\]

where \( c \in \mathbb{R}_+ \) is a constant that normalizes the weights over (a batch from) the training set \( T \), 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 \( \pi(y) = p_V(y) / p_T(y) \), where \( p_V(y) \) and \( p_T(y) \) 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 \( z(x, y; \xi) \), we minimize the weighted sum of the reconstruction loss for \( x \) and \( y \):

\[
\sum_{(x, y) \in T} \{ \lambda L_x(z(x, y; \xi), x) + (1 - \lambda) L_y(z(x, y; \xi), y) \},
\]

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 \( \lambda \) 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 \( \lambda = 0.5 \).

\subsection{3.3. Global Optimization of Weight Function Parameters}

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

One can treat this as a global optimization problem and sample candidate \( \alpha \)'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 \( \alpha \)'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 algo-
rithm 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 \( \alpha \)'s and their corresponding validation metrics \( \hat{M}(\hat{\theta}) \), we build a GPR model \( g(\alpha) \) to fit the validation metrics on \( \alpha \) for all previous observations. The next batch of candidate \( \alpha \)'s is then selected sequentially: we first sample an \( \alpha_{i+1}^0 \) based on the upper bound of the \( p \% \) prediction interval of \( g(\alpha) \), i.e., \( \alpha_{i+1}^0 = \arg\max_{\alpha} Q_{(50+p/2)\%}[g(\alpha)] \).

A larger value of hyperparameter \( p \) encourages exploration, whereas a smaller value encourages exploitation. After \( \alpha_{i+1} \) is sampled, we refit a GPR model with an added observation for \( \alpha_{i+1} \), as if we have observed a validation metric \( Q_{(50-q/2)\%}[g(\alpha_{i+1})] \), 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 \( \alpha_{i+1} \), and continue this process until all the candidate \( \alpha \)'s in the \( (i+1) \)-th batch are generated. Note that to ensure convergence, in practice, we usually generate candidate \( \alpha \)'s within a bounded domain \( B \).
Algorithm 2 Get Candidate $\alpha^{i+1}$

\[
S \leftarrow \text{set of } \alpha \text{ and validation metrics in prior batches}
K \leftarrow \text{number of candidates in a batch}
P \leftarrow \text{convex domain in which to sample } \alpha
p, q \in [0, 100] \leftarrow \text{explore-exploit hyperparameters}
\]

for $j = 1, 2, \ldots, K$ do

\[
g(\alpha) \leftarrow \text{GPR}(S)
\]

\[
\hat{\alpha}_j^{i+1} \leftarrow \arg \max_{\alpha \in P} \{Q(50 + p/2) \% g(\alpha)\}
\]

\[
S \leftarrow S \cup \{ (\hat{\alpha}_j^{i+1}, Q(50 - q/2) \% g(\hat{\alpha}_j^{i+1})) \}
\]

end for

\[
\alpha^{i+1} \leftarrow \{ \alpha_1^{i+1}, \ldots, \alpha_K^{i+1} \}
\]

Figure 2 shows 200 sampled candidate $\alpha$’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 $\alpha$ sampling process, GPR explores more evenly across the domain $P$. 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 $D$ and $D'$ 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 $\Psi : \mathbb{R}^K \rightarrow \mathbb{R}$ of $K$ simpler evaluation metrics defined on the test distribution:

\[
M(\theta) := \Psi(\phi_1(\theta), \ldots, \phi_K(\theta)),
\]

where

\[
\phi_k(\theta) := \mathbb{E}_{(x,y) \sim D} \left[ \ell_k(x, y; \theta) \right],
\]

with each $\ell_k : \mathbb{R}^D \times \mathcal{Y} \rightarrow \mathbb{R}_+$ 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:

\[
\ell_1(x, y; \theta) := 1(h(x; \theta) = 1, y = 0)
\]

\[
\ell_2(x, y; \theta) := 1(h(x; \theta) = 0, y = 1)
\]

\[
\Psi(\phi_1, \phi_2) := \frac{\mathbb{E}[y = 1] - \phi_2}{\mathbb{E}[y = 1] + \phi_1 - \phi_2}.
\]

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

Theorem 1 (Connection between complex metrics and example-weighting). Let $\theta^*$ be the optimal model parameters for $M(\theta)$. Let the train and test distributions $D$ and $D'$ have the same support. Let each $\phi_k$ be strictly convex in $\theta$ and let each $\ell_k(x, y; \theta) = (\varphi_k(x, y))^T L(h(x; \theta), y)$ for weight functions $\varphi_1, \ldots, \varphi_K : \mathbb{R}^D \times \mathcal{Y} \rightarrow \mathbb{R}_+$ and a loss function $L : \mathbb{R} \times \mathcal{Y} \rightarrow \mathbb{R}_+$. Let $\Psi$ 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 $\Psi' : \mathbb{R}^K \rightarrow \mathbb{R}_+$ is concave, $\Psi'' : \mathbb{R}^K \rightarrow \mathbb{R}_+$ is convex, $\Psi''(z) > 0 \forall z \in \mathbb{R}^K$, and $\Psi(z) - M(\theta^*)\Psi''(z)$ is strictly decreasing in each $z_i$. 

Figure 2: In Figures 2a-2f, red dots show the first two embedding dimensions of candidate $\alpha$’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 $\alpha$. The correlation between validation (g) and test (h) metrics is 0.97.
Then there exists a weight function $w : \mathbb{R}^d \times \mathcal{Y} \rightarrow \mathbb{R}^n$ such that $\theta^*$ is the unique solution for the following weighted loss minimization on the training distribution:

$$
\min_{\theta \in \mathbb{R}^d} E_{(x,y) \sim D} \left[ (w(x,y))^\top L(h(x; \theta), y) \right].
$$

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 $\Psi$ ensures that the lower the individual losses $\phi_k$, 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 $A$ 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:

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

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

(A) For all $k$, $\ell_k$ is sub-Gaussian with parameter $\sigma$.
(B) $\Psi$ is $L_\Psi$-Lipschitz continuous in $\phi$'s w.r.t. the $L_2$ norm.
(C) For all $k$, $\hat{\phi}_k(\hat{\theta}(\alpha))$ is $L_\phi$-Lipschitz continuous in $\alpha$ w.r.t. the $L_2$ norm.

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

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

Theorem 2 establishes that an exhaustive search in the unit ball finds a solution that approaches the optimal test metric in order $O(\sqrt{d \log N}/N)$. Note that the Lipschitz constant $L_\phi$ 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 $\alpha$’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.
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): \(\max_{i \in \{0, \ldots, 3\}} |\hat{y}_i - y|\).

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 \(41 \rightarrow 100 \rightarrow d \rightarrow 100 \rightarrow 41\) nodes in each layer. The main regressor was a sigmoid-activation network with \(40 \rightarrow 20 \rightarrow 10 \rightarrow 1\) nodes in each layer. We used \(B\) batches of \(K = 20\) \(\alpha\)'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 \(\alpha\)'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 \(\alpha\)'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 $48.27 \pm 0.41$.

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 $47.07 \pm 0.19$, 46.32 $\pm$ 0.15, 46.28 $\pm$ 0.15, 46.28 $\pm$ 0.16 and 46.46 $\pm$ 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 $\rightarrow$ 10 $\rightarrow$ 4 $\rightarrow$ 10 $\rightarrow$ 96 nodes. We sampled candidate $\alpha$’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.

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>
<thead>
<tr>
<th>Vanilla Approach</th>
<th>MOEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Accuracy (%)</td>
<td>86.82 $\pm$ 0.07</td>
</tr>
<tr>
<td>Uniform Fairness Violation (%)</td>
<td>79.23 $\pm$ 2.35</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Post-Shift Approach</th>
<th>Uniform</th>
<th>MOEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>79.12 $\pm$ 0.12</td>
<td>79.95 $\pm$ 0.17</td>
</tr>
<tr>
<td>Fairness Violation (%)</td>
<td>12.26 $\pm$ 0.30</td>
<td>10.06 $\pm$ 0.54</td>
</tr>
</tbody>
</table>

Table 2: 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.
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 $\alpha$’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 web-pages, 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 $\alpha$’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 $0.7113 \pm 0.0004$ with uniform weighting, and $0.7176 \pm 0.0004$ 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


