Anytime Representation Learning

Zhixiang (Eddie) Xu[1] XUZX@CSE.WUSTL.EDU
Matt J. Kusner[1] MKUSNER@WUSTL.EDU
Gao Huang[2] HUANG-G09@MAILS.TSINGHUA.EDU.CN
Kilian Q. Weinberger[1] KILIAN@WUSTL.EDU
[1]Washington University, One Brookings Dr., St. Louis, MO 63130 USA
[2]Tsinghua University, Beijing, China

Abstract

Evaluation cost during test-time is becoming increasingly important as many real-world applications need fast evaluation (*e.g.* web search engines, email spam filtering) or use expensive features (*e.g.* medical diagnosis). We introduce Anytime Feature Representations (AFR), a novel algorithm that explicitly addresses this trade-off in the data representation rather than in the classifier. This enables us to turn conventional classifiers, in particular Support Vector Machines, into test-time cost sensitive *anytime classifiers*—combining the advantages of anytime learning and large-margin classification.

1. Introduction

Machine learning algorithms have been successfully deployed into many real-world applications, such as web-search engines (Zheng et al., 2008; Mohan et al., 2011) and email spam filters (Weinberger et al., 2009). Traditionally, the focus of machine learning algorithms is to train classifiers with maximum accuracy—a trend that made Support Vector Machines (SVM) (Cortes & Vapnik, 1995) very popular because of their strong generalization properties. However, in large scale industrial-sized applications, it can be as important to keep the test-time CPU cost within budget. Further, in medical applications, features can correspond to costly examinations, which should only be performed when necessary (here cost may denote actual currency or patient agony). Carefully balancing this trade-off between accuracy and test-time cost introduces new challenges for machine learning.

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

Specifically, this test-time cost consists of (a) the CPU cost of evaluating a classifier and (b) the (CPU or monetary) cost of extracting corresponding features. We explicitly focus on the common scenario where the feature extraction cost is dominant and can vary drastically across different features, *e.g.* web-search ranking (Chen et al., 2012), email spam filtering (Dredze et al., 2007; Pujara et al., 2011), health-care applications (Raykar et al., 2010), image classification (Gao & Koller, 2011a).

We adopt the *anytime classification* setting (Grubb & Bagnell, 2012). Here, classifiers extract features on-demand during test-time and can be queried at any point to return the current best prediction. This may happen when the cost budget is exhausted, the classifier is believed to be sufficiently accurate or the prediction is needed urgently (*e.g.* in time-sensitive applications such as pedestrian detection (Gavrila, 2000)). Different from previous settings in budgeted learning, the cost budget is explicitly *unknown* during test-time.

Prior work addresses anytime classification primarily with additive ensembles, obtained through boosted classifiers (Viola & Jones, 2004; Grubb & Bagnell, 2011). Here, the prediction is refined through an increasing number of weak learners and can naturally be interrupted at any time to obtain the current classification estimate. Anytime adaptations of other classification algorithms where early querying of the evaluation function is not as natural—such as the popular SVM—have until now remained an open problem.

In this paper, we address this setting with a novel approach to budgeted learning. In contrast to most previous work we learn an additive anytime *representation*. During test-time, an input is mapped into a feature space with multiple stages: each stage refines the data representation and is accompanied by its own SVM classifier, but adds extra cost in terms of feature extraction. We show that the SVM classifiers and the

cost-sensitive anytime representations can be learned *jointly* in a single optimization.

Our method, Anytime Feature Representations (AFR), is the first to incorporate anytime learning into large margin classifiers—combining the benefits of both learning frameworks. On two real world benchmark data sets our *anytime* AFR out-performs or matches the performance of the Greedy Miser (Xu et al., 2012), a state-of-the-art cost-sensitive algorithm which is trained with a *known* test budget.

2. Related Work

Controlling test-time cost is often performed with classifier cascades (mostly for binary classification) (Viola & Jones, 2004; Lefakis & Fleuret, 2010; Saberian & Vasconcelos, 2010; Pujara et al., 2011; Wang & Saligrama, 2012). In these cascades, several classifiers are ordered into a sequence of stages. Each classifier can either (a) reject inputs and predict them, or (b) pass them on to the next stage. This decision is based on the current prediction of an input. The cascades can be learned with boosting (Viola & Jones, 2004; Freund & Schapire, 1995), clever sampling (Pujara et al., 2011), or can be obtained by inserting early-exits (Cambazoglu et al., 2010) into preexisting stage-wise classifiers (Friedman, 2001).

One can extend the cascade to tree-based structures to naturally incorporate decisions about feature extraction with respect to some cost budget (Xu et al., 2013; Busa-Fekete et al., 2012). Notably, Busa-Fekete et al. (2012) use a Markov decision process to construct a directed acyclic graph to select features for different instances during test-time. One limitation of these cascade and tree-structured techniques is that a cost budget must be specified prior to test-time. Gao & Koller (2011a) use locally weighted regression during test-time to predict and extract the features with maximum information gain. Different from our algorithm, their model is learned during test-time.

Saberian & Vasconcelos (2010); Chen et al. (2012); Xu et al. (2013) all learn classifiers from weak learners. Their approaches perform two separate optimizations: They first train weak learners and then re-order and re-weight them to balance their accuracy and cost. As a result, the final classifier has worse accuracy vs. cost trade-offs than our jointly optimized approach.

The Forgetron (Dekel et al., 2008) introduces a clever modification of the kernelized perceptron to stay within a pre-defined memory budget. Gao & Koller (2011b) introduce a framework to boost large-margin loss functions. Different from our work, they focus on learning a classifier and an output-coding matrix simultaneously as opposed to learning a feature representation (they use the original features), and they do not address the test-time budgeted learning scenario. Kedem et al. (2012) learn a feature representation with gradient boosted trees (Friedman, 2001)—however, with a different objective (for nearest neighbor classification) and without any cost consideration.

Grubb & Bagnell (2010) combine gradient boosting and neural networks through back-propagation. Their approach shares a similar structure with ours, as our algorithm can be regarded as a two layer neural network, where the first layer is non-linear decision trees and the second layer a large margin classifier. However, different from ours, their approach focuses on avoiding local minima and does not aim to reduce test-time cost.

3. Background

Let the training data consist of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{R}^d$ with corresponding discrete class labels $\{y_1, \ldots, y_n\} \in \{+1, -1\}$ (the extension to multi-class is straightforward and described in section 5). We assume that during test-time, features are computed *on-demand*, and each feature θ has an extraction cost $c_\theta > 0$ when it is extracted for the first time. Since feature values can be efficiently cached, subsequent usage of an already-extracted feature is free.

Our algorithm consists of two jointly integrated parts, classification and representation learning. For the former we use support vector machines (Cortes & Vapnik, 1995) and for the latter we use the Greedy Miser (Xu et al., 2012), a variant of gradient boosting (Friedman, 2001). In the following, we provide a brief overview of all three algorithms.

Support Vector Machines (SVMs). Let ϕ denote a mapping that transforms inputs $\mathbf{x}_i$ into feature vectors $\phi(\mathbf{x}_i)$. Further, we define a weight vector $\mathbf{w}$ and bias b. SVMs learn a maximum margin separating hyperplane by solving a constrained optimization problem,

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} + \frac{1}{2}C\sum_i^n [1 - y_i(\mathbf{w}^\top\phi(\mathbf{x}_i) + b)]_+^2, \quad (1)$$

where constant C is the regularization trade-off hyper-parameter, and $[a]_+ = \max(a, 0)$. The squared hinge-loss penalty guarantees differentiability of (1), and simplifies the derivation in section 4. A test input is classified by the sign of the SVM predicting function

$$f[\phi(\mathbf{x}_j)] = \mathbf{w}^\top\phi(\mathbf{x}_j) + b. \quad (2)$$

Gradient Boosted Trees (GBRT). Given a continuous and differentiable loss function $\mathcal{L}$, GBRT (Friedman, 2001) learns an additive classifier $H^T(\mathbf{x}) = \sum_{t=1}^{T} \eta_t h^t(\mathbf{x})$ that minimizes $\mathcal{L}(H^T)$. Each $h^t \in \mathcal{H}$ is a limited depth regression tree (Breiman, 1984) (also referred to as a *weak learner*) added to the current classifier at iteration t, with learning rate $\eta_t \geq 0$. The weak learner h^t is selected to minimize the function $\mathcal{L}(H^{t-1} + \eta_t h^t)$. This is achieved by approximating the negative gradient of $\mathcal{L}$ w.r.t. the current H^{t-1}:

$$h^t = \underset{h^t \in \mathcal{H}}{\operatorname{argmin}} \sum_i \Big(- \frac{\partial \mathcal{L}}{\partial H^{t-1}(\mathbf{x}_i)} - h^t(\mathbf{x}_i) \Big)^2. \quad (3)$$

The greedy CART algorithm (Breiman, 1984) finds an approximate solution to (3). Consequently, h^t can be obtained by supplying $-\frac{\partial \mathcal{L}}{\partial H^{t-1}(\mathbf{x}_i)}$ as the regression targets for all inputs $\mathbf{x}_i$ to an off-the-shelf CART implementation (Tyree et al., 2011).

Greedy Miser. Recently, Xu et al. (2012) introduced the Greedy Miser, which incorporates feature cost into gradient boosting. Let $c_f(H)$ denote the test-time feature extraction cost of a gradient boosted tree ensemble H and $c_e(H)$ denote the CPU time to evaluate all trees[3]. Let $B_f, B_e > 0$ be corresponding finite cost budgets. The Greedy Miser solves the following optimization problem:

$$\min_H \mathcal{L}(H), \text{ s.t. } c_e(H) \leq B_e \text{ and } c_f(H) \leq B_f, \quad (4)$$

where $\mathcal{L}$ is continuous and differentiable. To formalize the *feature cost*, they define an auxiliary function $\mathcal{F}_\theta(h^t) \in \{0, 1\}$ indicating if feature θ is used in tree h^t for the first time, (*i.e.* $\mathcal{F}_\theta(h^t) = 1$). The authors show that by incrementally selecting h^t according to

$$\min_{h^t \in \mathcal{H}} \sum_i \Big(- \frac{\partial \mathcal{L}}{\partial H^{t-1}(\mathbf{x}_i)} - h^t(\mathbf{x}_i) \Big)^2 + \lambda \sum_\theta \mathcal{F}_\theta(h^t) c_\theta, (5)$$

the constrained optimization problem in eq. (4) is (approximately) minimized up to a local minimum (stronger guarantees exist if $\mathcal{L}$ is convex). Here, λ trades off the classification loss with the feature extraction cost (enforcing budget B_f) and the maximum number of iterations limits the tree evaluation cost (enforcing budget B_e).

4. SVM on a Test-time Budget

As a lead-up to Anytime Feature Representations, we formulate the learning of the feature representation mapping $\phi : \mathcal{R}^d \to \mathcal{R}^S$ and the SVM classifier $(\mathbf{w}, b)$ such that the costs of the final classification $c_f(f[\phi(\mathbf{x})]), c_e(f[\phi(\mathbf{x})])$ are within cost budgets B_f, B_e. In the following section we extend this formulation to an anytime setting, where B_f and B_e are unknown and the user can interrupt the classifier at any time. As the SVM classifier is *linear*, we consider its evaluation free during test-time and the cost c_e originates entirely from the computation of $\phi(\mathbf{x})$.

Boosted representation. We learn a representation with a variant of the boosting trick (Trzcinski et al., 2012; Chapelle et al., 2011). To differentiate the original features $\mathbf{x}$ and the new feature representation $\phi(\mathbf{x})$, we refer only to original features as "*features*", and the components of the new representation as "*dimensions*". In particular, we learn a representation $\phi(\mathbf{x}) \in \mathcal{R}^S$ through the mapping function ϕ, where S is the total number of dimensions of our new representation. Each dimension s of $\phi(\mathbf{x})$ (denoted $[\phi]_s$) is a gradient boosted classifier, *i.e.* $[\phi]_s = \eta \sum_{t=0}^{T} h_s^t$. Specifically, each h_s^t is a limited depth regression tree.

For each dimension s, we initialize $[\phi]_s$ with the s^{th} tree obtained from running the Greedy Miser for S iterations with a very small feature budget B_f. Subsequent trees are learned as described in the following. During classification, the SVM weight vector $\mathbf{w}$ assigns a weight w_s to each dimension $[\phi]_s$.

Train/Validation Split. As we learn the feature representation ϕ *and* the classifier $\mathbf{w}, b$ jointly, overfitting is a concern, and we carefully address it in our learning setup. Usually, overfitting in SVMs can be overcome by setting the regularization trade-off parameter C carefully with cross-validation. In our setting, however, the representation *changes* and the hyper-parameter C needs to be adjusted correspondingly. We suggest a more principled setup, inspired by Chapelle et al. (2002), and also learn the hyper-parameter C. To avoid trivial solutions, we divide our training data into two equally-sized parts, which we refer to as training and validation sets, $\mathcal{T}$ and $\mathcal{V}$. The representation is learned on both sets, whereas the classifier $\mathbf{w}, b$ is trained only on $\mathcal{T}$, and the hyper-parameter is tuned for $\mathcal{V}$. We further split the validation set into validation $\mathcal{V}$ and a held-out set $\mathcal{O}$ in a 80/20 split. The held-out set $\mathcal{O}$ is used for early-stopping.

Nested optimization. We define a loss function that approximates the 0-1 loss on the *validation* set $\mathcal{V}$,

$$\mathcal{L}_\mathcal{V}(\phi; \mathbf{w}, b) = \sum_{\mathbf{x}_i \in \mathcal{V}} \beta_{y_i} \sigma\Big(f(\phi(\mathbf{x}_i))\Big), \quad (6)$$

where $\sigma(z) = \frac{1}{1+e^{az}}$ is a soft approximation of the $sign(\cdot)$ step function (we use $a = 5$ throughout, similar

[3] Note that both costs can be in different units. Also, it is possible to set $c_e(H) = 0$ for all H. We set the evaluation cost of a single tree to 1 cost unit.

to Chapelle et al. (2002)) and $\beta_{y_i} > 0$ denotes a class specific weight to address potential class imbalance. $f(\cdot)$ is the SVM predicting function defined in (2). The classifier parameters $(\mathbf{w}, b)$ are assumed to be the optimal solution of (1) for the training set $\mathcal{T}$. We can express this relation as a nested optimization problem (in terms of the SVM parameters $\mathbf{w}, b$) and incorporate our test-time budgets B_e, B_f:

$$\min_{\phi, C} \mathcal{L}_{\mathcal{V}}(\phi, \mathbf{w}, b) \text{ s.t. } c_e(\phi) \leq B_e \text{ and } c_f(\phi) \leq B_f \quad (7)$$

$$\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2} C \sum_{i}^{n} \beta_{y_i} [1 - y_i(\mathbf{w}^\top \phi(\mathbf{x}_i) + b)]_+^2.$$

According to Theorem 4.1 in Bonnans & Shapiro (1998), $\mathcal{L}_{\mathcal{V}}$ is continuous and differentiable based on the uniqueness of the optimal solution $\mathbf{w}^*, b^*$. This is a sufficient prerequisite for being able to solve $\mathcal{L}_{\mathcal{V}}$ via the Greedy Miser (5), and since the constraints in (7) are analogous to (4), we can optimize it accordingly.

Tree building. The optimization (7) is essentially solved by a modified version of gradient descent, updating ϕ and C. Specifically, for fast computation, we update one dimension $[\phi]_s$ at a time, as we can utilize the previous learned tree in the same dimension to speed up computation for the next tree (Tyree et al., 2011). The computation of $\frac{\partial \mathcal{L}_{\mathcal{V}}}{\partial [\phi]_s}$ and $\frac{\partial \mathcal{L}_{\mathcal{V}}}{\partial C}$ is described in detail in section 4.2. At each iteration, the tree h_s^t is selected to trade-off the gradient fit of the loss function $\mathcal{L}_{\mathcal{V}}$ with the feature cost of the tree,

$$\min_{h_s^t} \sum_i \Big(- \frac{\partial \mathcal{L}_{\mathcal{V}}}{\partial [\phi]_s(\mathbf{x}_i)} - h_s^t(\mathbf{x}_i) \Big)^2 + \lambda \sum_{\theta} \mathcal{F}_\theta(h_s^t) c_\theta. \quad (8)$$

We use the learned tree h_s^t to update the representation $[\phi]_s = [\phi]_s + \eta h_s^t$. At the same time, the variable C is updated with small gradient steps.

4.1. Anytime Feature Representations

Minimizing (7) results in a cost-sensitive SVM $(\mathbf{w}, b)$ that uses a feature representation $\phi(\mathbf{x})$ to make classifications within test-time budgets B_f, B_e. In the anytime learning setting, however, the test-time budgets are *unknown*. Instead, the user can interrupt the test evaluation at any time.

Anytime parameters. We refer to our approach as *Anytime Feature Representations* (AFR) and Algorithm 1 summarizes the individual steps of AFR in pseudo-code. We obtain an anytime setting by steadily increasing B_e and B_f until the cost constraint has no effect on the optimal solution. In practice, the tree budget (B_e) increase is enforced by adding one tree h_s^t at a time (where t ranges from 1 to T). The feature budget B_f is enforced by the parameter λ in (8).

Algorithm 1 AFR in pseudo-code.

```
Initialize λ=λ_0, s_0 = 1
while λ > ε do
    Initialize φ = [h^0_{s_0}(·), ..., h^0_{s_0+S}(·)]^⊤ with (5).
    for s = s_0 to s_0 + S do
        for t = 1 to T do
            Train an SVM using φ to obtain w and b.
            If accuracy on O has increased, continue.
            Compute gradients ∂L_V/∂[φ]_s and ∂L_V/∂C
            Update C = C − γ ∂L_V/∂C
            Call CART with impurity (8) to obtain h^t_s
            Stop if Σ_i h^t_s(x_i) ∂L_V/∂[φ]_s(x_i) < 0
            Update [φ]_s = [φ]_s + η h^t_s.
        end for
    end for
    λ := λ/2 and s_0 += S.
end while
```

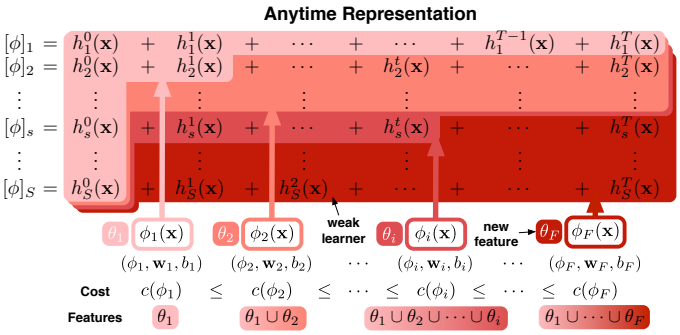

Figure 1. A schematic layout of Anytime Feature Representations. Different shaded areas indicate representations of different costs, the darker the costlier. During training time, SVM parameters $\mathbf{w}, b$ are saved every time a new feature θ_i is extracted. During test-time, under budgets B_e, B_f, we use the most expensive triplet $(\phi_k, \mathbf{w}_k, b_k)$ with cost $c_e(\phi_k) \leq B_e$ and $c_f(\phi_k) \leq B_f$.

As the feature cost is dominant, we slowly decrease λ (starting from some high value λ_0). For each intermediate value of λ we learn S dimensions of $\phi(\mathbf{x})$ (each dimension consisting of T trees). Whenever all S dimensions are learned, λ is divided by a factor of 2 and an additional S dimensions of $\phi(\mathbf{x})$ are learned and concatenated to the existing representation.

Whenever a new feature is extracted by a tree h_s^t, the cost increases substantially. Therefore we store the learned representation mapping function and the learned SVM parameters whenever a new feature is extracted. We overload ϕ_f to denote the representation learned with feature f^{th} extracted, and $\mathbf{w}_f, b_f$ as the corresponding SVM parameters. Storing these parameters results in a series of triplets $(\phi_1, \mathbf{w}_1, b_1) \ldots (\phi_F, \mathbf{w}_F, b_F)$ of increasing cost, *i.e.* $c(\phi_1) \leq \cdots \leq c(\phi_F)$ (where F is the total number of extracted features). Note that we save the map-

ping function ϕ, rather than the representation of each training input $\phi(\mathbf{x})$.

Evaluation. During test time, the classifier may be stopped during the extraction of the $f{+}1^{th}$ feature, because the feature budget B_f (unknown during training time) has been reached. In this case, to make a prediction, we sum the previously-learned representations generated by the first f features $\mathbf{w}_f^\top \sum_{k=1}^{f} \phi_k(\mathbf{x}) + b_f$. This approach is schematically depicted in figure 1.

Early-stopping. Updating each dimension with a fixed number of T trees may lead to overfitting. We apply early-stopping by evaluating the prediction accuracy on the hold-out set $\mathcal{O}$. We stop adding trees to each dimension whenever this accuracy decreases. Algorithm (1) details all steps of our algorithm.

4.2. Optimization

Updating feature representation $\phi(\mathbf{x})$ requires computing the gradient of the loss function $\mathcal{L}_\mathcal{V}$ w.r.t. $\phi(\mathbf{x})$ as stated in eq. (8). In this section we explain how to compute the necessary gradients efficiently.

Gradient w.r.t. $\phi(\mathbf{x})$. We use the chain rule to compute the derivative of $\mathcal{L}_\mathcal{V}$ w.r.t. each dimension $[\phi]_s$,

$$\frac{\partial \mathcal{L}_\mathcal{V}}{\partial [\phi]_s} = \frac{\partial \mathcal{L}_\mathcal{V}}{\partial f} \frac{\partial f}{\partial [\phi]_s}, \tag{9}$$

where f is the prediction function in eq. (2). As changing $[\phi]_s$ not only affects the validation data, but also the representation of the training set, $\mathbf{w}$ and b are also functions of $[\phi]_s$. The derivative of f w.r.t. the representation of the training inputs, $[\phi]_s \in \mathcal{T}$ is

$$\frac{\partial f}{\partial [\phi]_s} = \left(\frac{\partial \mathbf{w}}{\partial [\phi]_s}\right)^\top \phi_\mathcal{V} + \frac{\partial b}{\partial [\phi]_s}, \tag{10}$$

where we denote all validation inputs by $\phi_\mathcal{V}$. For validation inputs, the derivative w.r.t. $[\phi]_s \in \mathcal{V}$ is

$$\frac{\partial f}{\partial [\phi]_s} = \mathbf{w}^\top \frac{\partial \phi_\mathcal{V}}{\partial [\phi]_s}. \tag{11}$$

Note that with $|\mathcal{T}|$ training inputs and $|\mathcal{V}|$ validation inputs, the gradient consists of $|\mathcal{T}| + |\mathcal{V}|$ components.

In order to compute the remaining derivatives $\frac{\partial \mathbf{w}}{\partial [\phi]_s}$ and $\frac{\partial b}{\partial [\phi]_s}$ we will express $\mathbf{w}$ and b in closed-form w.r.t. $[\phi]_s$. First, let us define the contribution to the loss of input $\mathbf{x}_i$ as $\xi_i = [1 - y_i(\mathbf{w}^{*\top}\phi(\mathbf{x}_i) + b^*)]_+$. The optimal value $\mathbf{w}^*, b^*$ is only affected by support vectors (inputs with $\xi_i > 0$). Without loss of generality, let us assume that those inputs are the first m in our ordering, $\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{m}$. We remove all non-support vectors, and let $\widehat{\mathbf{\Phi}} = [y_1\phi_1, \ldots, y_{n_\mathrm{m}}\phi_{n_\mathrm{m}}]$, and $\xi = [\xi_1, \ldots, \xi_{n_\mathrm{m}}]^\top$. We also define a diagonal matrix $\Lambda \in \mathcal{R}^{n_\mathrm{m} \times n_\mathrm{m}}$ whose diagonal elements are class weight $\Lambda_{ii} = \beta_{y_i}$. We can then rewrite the nested SVM optimization problem within (7) in matrix form:

$$\min_{\mathbf{w},b} \; L = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}(\mathbf{1} - \mathbf{w}^\top\widehat{\mathbf{\Phi}} - b\mathbf{y})^\top \Lambda (\mathbf{1} - \mathbf{w}^\top\widehat{\mathbf{\Phi}} - b\mathbf{y}).$$

As this objective is convex, we can obtain the optimal solution of $\mathbf{w}, b$ by setting $\frac{\partial L}{\partial \mathbf{w}}$ and $\frac{\partial L}{\partial b}$ to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Longrightarrow \mathbf{w} - C\widehat{\mathbf{\Phi}}\Lambda(\mathbf{1} - \widehat{\mathbf{\Phi}}^\top\mathbf{w} - b\mathbf{y}^\top) = \mathbf{0}.$$
$$\frac{\partial L}{\partial b} = 0 \Longrightarrow -\mathbf{y}\Lambda(\mathbf{1} - \widehat{\mathbf{\Phi}}^\top\mathbf{w} - b\mathbf{y}^\top) = 0.$$

By re-arranging the above equations, we can express them as a matrix equality,

$$\underbrace{\begin{bmatrix} \frac{\mathbf{I}}{C} + \widehat{\mathbf{\Phi}}\Lambda\widehat{\mathbf{\Phi}}^\top & \widehat{\mathbf{\Phi}}\Lambda\mathbf{y}^\top \\ \mathbf{y}\Lambda\widehat{\mathbf{\Phi}}^\top & \mathbf{y}\Lambda\mathbf{y}^\top \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \underbrace{\begin{bmatrix} \widehat{\mathbf{\Phi}}\Lambda\mathbf{1} \\ \mathbf{y}\Lambda\mathbf{1} \end{bmatrix}}_{\mathbf{z}}.$$

We absorb the coefficients on the left-hand side into a design matrix $\mathbf{M} \in \mathcal{R}^{d+1 \times d+1}$, and right-hand side into a vector $\mathbf{z} \in \mathcal{R}^{d+1}$. Consequently, we can express $\mathbf{w}$ and b as a function of $\mathbf{M}^{-1}$ and $\mathbf{z}$, and derive their derivatives w.r.t. $[\phi]_s$ from the matrix inverse rule (Petersen & Pedersen, 2008), leading to

$$\frac{\partial [\mathbf{w}^\top, b]^\top}{\partial [\phi_i]_s} = \mathbf{M}^{-1}\left(\frac{\partial \mathbf{z}}{\partial [\phi_i]_s} - \frac{\partial \mathbf{M}}{\partial [\phi_i]_s}\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}\right) \tag{12}$$

To compute the derivatives $\frac{\partial \mathbf{M}}{\partial [\phi]_s}$, we note that the upper left block of $\mathbf{M}$ is a $d \times d$ inner product matrix scaled by Λ and translated by $\frac{\mathbf{I}}{C}$, and we obtain the derivative w.r.t. each element of the upper left block,

$$\frac{\partial (\frac{\mathbf{I}}{C} + \widehat{\mathbf{\Phi}}\Lambda\widehat{\mathbf{\Phi}}^\top)_{rs}}{\partial [\phi]_s(\mathbf{x}_i)} = \begin{cases} \beta_{y_i}[\phi]_r(\mathbf{x}_i) & \text{if } r \neq s, \\ 2\beta_{y_i}[\phi]_s(\mathbf{x}_i) & \text{if } r = s. \end{cases}$$

The remaining derivatives are $\frac{\partial \widehat{\mathbf{\Phi}}\Lambda\mathbf{y}^\top}{\partial [\phi]_s(\mathbf{x}_i)} = \beta_{y_i}$ and $\frac{\partial \mathbf{z}}{[\phi]_s(\mathbf{x}_i)} = [0, \ldots, y_i\beta_{y_i}, \ldots, 0]^\top \in \mathcal{R}^{d+1}$. To complete the chain rule in eq. (9), we also need

$$\frac{\partial \mathcal{L}_\mathcal{V}}{\partial f} = -y_i\sigma(y_i f[\phi(\mathbf{x}_i)])(1 - \sigma(y_i f[\phi(\mathbf{x}_i)])). \tag{13}$$

Combining eqs. (10), (11), (12) and (13) completes the gradient $\frac{\partial \mathcal{L}_\mathcal{V}}{\partial [\phi]_s}$.

Gradient w.r.t. C. The derivative $\frac{\partial f}{\partial C}$ is very similar to $\frac{\partial f}{\partial [\phi]_s}$, the difference being in $\frac{\partial \mathbf{M}}{\partial C}$, which only has non-zero value on diagonal elements,

$$\frac{\partial \mathbf{M}_{rs}}{\partial C} = \begin{cases} -\frac{1}{C^2} & \text{if } s = r \wedge r \neq m+1, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Although computing the derivative requires the inversion of matrix $\mathbf{M}$, $\mathbf{M}$ is only a $(d+1) \times (d+1)$ matrix. Because our algorithm converges after generating a few ($d \approx 100$) dimensions, the inverse operation is not computationally intensive.

5. Results

We evaluate our algorithm on a synthetic data set in order to demonstrate the AFR learning approach, as well as two benchmark data sets from very different domains: the Yahoo! Learning to Rank Challenge data set (Chapelle & Chang, 2011) and the Scene 15 recognition data set from Lazebnik et al. (2006).

Synthetic data. To visualize the learned anytime feature representation, we construct a synthetic data set as follows. We generate $n = 1000$ points (640 for training/validation and 360 for testing) uniformly sampled from four different regions of two-dimensional space (as shown in figure 2, left). Each point is labeled to be in class 1 or class 2 according to the XOR rule. These points are then randomly-projected into a ten-dimensional feature space (not shown). Each of these ten features is assigned an extraction cost: $\{1, 1, 1, 2, 5, 15, 25, 70, 100, 1000\}$. Correspondingly, each feature θ has zero-mean Gaussian noise added to it, with variance $\frac{1}{c_\theta}$ (where c_θ is the cost of feature θ). As such, cheap features are poorly representative of the classes while more expensive features more accurately distinguish the two classes. To highlight the feature-selection capabilities of our technique we set the evaluation cost c_e to 0. Using this data, we constrain the algorithm to learn a two-dimensional anytime representation (*i.e.* $\phi(\mathbf{x}) \in \mathcal{R}^2$).

The center portion of figure 2 shows the anytime representations of testing points for various test-time budgets, as well as the learned hyperplane (black line), margins (gray lines) and classification accuracies. As the allowed feature cost budget is increased, AFR steadily adjusts the representation and classifier to better distinguish the two classes. Using a small set of features (cost = 95) AFR can achieve nearly perfect test accuracy and using all features AFR fully separates the test data.

The rightmost part of figure 2 shows how the learned SVM classifier changes as the representation changes. The coefficients of the hyperplane $\mathbf{w} = [w_1, w_2]^\top$ initially change drastically to appropriately weight the AFR features, then decrease gradually as more weak learners are added to ϕ. Throughout, the hyperparameter C is also optimized.

Yahoo Learning to Rank. The Yahoo! Learning to Rank Challenge data set consists of query-document instance pairs, with labels having values from $\{0, 1, 2, 3, 4\}$, where 4 means the document is perfectly relevant to the query and 0 means it is irrelevant. Following the steps of Chen et al. (2012), we transform the data into a binary classification problem by distinguishing purely between relevant ($y_i \geq 3$) and irrelevant ($y_i < 3$) documents. The resulting labels are $y_i \in \{+1, -1\}$. The total binarized data set contains 2000, 2002, and 2001 training, validation and testing queries and 20258, 20258, 26256 query-document instances respectively. As in Chen et al. (2012) we replicate each negative, irrelevant instance 10 times to simulate the scenario where only a few documents out of hundreds of thousands of candidate documents are highly relevant. Indeed in real world applications, the distribution of the two classes is often very skewed, with vastly more negative examples presented.

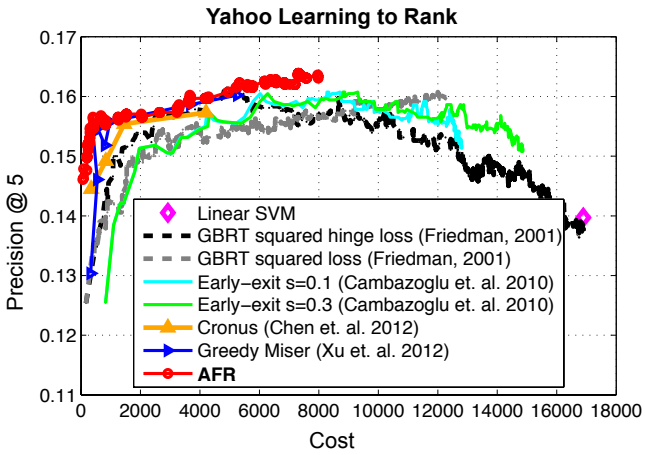

Figure 3. The accuracy/cost trade-off curves for a number of state-of-the-art algorithms on the Yahoo! Learning to Rank Challenge data set. The cost is measured in units of the time required to evaluate one weak learner.

Each input contains 519 features, and the feature extraction costs are in the set $\{1, 5, 10, 20, 50, 100, 150, 200\}$. The unit of cost is the time required to evaluate one limited-depth regression tree $h^t(\cdot)$, thus the evaluation cost c_e is set to 1. To evaluate the cost-accuracy performance, we follow the typical convention for a binary ranking data set and use the Precision@5 metric. This counts how many documents are relevant in the top 5 retrieved documents for each query.

In order to address the label inbalance, we add a multiplicative weight to the loss of all positive examples, β_+, which is set by cross validation ($\beta_+ = 2$). We set the hyper-parameters to $T = 10$, $S = 20$ and $\lambda_0 = 10$. As the algorithm is by design fairly insensitive to hyper-parameters, this setting was determined without needing to search through (T, S, λ_0) space.

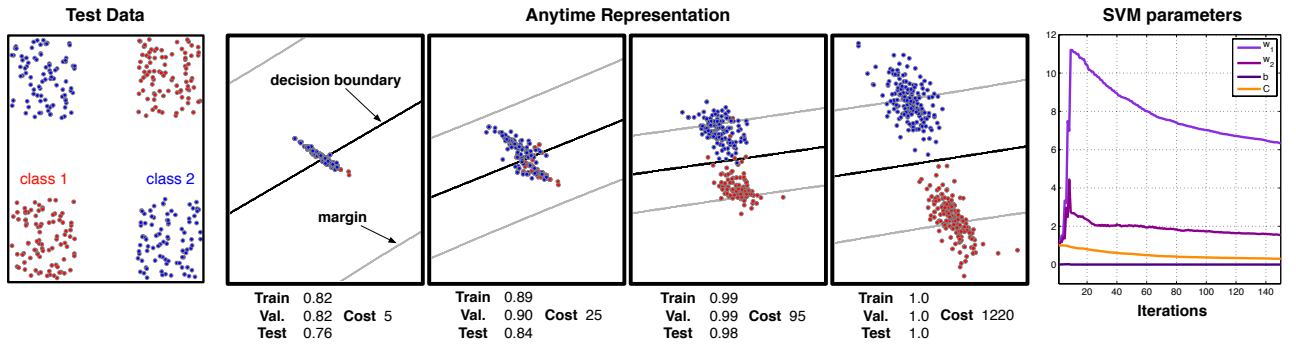

Figure 2. A demonstration of our method on a synthetic data set (shown at left). As the feature representation is allowed to use more expensive features, AFR can better distinguish the test data of the two classes. At the bottom of each representation is the classification accuracies of the training/validation/testing data and the cost of the representation. The rightmost plot shows the values of SVM parameters $\mathbf{w}, b$ and hyper-parameter C at each iteration.

Comparison. The most basic baseline is GBRT without cost consideration. We apply GBRT using two different loss functions: the squared loss and the unregularized squared hinge loss. In total we train 2000 trees. We plot the cost and accuracy curves of GBRT by adding 10 trees at a time. In addition to this additive classifier, we show the results of a linear SVM applied to the original features as well.

We also compare against current state-of-the-art competing algorithms. We include *Early-Exit* (Cambazoglu et al., 2010), which is based on GBRT. It short-circuits the evaluation of lower ranked and unpromising documents at test-time, based on some threshold s (we show $s = 0.1, 0.3$), reducing the overall test-time cost. *Cronus* (Chen et al., 2012) improves over Early-Exit by reweighing and re-ordering the learned trees into a feature-cost sensitive cascade structure. We show results of a cascade with a maximum of 10 nodes. All of its hyper-parameters (cascade length, keep ratio, discount, early-stopping) were set based on the validation set. We generate the cost/accuracy curve by varying the trade-off parameter λ, in their paper. Finally, we compare against *Greedy Miser* (Xu et al., 2012) trained using the unregularized squared hinge loss. The cost/accuracy curve is generated by re-training the algorithm with different cost/accuracy trade-off parameters λ. We also use the validation set to select the best number of trees needed for each λ.

Figure 3 shows the performance of all algorithms. Although the linear SVM uses all features to make cost-insensitive predictions, it achieves a relatively poor result on this ranking data set, due to the limited power of a linear decision boundary on the original feature space. This trend has previously been observed in Chapelle & Chang (2011). GBRT with unregularized squared hinge loss and squared loss achieve peak accuracy after using a significant amount of the feature set. Early-Exit only provides limited improvement over GBRT when the budget is low. This is primarily because, in this case, the test-time cost is dominated by feature extraction rather than the evaluation cost. Cronus improves over Early-Exit significantly due to its automatic stage reweighing and reordering. However, its power is still limited by its feature representation, which is not cost-sensitive. AFR out-performs the best performance of Greedy Miser for a variety of cost budgets. Different from Greedy Miser, which must be re-trained for different budgets along the cost/accuracy trade-off curve (each resulting in a different model), AFR consists of a single model which can be halted at any point along its curve—providing a state-of-the-art *anytime* classifier. It is noteworthy that AFR obtains the highest test-scores overall, which might be attributed to the better generalization of large-margin classifiers.

Scene recognition. The second data set we experiment with is from the image domain. The scene 15 (Lazebnik et al., 2006) data set contains 4485 images from 15 scene classes. The task is to classify the scene in each image. Following the procedure use by Li et al. (2010); Lazebnik et al. (2006), we construct the training set by selecting 100 images from each class, and leave the remaining 2865 images for testing. We extract a variety of vision features from Xiao et al. (2010) with very different computational costs: GIST, spatial HOG, Local Binary Pattern (LBP), self-similarity, texton histogram, geometric texton, geometric color, and Object Bank (Li et al., 2010). As mentioned by the authors of Object Bank, each object detector works independently. Therefore we apply 177 object detectors to each image, and treat each of them as independent descriptors. In total, we have 184 different image

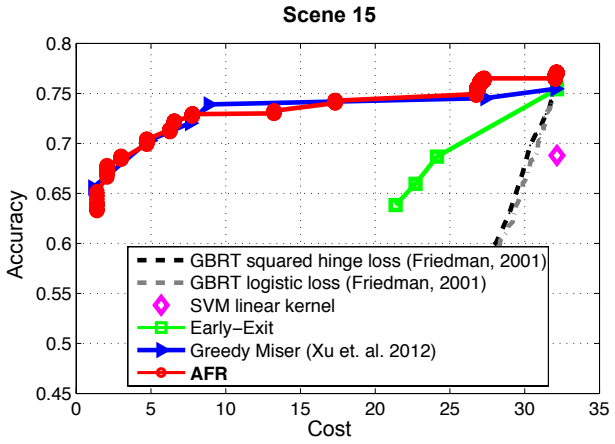

Figure 4. The accuracy/cost performance trade-off for different algorithms on the Scene 15 multi-class scene recognition problem. The cost is in units of CPU time.

descriptors, and the total number of resulting raw features is 76187. The feature extraction cost is the actual CPU time to compute each feature on a desktop with dual 6-core Intel i7 CPUs with 2.66GHz, ranging from 0.037s (Object Bank) to 9.282s (geometric texton). Since computing each type of image descriptor results in a group of features, as long as any of the features in a descriptor is requested, we extract the entire descriptor. Thus, subsequent requests for features in that descriptor are free.

We train 15 one-vs-all classifiers, and learn the feature representation mapping ϕ, the SVM parameters ($\mathbf{w}$,b,C) for each classifier separately. Since each descriptor is free once extracted, we also set the descriptor cost to zero whenever it is use by one of the 15 classifiers. To overcome the problem of different decision value scales resulting from different one-vs-all classifiers, we use Platt scaling (Platt, 1999) to rescale each classifier prediction within $[0, 1]$.[4] We use the same hyper-parameters as the Yahoo! data set, except we set $\lambda_0 = 2^{10}$, as the unit of cost in scene15 is much smaller.

Figure 4 demonstrates the cost/accuracy performance of several current state-of-the-art techniques and our algorithm. The GBRT-based algorithms include GBRT using the logistic loss and the squared loss, where we use Platt scaling for the hinge loss variant to cope with the scaling problem. We generate the curve by adding 10 trees at a time. Although these two methods achieve high accuracy, their costs are also significantly higher due to their cost-insensitive nature. We also evaluate a linear SVM. Because it is only able to learn a linear decision boundary on the *original* feature space, it has a lower accuracy than the GBRT-based techniques for a given cost. For cost-sensitive methods, we first evaluate *Early-Exit*. As this is a multi-class classification problem, we introduce an early-exit every 10 trees, and we remove test inputs after platt-scaling results in a score greater than a threshold s. We plot the curve by varying s. Since Early-Exit lacks the capability to automatically pick expensive and accurate features early-on, its improvement is very limited. For *Greedy Miser*, we split the training data into 75/25 and use the smaller subset as validation to set the number of trees. We use un-regularized squared hinge-loss with different values of the cost/accuracy trade-off parameter $\lambda \in \{4^0, 4^1, 4^2, 4^3, 4^4, 4^5\}$. Greedy Miser performs better than the previous baselines, and our approach consistently matches it, save one setting. Our method *AFR* generates a smoother budget curve, and can be stopped anytime to provide predictions at test-time.

[4]Platt scaling makes SVM predictions interpretable as probabilities. This can also be use to monitor the confidence threshold of the anytime classifiers to stop evaluation when a confidence threshold is met (*e.g.* in medical applications to avoid further costly feature extraction).

6. Discussion

To our knowledge, we provide the first learning algorithm for cost-sensitive anytime feature representations. Our results are highly encouraging, in particular AFR matches or even outperforms the results of the current best cost-sensitive classifiers, which must be provided with knowledge about the exact test-time budget during training.

Addressing the *anytime classification* setting in a principled fashion has high impact potential in several ways: i) reducing the cost required for the *average* case frees up more resources for the rare difficult cases—thus improving accuracy; ii) decreasing computational demands of massive industrial computations can substantially reduce energy consumption and greenhouse emissions; iii) classifier querying enables time-sensitive applications like pedestrian detection in cars with inherent accuracy/urgency trade-offs.

Learning anytime *representations* adds new flexibility towards the choice of classifier and the learning setting and may enable new use cases and application areas. As future work, we plan to focus on incorporating other classification frameworks and apply our setting to critical applications such as real-time pedestrian detection and medical applications.

Acknowledgements KQW, ZX, and MK are supported by NIH grant U01 1U01NS073457-01 and NSF grants 1149882 and 1137211. The authors thank Stephen W. Tyree for clarifying discussions and suggestions.

References

Bonnans, J Frédéric and Shapiro, Alexander. Optimization problems with perturbations: A guided tour. *SIAM review*, 40(2):228–264, 1998.

Breiman, L. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

Busa-Fekete, R., Benbouzid, D., Kégl, B., et al. Fast classification using sparse decision dags. In *ICML*, 2012.

Cambazoglu, B.B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., and Degenhardt, J. Early exit optimizations for additive machine learned ranking systems. In *WSDM'3*, pp. 411–420, 2010.

Chapelle, O. and Chang, Y. Yahoo! learning to rank challenge overview. In *JMLR: Workshop and Conference Proceedings*, volume 14, pp. 1–24, 2011.

Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

Chapelle, O., Shivaswamy, P., Vadrevu, S., Weinberger, K., Zhang, Y., and Tseng, B. Boosted multi-task learning. *Machine learning*, 85(1):149–173, 2011.

Chen, M., Xu, Z., Weinberger, K. Q., and Chapelle, O. Classifier cascade for minimizing feature evaluation cost. In *AISTATS*, 2012.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Dekel, Ofer, Shalev-Shwartz, Shai, and Singer, Yoram. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372, 2008.

Dredze, M., Gevaryahu, R., and Elias-Bachrach, A. Learning fast classifiers for image spam. In *proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2007.

Freund, Y. and Schapire, R. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pp. 23–37. Springer, 1995.

Friedman, J.H. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, pp. 1189–1232, 2001.

Gao, T. and Koller, D. Active classification based on value of classifier. In *NIPS*, pp. 1062–1070. 2011a.

Gao, Tianshi and Koller, Daphne. Multiclass boosting with hinge loss based on output coding. ICML '11, pp. 569–576, 2011b.

Gavrila, D. Pedestrian detection from a moving vehicle. *ECCV 2000*, pp. 37–49, 2000.

Grubb, A. and Bagnell, J. A. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, 2012.

Grubb, A. and Bagnell, J.A. Generalized boosting algorithms for convex optimization. *arXiv preprint arXiv:1105.2054*, 2011.

Grubb, Alexander and Bagnell, J Andrew. Boosted backpropagation learning for training deep modular networks. In *Proceedings of the International Conference on Machine Learning (27th ICML)*, 2010.

Kedem, Dor, Tyree, Stephen, Weinberger, Kilian Q., Sha, Fei, and Lanckriet, Gert. Non-linear metric learning. In *NIPS*, pp. 2582–2590. 2012.

Lazebnik, S., Schmid, C., and Ponce, J. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pp. 2169–2178, 2006.

Lefakis, L. and Fleuret, F. Joint cascade optimization using a product of boosted classifiers. In *NIPS*, pp. 1315–1323. 2010.

Li, L.J., Su, H., Xing, E.P., and Fei-Fei, L. Object bank: A high-level image representation for scene classification and semantic feature sparsification. *NIPS*, 2010.

Mohan, A., Chen, Z., and Weinberger, K. Q. Web-search ranking with initialized gradient boosted regression trees. *JMLR: Workshop and Conference Proceedings*, 14:77–89, 2011.

Petersen, K. B. and Pedersen, M. S. The matrix cookbook, Oct 2008.

Platt, J.C. Fast training of support vector machines using sequential minimal optimization. 1999.

Pujara, J., Daumé III, H., and Getoor, L. Using classifier cascades for scalable e-mail classification. In *CEAS*, 2011.

Raykar, V.C., Krishnapuram, B., and Yu, S. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *ACM SIGKDD*, pp. 853–860, 2010.

Saberian, M. and Vasconcelos, N. Boosting classifier cascades. In *NIPS*, pp. 2047–2055. 2010.

Trzcinski, Tomasz, Christoudias, Mario, Lepetit, Vincent, and Fua, Pascal. Learning image descriptors with the boosting-trick. In *NIPS*, pp. 278–286. 2012.

Tyree, S., Weinberger, K.Q., Agrawal, K., and Paykin, J. Parallel boosted regression trees for web search ranking. In *WWW*, pp. 387–396. ACM, 2011.

Viola, P. and Jones, M.J. Robust real-time face detection. *IJCV*, 57(2):137–154, 2004.

Wang, J. and Saligrama, V. Local supervised learning through space partitioning. In *NIPS*, pp. 91–99, 2012.

Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *ICML*, pp. 1113–1120, 2009.

Xiao, Jianxiong, Hays, James, Ehinger, Krista A, Oliva, Aude, and Torralba, Antonio. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pp. 3485–3492. IEEE, 2010.

Xu, Z., Weinberger, K.Q., and Chapelle, O. The greedy miser: Learning under test-time budgets. In *ICML*, pp. 1175–1182, 2012.

Xu, Zhixiang, Kusner, Matt J., Weinberger, Kilian Q., and Chen, Minmin. Cost-sensitive tree of classifiers. In Dasgupta, Sanjoy and McAllester, David (eds.), *ICML '13*, pp. to appear, 2013.

Zheng, Z., Zha, H., Zhang, T., Chapelle, O., Chen, K., and Sun, G. A general boosting method and its application to learning ranking functions for web search. In *NIPS*, pp. 1697–1704. Cambridge, MA, 2008.