Cost-Sensitive Tree of Classifiers

Zhixiang (Eddie) Xu XUZX@CSE.WUSTL.EDU
Matt J. Kusner MKUSNER@WUSTL.EDU
Kilian Q. Weinberger KILIAN@WUSTL.EDU
Minmin Chen MCHEN@WUSTL.EDU
Washington University, One Brookings Dr., St. Louis, MO 63130 USA

Abstract

Recently, machine learning algorithms have successfully entered large-scale real-world industrial applications (*e.g.* search engines and email spam filters). Here, the CPU cost during test-time must be budgeted and accounted for. In this paper, we address the challenge of balancing the test-time cost and the classifier accuracy in a principled fashion. The test-time cost of a classifier is often dominated by the computation required for feature extraction—which can vary drastically across features. We decrease this extraction time by constructing a tree of classifiers, through which test inputs traverse along individual paths. Each path extracts different features and is optimized for a specific subpartition of the input space. By only computing features for inputs that benefit from them the most, our cost-sensitive tree of classifiers can match the high accuracies of the current state-of-the-art at a small fraction of the computational cost.

1. Introduction

Machine learning algorithms are widely used in many real-world applications, ranging from email-spam (Weinberger et al., 2009) and adult content filtering (Fleck et al., 1996), to web-search engines (Zheng et al., 2008). As machine learning transitions into these industry fields, managing the CPU cost at test-time becomes increasingly important. In applications of such large scale, computation must be budgeted and accounted for. Moreover, reducing energy wasted on unnecessary computation can lead to monetary savings and reductions of greenhouse gas emissions.

The *test-time cost* consists of the time required to evaluate a classifier and the time to extract features for that classifier, where the extraction time across features is highly variable. Imagine introducing a new feature to an email spam filtering algorithm that requires 0.01 seconds to extract per incoming email. If a web-service receives one billion emails (which many do daily), it would require 115 extra CPU days to extract just this feature. Although this additional feature may increase the accuracy of the filter, the cost of computing it for *every email* is prohibitive. This introduces the problem of balancing the test-time cost and the classifier accuracy. Addressing this trade-off in a principled manner is crucial for the applicability of machine learning.

In this paper, we propose a novel algorithm, *Cost-Sensitive Tree of Classifiers* (CSTC). A CSTC tree (illustrated schematically in Fig. 1) is a tree of classifiers that is carefully constructed to reduce the *average* test-time complexity of machine learning algorithms, while maximizing their accuracy. Different from prior work, which reduces the total cost for every input (Efron et al., 2004) or which stages the feature extraction into linear cascades (Viola & Jones, 2004; Lefakis & Fleuret, 2010; Saberian & Vasconcelos, 2010; Pujara et al., 2011; Chen et al., 2012), a CSTC tree incorporates *input-dependent feature selection* into training and dynamically allocates higher feature budgets for infrequently traveled tree-paths. By introducing a probabilistic tree-traversal framework, we can compute the exact expected test-time cost of a CSTC tree. CSTC is trained with a single global loss function, whose test-time cost penalty is a direct relaxation of this expected cost. This principled approach leads to unmatched test-cost/accuracy tradeoffs as it naturally divides the input space into sub-regions and extracts expensive features only when necessary.

We make several novel contributions: 1. We introduce

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

the meta-learning framework of CSTC trees and derive the expected cost of an input traversing the tree during test-time. 2. We relax this expected cost with a mixed-norm relaxation and derive a single global optimization problem to train all classifiers jointly. 3. We demonstrate on synthetic data that CSTC effectively allocates features to classifiers where they are most beneficial and show on large-scale real-world web-search ranking data that CSTC significantly outperforms the current state-of-the-art in test-time cost-sensitive learning—maintaining the performance of the best algorithms for web-search ranking at a fraction of their computational cost.

2. Related Work

A basic approach to control test-time cost is the use of l_1-norm regularization (Efron et al., 2004), which results in a sparse feature set, and can significantly reduce the feature cost during test-time (as unused features are never computed). However, this approach fails to address the fact that some inputs may be successfully classified by only a few cheap features, whereas others strictly require expensive features for correct classification.

There is much previous work that extends single classifiers to classifier cascades (mostly for binary classification) (Viola & Jones, 2004; Lefakis & Fleuret, 2010; Saberian & Vasconcelos, 2010; Pujara et al., 2011; Chen et al., 2012). In these cascades, several classifiers are ordered into a sequence of stages. Each classifier can either reject inputs (predicting them), or pass them on to the next stage, based on the prediction of each input. To reduce the test-time cost, these cascade algorithms enforce that classifiers in early stages use very few and/or cheap features and reject many easily-classified inputs. Classifiers in later stages, however, are more expensive and cope with more difficult inputs. This linear structure is particularly effective for applications with highly skewed class imbalance and generic features. One celebrated example is face detection in images, where the majority of all image regions do not contain faces and can often be easily rejected based on the response of a few simple Haar features (Viola & Jones, 2004). The linear cascade model is however less suited for learning tasks with *balanced classes* and *specialized features*. It cannot fully capture the scenario where different partitions of the input space require different expert features, as all inputs follow the same linear chain.

Grubb & Bagnell (2012) and Xu et al. (2012) focus on training a classifier that explicitly trades-off test-time cost and accuracy. Instead of optimizing the trade-off by building a cascade, they push the cost trade-off into the construction of the weak learners. It should be noted that, in spite of the high accuracy achieved by these techniques, the algorithms are based heavily on stage-wise regression (gradient boosting) (Friedman, 2001), and are less likely to work with more general weak learners.

Gao & Koller (2011) use locally weighted regression during test time to predict the information gain of unknown features. Different from our algorithm, their model is learned during test-time, which introduces an additional cost especially for large data sets. In contrast, our algorithm learns and fixes a tree structure in training and has a test-time complexity that is constant with respect to the training set size.

Karayev et al. (2012) use reinforcement learning to dynamically select features to maximize the average precision over time in an object detection setting. In this case, the dataset has multi-labeled inputs and thus warrants a different approach than ours.

Hierarchical Mixture of Experts (HME) (Jordan & Jacobs, 1994) also builds tree-structured classifiers. However, in contrast to CSTC, this work is not motivated by reductions in test-time cost and results in fundamentally different models. In CSTC, each classifier is trained with the test-time cost in mind and each test-input only traverses a *single* path from the root down to a terminal element, accumulating path-specific costs. In HME, all test-inputs traverse all paths and all leaf-classifiers contribute to the final prediction, incurring the same cost for all test-inputs.

Recent tree-structured classifiers include the work of Deng et al. (2011), who speed up the training and evaluation of label trees (Bengio et al., 2010), by avoiding many binary one-vs-all classifier evaluations. Differently, we focus on problems in which feature extraction time dominates the test-time cost which motivates different algorithmic setups. Dredze et al. (2007) combine the cost to select a feature with the mutual information of that feature to build a decision tree that reduces the feature extraction cost. Different from this work, they do not directly minimize the total test-time cost of the decision tree or the risk. Possibly most similar to our work are (Busa-Fekete et al., 2012), who learn a directed acyclic graph via a Markov decision process to select features for different instances, and (Wang & Saligrama, 2012), who adaptively partition the feature space and learn local region-specific classifiers. Although each work is similar in motivation, the algorithmic frameworks are very different and can be regarded complementary to ours.

3. Cost-sensitive classification

We first introduce our notation and then formalize our test-time cost-sensitive learning setting. Let the training data consist of inputs $\mathcal{D}=\{\mathbf{x}_1,\dots,\mathbf{x}_n\}\subset\mathcal{R}^d$ with corresponding class labels $\{y_1,\dots,y_n\}\subseteq\mathcal{Y}$, where $\mathcal{Y}=\mathcal{R}$ in the case of regression ($\mathcal{Y}$ could also be a finite set of categorical labels—because of space limitations we do not focus on this case in this paper).

Non-linear feature space. Throughout this paper, we focus on linear classifiers but in order to allow non-linear decision boundaries we map the input into a non-linear feature space with the "boosting trick" (Friedman, 2001; Chapelle et al., 2011), prior to our optimization. In particular, we first train gradient boosted regression trees with a squared loss penalty (Friedman, 2001), $H'(\mathbf{x}_i)=\sum_{t=1}^{T}h_t(\mathbf{x}_i)$, where each function $h_t(\cdot)$ is a limited-depth CART tree (Breiman, 1984). We then apply the mapping $\mathbf{x}_i\rightarrow\phi(\mathbf{x}_i)$ to all inputs, where $\phi(\mathbf{x}_i)=[h_1(\mathbf{x}_i),\dots,h_T(\mathbf{x}_i)]^\top$. To avoid confusion between CART trees and the CSTC tree, we refer to CART trees $h_t(\cdot)$ as *weak learners*.

Risk minimization. At each node in the CSTC tree we propose to learn a linear classifier in this feature space, $H(\mathbf{x}_i)=\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}$ with $\boldsymbol{\beta}\in\mathcal{R}^T$, which is trained to explicitly reduce the CPU cost during test-time. We learn the weight-vector $\boldsymbol{\beta}$ by minimizing a convex empirical risk function $\ell(\phi(\mathbf{x}_i)^\top\boldsymbol{\beta},y_i)$ with l_1 regularization, $|\boldsymbol{\beta}|$. In addition, we incorporate a cost term $c(\boldsymbol{\beta})$, which we derive in the following subsection, to restrict test-time cost. The combined test-time cost-sensitive loss function becomes

$$\mathcal{L}(\boldsymbol{\beta})=\underbrace{\sum_i\ell(\phi(\mathbf{x}_i)^\top\boldsymbol{\beta},y_i)+\rho|\boldsymbol{\beta}|}_{\text{regularized risk}}+\lambda\underbrace{c(\boldsymbol{\beta})}_{\text{test-cost}},\qquad(1)$$

where λ is the accuracy/cost trade-off parameter, and ρ controls the strength of the regularization.

Test-time cost. There are two factors that contribute to the test-time cost of each classifier. The weak learner evaluation cost of all active $h_t(\cdot)$ (with $|\beta_t|>0$) and the feature extraction cost for all features used in these weak learners. We assume that features are computed *on demand* with the cost $\mathbf{c}$ the first time they are used, and are free for future use (as feature values can be cached). We define an auxiliary matrix $\mathbf{F}\in\{0,1\}^{d\times T}$ with $F_{\alpha t}=1$ if and only if the weak learner h_t uses feature f_α. Let $e_t>0$ be the cost to evaluate a $h_t(\cdot)$, and c_α be the cost to extract feature f_α. With this notation, we can formulate the total test-time cost for an instance precisely as

$$c(\boldsymbol{\beta})=\underbrace{\sum_t e_t\|\beta_t\|_0}_{\text{evaluation cost}}+\underbrace{\sum_\alpha c_\alpha\left\|\sum_t|F_{\alpha t}\beta_t|\right\|_0}_{\text{feature extraction cost}},\qquad(2)$$

where the l_0 norm for scalars is defined as $\|a\|_0\in\{0,1\}$ with $\|a\|_0=1$ if and only if $a\neq0$. The first term assigns cost e_t to every weak learner used in $\boldsymbol{\beta}$, the second term assigns cost c_α to every feature that is extracted by *at least one* of such weak learners.

Test-cost relaxation. The cost formulation in (2) is exact but difficult to optimize as the l_0 norms are non-continuous and non-differentiable. As a solution, throughout this paper we use the mixed-norm relaxation of the l_0 norm over sums,

$$\sum_j\left\|\sum_i|a_{ij}|\right\|_0\rightarrow\sum_j\sqrt{\sum_i(a_{ij})^2},\qquad(3)$$

described by (Kowalski, 2009). Note that for a single element this relaxation relaxes the l_0 norm to the l_1 norm, $\|a_{ij}\|_0\rightarrow\sqrt{(a_{ij})^2}=|a_{ij}|$, and recovers the commonly used approximation to encourage sparsity (Efron et al., 2004; Schölkopf & Smola, 2001). We plug the cost-term (2) into the loss in (1) and apply the relaxation (3) to all l_0 norms to obtain

$$\underbrace{\sum_i\ell_i+\rho|\boldsymbol{\beta}|}_{\text{regularized loss}}+\lambda\Bigg(\underbrace{\sum_t e_t|\beta_t|}_{\text{ev. cost penalty}}+\underbrace{\sum_\alpha c_\alpha\sqrt{\sum_t(F_{\alpha t}\beta_t)^2}}_{\text{feature cost penalty}}\Bigg),\qquad(4)$$

where we abbreviate $\ell_i=\ell(\phi(\mathbf{x}_i)^\top\boldsymbol{\beta},y_i)$ for simplicity. While (4) is cost-sensitive, it is restricted to a single linear classifier. In the next section we describe how to expand this formulation into a cost-effective tree-structured model.

4. Cost-sensitive tree

We begin by introducing foundational concepts regarding the CSTC tree and derive a global loss function (5). Similar to the previous section, we first derive the exact cost term and then relax it with the mixed-norm. Finally, we describe how to optimize this function efficiently and to undo some of the inaccuracy induced by the mixed-norm relaxations.

CSTC nodes. We make the assumption that instances with similar labels can utilize similar features.[1]

[1]For example, in web-search ranking, features generated by browser statistics are typically predictive only for highly relevant pages as they require the user to spend significant time on the page and interact with it.

$$\min_{\boldsymbol{\beta}^1,\theta^1,\ldots,\boldsymbol{\beta}^{|V|},\theta^{|V|}} \underbrace{\sum_{v^k\in V}\left(\frac{1}{n}\sum_{i=1}^{n} p_i^k \ell_i^k + \rho|\boldsymbol{\beta}^k|\right)}_{\text{regularized risk}} + \lambda \sum_{v^l\in L} p^l \Bigg[\underbrace{\sum_t e_t \sqrt{\sum_{v^j\in\pi^l}(\beta_t^j)^2}}_{\text{evaluation cost penalty}} + \underbrace{\sum_\alpha c_\alpha \sqrt{\sum_{v^j\in\pi^l}\sum_t (F_{\alpha t}\beta_t^j)^2}}_{\text{feature cost penalty}} \Bigg] \tag{5}$$

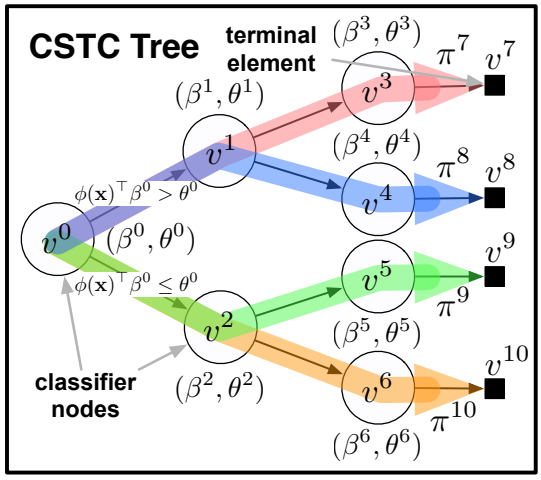

Figure 1. A schematic layout of a CSTC tree. Each node v^k has a threshold θ^k to send instances to different parts of the tree and a weight vector $\boldsymbol{\beta}^k$ for prediction. We solve for $\boldsymbol{\beta}^k$ and θ^k that best balance the accuracy/cost trade-off for the whole tree. All paths of a CSTC tree are shown in color.

We therefore design our tree algorithm to partition the input space based on classifier predictions. Classifiers that reside deep in the tree become experts for a small subset of the input space and intermediate classifiers determine the path of instances through the tree. We distinguish between two different elements in a CSTC tree (depicted in Figure 1): *classifier nodes* (white circles) and *terminal elements* (black squares). Each *classifier node* v^k is associated with a weight vector $\boldsymbol{\beta}^k$ and a threshold θ^k. Different from cascade approaches, these classifiers not only classify inputs using $\boldsymbol{\beta}^k$, but also branch them by their threshold θ^k, sending inputs to their upper child if $\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}^k > \theta^k$, and to their lower child otherwise. *Terminal elements* are "dummy" structures and are *not* classifiers. They return the predictions of their direct parent classifier nodes—essentially functioning as a placeholder for an exit out of the tree. The tree structure may be a full balanced binary tree of some depth (eg. figure 1), or can be pruned based on a validation set (eg. figure 4, left).

During test-time, inputs are first applied to the root node v^0. The root node produces predictions $\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}^0$ and sends the input $\mathbf{x}_i$ along one of two different paths, depending on whether $\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}^0 > \theta^0$. By repeatedly branching the test-inputs, classifier nodes sitting deeper in the tree only handle a small subset of all inputs and become specialized towards that subset of the input space.

4.1. Tree loss

We derive a single global loss function over all nodes in the CSTC tree.

Soft tree traversal. Training the CSTC tree with hard thresholds leads to a combinatorial optimization problem, which is NP-hard. Therefore, during training, we *softly* partition the inputs and assign *traversal probabilities* $p(v^k|\mathbf{x}_i)$ to denote the likelihood of input $\mathbf{x}_i$ traversing through node v^k. Every input $\mathbf{x}_i$ traverses through the root, so we define $p(v^0|\mathbf{x}_i) = 1$ for all i. We use the sigmoid function to define a soft belief that an input $\mathbf{x}_i$ will transition from classifier node v^k to its *upper* child v^j as $p(v^j|\mathbf{x}_i, v^k) = \sigma(\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}^k - \theta^k)$.[2] The probability of reaching child v^j from the root is, recursively, $p(v^j|\mathbf{x}_i) = p(v^j|\mathbf{x}_i, v^k)p(v^k|\mathbf{x}_i)$, because each node has exactly one parent. For a *lower* child v^l of parent v^k we naturally obtain $p(v^l|\mathbf{x}_i) = \big[1 - p(v^j|\mathbf{x}_i, v^k)\big]p(v^k|\mathbf{x}_i)$. In the following paragraphs we incorporate this probabilistic framework into the single-node risk and cost terms of eq. (4) to obtain the corresponding *expected* tree risk and tree cost.

Expected tree risk. The *expected tree risk* can be obtained byWg over all nodes V and inputs and weighing the risk $\ell(\cdot)$ of input $\mathbf{x}_i$ at node v^k by the probability $p_i^k = p(v^k|\mathbf{x}_i)$,

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{v^k\in V} p_i^k \ell(\phi(\mathbf{x}_i)^\top\boldsymbol{\beta}^k, y_i). \tag{6}$$

This has two effects: 1. the local risk for each node focusses more on likely inputs; 2. the global risk attributes more weight to classifiers that serve many inputs.

Expected tree costs. The cost of a test-input is the cumulative cost across all classifiers along its path through the CSTC tree. Figure 1 illustrates an exam-

[2]The sigmoid function is defined as $\sigma(a) = \frac{1}{1+exp(-a)}$ and takes advantage of the fact that $\sigma(a) \in [0,1]$ and that $\sigma(\cdot)$ is strictly monotonic.

ple of a CSTC tree with all paths highlighted in color. Every test-input must follow along exactly one of the paths from the root to a terminal element. Let L denote the set of all terminal elements (*e.g.*, in figure 1 we have $L=\{v^7, v^8, v^9, v^{10}\}$), and for any $v^l \in L$ let π^l denote the set of all *classifier nodes* along the unique path from the root v^0 before terminal element v^l (*e.g.*, $\pi^9 = \{v^0, v^2, v^5\}$). The evaluation and feature cost of this unique path is exactly

$$c^l = \underbrace{\sum_t e_t \left\| \sum_{v^j \in \pi^l} |\beta_t^j| \right\|_0}_{\text{evaluation cost}} + \underbrace{\sum_\alpha c_\alpha \left\| \sum_{v^j \in \pi^l} \sum_t |F_{\alpha t} \beta_t^j| \right\|_0}_{\text{feature cost}}.$$

This term is analogous to eq. (2), except the cost e_t of the weak learner h_t is paid if *any* of the classifiers v^j in path π^l use this tree (*i.e.* assign β_t^j non-zero weight). Similarly, the cost c_α of a feature f_α is paid exactly once if any of the weak learners of any of the classifiers along π^l require it. Once computed, a feature or weak learner can be reused by all classifiers along the path for free (as the computation can be cached very efficiently).

Given an input $\mathbf{x}_i$, the probability of reaching terminal element $v^l \in L$ (traversing along path π^l) is $p_i^l = p(v^l|\mathbf{x}_i)$. Therefore, the marginal probability that a training input (picked uniformly at random from the training set) reaches v^l is $p^l = \sum_i p(v^l|\mathbf{x}_i)p(\mathbf{x}_i) = \frac{1}{n}\sum_{i=1}^n p_i^l$. With this notation, the *expected cost* for an input traversing the CSTC tree becomes $\mathbb{E}[c^l] = \sum_{v^l \in L} p^l c^l$. Using our l_0-norm relaxation in eq. (3) on both l_0 norms in c^l gives the final expected tree cost penalty

$$\sum_{v^l \in L} p^l \left[\sum_t e_t \sqrt{\sum_{v^j \in \pi^l} (\beta_t^j)^2} + \sum_\alpha c_\alpha \sqrt{\sum_{v^j \in \pi^l} \sum_t (F_{\alpha t}\beta_t^j)^2} \right],$$

which naturally encourages weak learner and feature re-use along paths through the CSTC tree.

Optimization problem. We combine the risk (6) with the cost penalties and add the l_1-regularization term (which is unaffected by our probabilistic splitting) to obtain the global optimization problem (5). (We abbreviate the empiWisk at node v^k as $\ell_i^k = \ell(\phi(\mathbf{x}_i)^\top \boldsymbol{\beta}^k, y_i)$.)

4.2. Optimization Details

There are many techniques to minimize the loss in (5). We use a cyclic optimization procedure, solving $\frac{\partial \mathcal{L}}{\partial(\boldsymbol{\beta}^k, \theta^k)}$ for each classifier node v^k one at a time, keeping all other nodes fixed. For a given classifier node v^k, the traversal probabilities p_i^j of a descendant node v^j and the probability of an instance reaching a terminal element p^l also depend on $\boldsymbol{\beta}^k$ and θ^k (through its recursive definition) and must be incorporated into the gradient computation.

To minimize (5) with respect to parameters $\boldsymbol{\beta}^k, \theta^k$, we use the lemma below to overcome the non-differentiability of the square-root terms (and l_1 norms) resulting from the l_0-relaxations (3).

Lemma 1. *Given any* $g(x) > 0$, the following holds:

$$\sqrt{g(x)} = \min_{z>0} \frac{1}{2}\left[\frac{g(x)}{z} + z\right]. \tag{7}$$

The lemma can be proved as $z = \sqrt{g(x)}$ minimizes the function on the right hand side. Further, it is shown in (Boyd & Vandenberghe, 2004) that the right hand side is jointly convex in x and z, so long as $g(x)$ is convex.

For each square-root or l_1 term we introduce an auxiliary variable (i.e., z above) and alternate between minimizing the loss in (5) with respect to $\boldsymbol{\beta}^k, \theta^k$ and the auxiliary variables. The former is performed with conjugate gradient descent and the latter can be computed efficiently in closed form. This pattern of block-coordinate descent followed by a closed form minimization is repeated until convergence. Note that the loss is guaranteed to converge to a fixed point because each iteration decreases the loss function, which is bounded below by 0.

Initialization. The minimization of eq. (5) is non-convex and therefore initialization dependent. However, minimizing eq. (5) with respect to the parameters of leaf classifier nodes *is convex*, as the loss function, after substitutions based on lemma 1, becomes jointly convex (because of the lack of descendant nodes). We therefore initialize the tree top-to-bottom, starting at v^0, and optimize over $\boldsymbol{\beta}^k$ by minimizing (5) while considering all descendant nodes of v^k as "cut-off" (thus pretending node v^k is a leaf).

Tree pruning. To obtain a more compact model and to avoid overfitting, the CSTC tree can be pruned with the help of a validation set. As each node is a classifier, we can apply the CSTC tree on a validation set and compute the validation error at each node. We prune away nodes that, upon removal, do not decrease the performance of CSTC on the validation set (in the case of ranking data, we even can use validation NDCG as our pruning criterion).

Fine-tuning. The relaxation in (3) makes the exact l_0 cost terms differentiable and is well suited to approximate *which* dimensions in a vector $\boldsymbol{\beta}^k$ should be

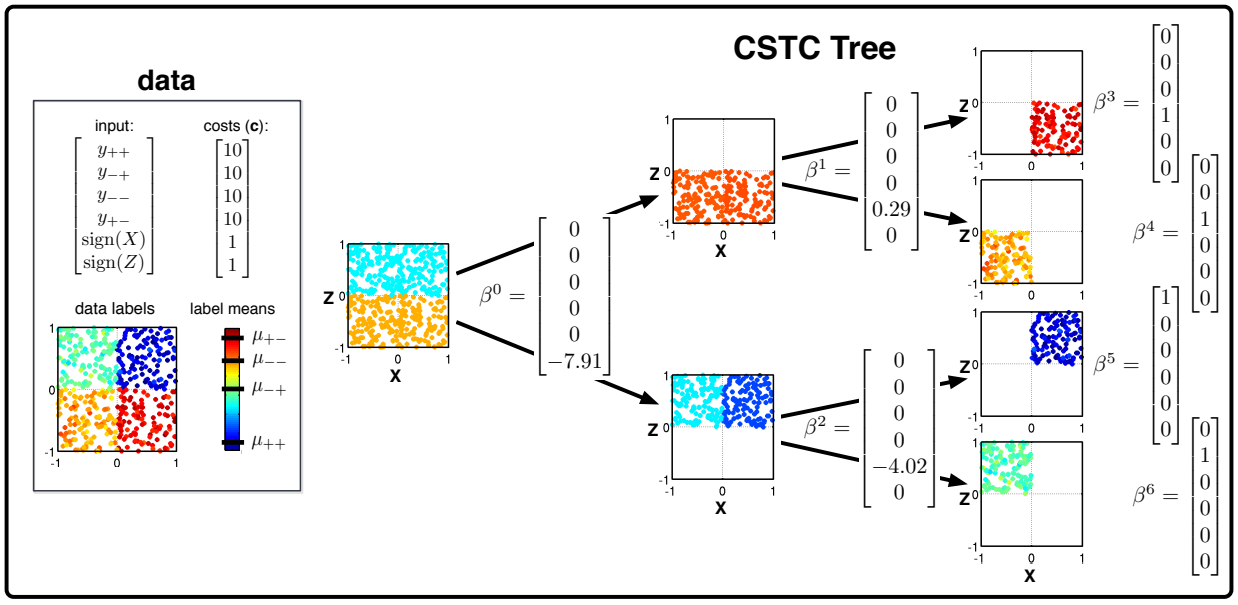

Figure 2. CSTC on synthetic data. The box at left describes the artificial data set. The rest of the figure shows the CSTC tree built for the data set. At each node we show a plot of the predictions made by that classifier. After each node we show the weight vector that was selected to make predictions and send instances to child nodes (if applicable).

assigned non-zero weights. The mixed-norm does however impact the performance of the classifiers because (different from the l_0 norm) larger weights in $\boldsymbol{\beta}$ incur larger penalties in the loss. We therefore introduce a post-processing step to correct the classifiers from this unwanted regularization effect. We re-optimize all *predictive* classifiers (classifiers with terminal element children, *i.e.* classifiers that make final predictions), while clamping all features with zero-weight to strictly remain zero.

$$\min_{\bar{\boldsymbol{\beta}}^k} \sum_i p_i^k \ell(\phi(\mathbf{x}_i)^\top \bar{\boldsymbol{\beta}}^k, y_i) + \rho|\bar{\boldsymbol{\beta}}^k|$$
$$\text{subject to: } \bar{\beta}_t^k = 0 \text{ if } \beta_t^k = 0. \tag{8}$$

The final CSTC tree uses these re-optimized weight vectors $\bar{\boldsymbol{\beta}}^k$ for all predictive classifier nodes v^k.

5. Results

In this section, we first evaluate CSTC on a carefully constructed synthetic data set to test our hypothesis that CSTC learns specialized classifiers that rely on different feature subsets. We then evaluate the performance of CSTC on the large scale Yahoo! Learning to Rank Challenge data set and compare it with state-of-the-art algorithms.

5.1. Synthetic data

We construct a synthetic regression dataset, sampled from the four quadrants of the X, Z-plane, where $X = Z = [-1, 1]$. The features belong to two categories: cheap features, $sign(x), sign(z)$ with cost $c=1$, which can be used to identify the quadrant of an input; and four expensive features $y_{++}, y_{+-}, y_{-+}, y_{--}$ with cost $c = 10$, which represent the exact label of an input if it is from the corresponding region (a random number otherwise). Since in this synthetic data set we do not transform the feature space, we have $\phi(\mathbf{x}) = \mathbf{x}$, and $\mathbf{F}$ (the weak learner feature-usage variable) is the 6×6 identity matrix. By design, a perfect classifier can use the two cheap features to identify the sub-region of an instance and then extract the correct expensive feature to make a perfect prediction. The minimum feature cost of such a perfect classifier is exactly $c=12$ per instance. The labels are sampled from Gaussian distributions with quadrant-specific means $\mu_{++}, \mu_{-+}, \mu_{+-}, \mu_{--}$ and variance 1. Figure 2 shows the CSTC tree and the predictions of test inputs made by each node. In every path along the tree, the first two classifiers split on the two cheap features and identify the correct sub-region of the input. The final classifier extracts a single expensive feature to predict the labels. As such, the mean squared error of the training and testing data both approach 0.

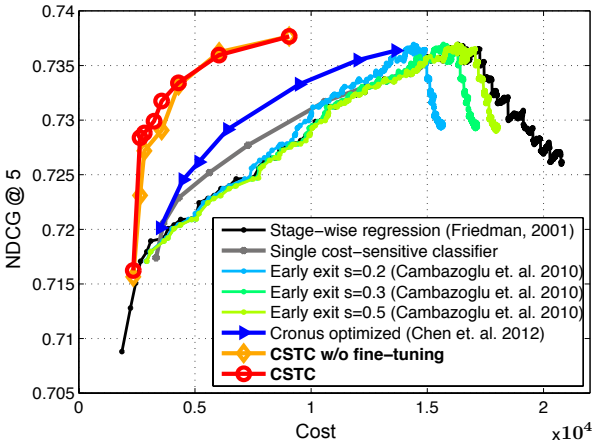

Figure 3. The test ranking accuracy (NDCG@5) and cost of various cost-sensitive classifiers. CSTC maintains its high retrieval accuracy significantly longer as the cost-budget is reduced. Note that fine-tuning does not improve NDCG significantly because, as a metric, it is *insensitive* to mean squared error.

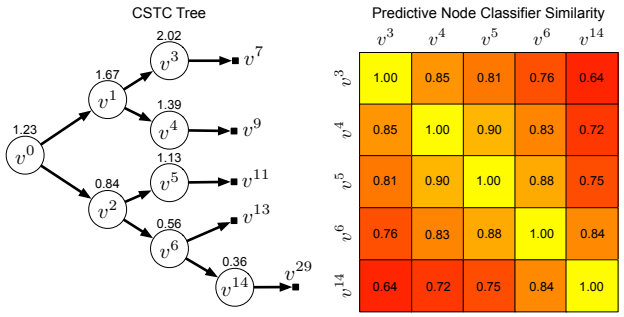

Figure 4. (*Left*) The pruned CSTC-tree generated from the Yahoo! Learning to Rank data set. (*Right*) Jaccard similarity coefficient between classifiers within the learned CSTC tree.

5.2. Yahoo! Learning to Rank

To evaluate the performance of CSTC on real-world tasks, we test our algorithm on the public Yahoo! Learning to Rank Challenge data set[3] (Chapelle & Chang, 2011). The set contains 19,944 queries and 473,134 documents. Each query-document pair $\mathbf{x}_i$ consists of 519 features. An extraction cost, which takes on a value in the set $\{1, 5, 20, 50, 100, 150, 200\}$, is associated with each feature[4]. The unit of these values is the time required to evaluate a weak learner $h_t(\cdot)$. The label $y_i \in \{4, 3, 2, 1, 0\}$ denotes the relevancy of a document to its corresponding query, with 4 indicating a perfect match. In contrast to Chen et al. (2012), we do not inflate the number of irrelevant documents (by counting them 10 times). We measure the performance using NDCG@5 (Järvelin & Kekäläinen, 2002), a preferred ranking metric when multiple levels of relevance are available. Unless otherwise stated, we restrict CSTC to a maximum of 10 nodes. All results are obtained on a desktop with two 6-core Intel i7 CPUs. Minimizing the global objective requires less than 3 hours to complete, and fine-tuning the classifiers takes about 10 minutes.

Comparison with prior work. Figure 3 shows a comparison of CSTC with several recent algorithms for test-time cost-sensitive learning. We show NDCG versus cost (in units of weak learner evaluations). The plot shows different stages in our derivation of CSTC: the initial cost-insensitive ensemble classifier $H'(\cdot)$ (Friedman, 2001) from section 3 (*stage-wise regression*), a *single cost-sensitive classifier* as described in eq. (4), the CSTC tree (5) and CSTC tree *with* fine-tuning (8). We obtain the curves by varying the accuracy/cost trade-off parameter λ (and perform early stopping based on the validation data, for fine-tuning). For CSTC tree we evaluate six settings, $\lambda = \{\frac{1}{3}, \frac{1}{2}, 1, 2, 3, 4, 5, 6\}$. In the case of *stage-wise regression*, which is not cost-sensitive, the curve is simply a function of boosting iterations.

For competing algorithms, we include *Early exit* (Cambazoglu et al., 2010) which improves upon stage-wise regression by short-circuiting the evaluation of unpromising documents at test-time, reducing the overall test-time cost. The authors propose several criteria for rejecting inputs early and we use the best-performing method "early exits using proximity threshold". For *Cronus* (Chen et al., 2012), we use a cascade with a maximum of 10 nodes. All hyper-parameters (cascade length, keep ratio, discount, early-stopping) were set based on a validation set. The cost/accuracy curve was generated by varying the corresponding trade-off parameter, λ.

As shown in the graph, CSTC significantly improves the cost/accuracy trade-off curve over all other algorithms. The power of Early exit is limited in this case as the test-time cost is dominated by feature extraction, rather than the evaluation cost. Compared with Cronus, CSTC has the ability to identify features that are most beneficial to different groups of inputs. It is this ability, which allows CSTC to maintain the high NDCG significantly longer as the cost-budget is reduced.

[3] http://learningtorankchallenge.yahoo.com

[4] The extraction costs were provided by a Yahoo! employee.

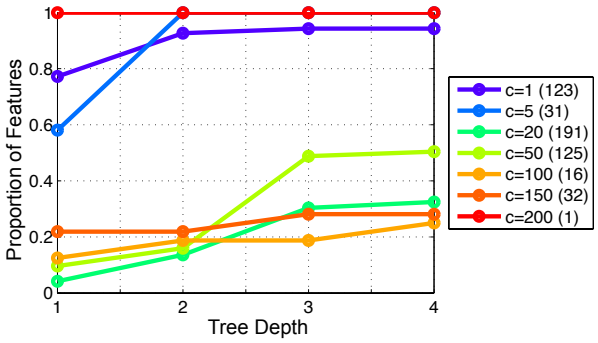

Figure 5. The ratio of features, grouped by cost, that are extracted at different depths of CSTC (the number of features in each cost group is indicated in parentheses in the legend). More expensive features ($c \geq 20$) are gradually extracted as we go deeper.

Note that CSTC with fine-tuning only achieves very tiny improvement over CSTC without it. Although the fine-tuning step decreases the mean squared error on the test-set, it has little effect on NDCG, which is only based on the relative ranking of the documents (as opposed to their exact predictions). Moreover, because we fine-tune prediction nodes until validation NDCG decreases, for the majority of λ values, only a small amount of fine-tuning occurs.

Input space partition. Figure 4 (*left*) shows a pruned CSTC tree ($\lambda = 4$) for the Yahoo! data set. The number above each node indicates the average label of theWg inputs passing through that node. We can observe that different branches aim at different parts of the input domain. In general, the upper branches focus on correctly classifying higher ranked documents, while the lower branches target low-rank documents. Figure 4 (*right*) shows the Jaccard matrix of the predictive classifiers ($v^3, v^4, v^5, v^6, v^{14}$) from the same CSTC tree. The matrix shows a clear trend that the Jaccard coefficients decrease monotonically away from the diagonal. This indicates that classifiers share fewer features in common if their average labels are further apart—the most different classifiers v^3 and v^{14} have only 64% of their features in common—and validates that classifiers in the CSTC tree extract different features in different regions of the tree.

Feature extraction. We also investigate the features extracted in individual classifier nodes. Figure 5 shows the fraction of features, with a particular cost, extracted at different depths of the CSTC tree for the Yahoo! data. We observe a general trend that as depth increases, more features are being used. However, cheap features ($c \leq 5$) are fully extracted early-on, whereas expensive features ($c \geq 20$) are extracted by classifiers sitting deeper in the tree, where each individual classifier only copes with a small subset of inputs. The expensive features are used to classify these subsets of inputs more precisely. The only feature that has cost 200 is extracted at all depths—which seems essential to obtain high NDCG (Chen et al., 2012).

6. Conclusions

We introduce Cost-Sensitive Tree of Classifiers (CSTC), a novel learning algorithm that explicitly addresses the trade-off between accuracy and expected test-time CPU cost in a principled fashion. The CSTC tree partitions the input space into sub-regions and identifies the most cost-effective features for each one of these regions—allowing it to match the high accuracy of the state-of-the-art at a small fraction of the cost. We obtain the CSTC algorithm by formulating the expected test-time cost of an instance passing through a tree of classifiers and relax it into a continuous cost function. This cost function can be minimized while learning the parameters of all classifiers in the tree jointly. By making the test-time cost vs. accuracy tradeoff explicit we enable high performance classifiers that fit into computational budgets and can reduce unnecessary energy consumption in large-scale industrial applications. Further, engineers can design highly specialized features for particular edges-cases of their input domain and CSTC will automatically incorporate them on-demand into its tree structure.

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

References

Bengio, S., Weston, J., and Grangier, D. Label embedding trees for large multi-class tasks. *NIPS*, 23:163–171, 2010.

Boyd, S.P. and Vandenberghe, L. *Convex optimization*. Cambridge Univ Pr, 2004.

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., 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.

Deng, J., Satheesh, S., Berg, A.C., and Fei-Fei, L. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011.

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.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. *The Annals of Statistics*, 32(2): 407–499, 2004.

Fleck, M., Forsyth, D., and Bregler, C. Finding naked people. *ECCV*, pp. 593–602, 1996.

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. 2011.

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

Järvelin, K. and Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. *ACM TOIS*, 20(4):422–446, 2002.

Jordan, M.I. and Jacobs, R.A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6 (2):181–214, 1994.

Karayev, S., Baumgartner, T., Fritz, M., and Darrell, T. Timely object recognition. In *Advances in Neural Information Processing Systems 25*, pp. 899–907, 2012.

Kowalski, M. Sparse regression using mixed norms. *Applied and Computational Harmonic Analysis*, 27(3):303–324, 2009.

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

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

Saberian, M. and Vasconcelos, N. Boosting classifier cascades. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R.S., and Culotta, A. (eds.), *NIPS*, pp. 2047–2055. 2010.

Schölkopf, B. and Smola, A.J. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

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 *Advances in Neural Information Processing Systems 25*, 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.

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

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.