# Feature-Cost Sensitive Learning with Submodular Trees of Classifiers

**Matt J. Kusner, Wenlin Chen, Quan Zhou, Zhixiang (Eddie) Xu, Kilian Q. Weinberger, Yixin Chen**
Washington University in St. Louis, 1 Brookings Drive, MO 63130
Tsinghua University, Beijing 100084, China
{mkusner, wenlinchen, zhixiang.xu, kilian, ychen25}@wustl.edu
zhouq10@mails.tsinghua.edu.cn

## Abstract

During the past decade, machine learning algorithms have become commonplace in large-scale real-world industrial applications. In these settings, the computation time to train and test machine learning algorithms is a key consideration. At training-time the algorithms must scale to very large data set sizes. At testing-time, the cost of feature extraction can dominate the CPU runtime. Recently, a promising method was proposed to account for the feature extraction cost at testing time, called Cost-sensitive Tree of Classifiers (CSTC). Although the CSTC problem is NP-hard, the authors suggest an approximation through a mixed-norm relaxation across many classifiers. This relaxation is slow to train and requires involved optimization hyperparameter tuning. We propose a different relaxation using approximate submodularity, called Approximately Submodular Tree of Classifiers (ASTC). ASTC is much simpler to implement, yields equivalent results but requires no optimization hyperparameter tuning and is up to *two orders of magnitude faster to train*.

## Introduction

Machine learning is rapidly moving into real-world industrial settings with widespread impact. Already, it has touched ad placement (Bottou et al. 2013), web-search ranking (Zheng et al. 2008), large-scale threat detection (Zhang et al. 2013), and email spam classification (Weinberger et al. 2009). This shift, however, often introduces *resource constraints* on machine learning algorithms. Specifically, Google reported an average of 5.9 billion searches executed per day in 2013. Running a machine learning algorithm is impractical unless it can generate predictions in tens of milliseconds. Additionally, a large scale corporation may want to limit the amount of carbon consumed through electricity use. Taking into account the needs of the learning setting is a step machine learning must make if it is to be widely adopted outside of academic settings.

In this paper we focus on the resource of *test-time CPU cost*. This test-time cost is the total cost to *evaluate* the classifier and to *extract features*. In industrial settings, where linear models are prevalent, the *feature extraction cost* dominates the test-time CPU cost. Typically, this cost must be strictly within budget *in expectation*. For example, an email spam classifier may have 10 milliseconds per email, but can spend more time on difficult cases as long as it can be faster on easier ones. A key insight to facilitate this flexibility is to extract different features for different test instances.

Recently, a model for the feature-cost efficient learning setting was introduced that demonstrates impressive test-time cost reductions for web search (Xu et al. 2014). This model, called *Cost-Sensitive Tree of Classifiers* (CSTC), partitions test inputs into subsets using a tree structure. Each node in the tree is a *sparse* linear classifier that decides what subset a test input should fall into, by passing these inputs along a path through the tree. Once an input reaches a leaf node it is classified by a specialized leaf classifier. By traversing the tree along different paths, different features are extracted for different test inputs. While CSTC demonstrates impressive real-world performance, the training time is non-trivial and the optimization procedure involves several sensitive hyperparameters.

Here, we present a simplification of CSTC, that takes advantage of approximate submodularity to train a near optimal tree of classifiers. We call our model *Approximately Submodular Tree of Classifiers* (ASTC). We make a number of novel contributions: 1. We show how the CSTC optimization can be formulated as an approximately submodular set function optimization problem, which can be solved greedily. 2. We reduce the training complexity of each classifier within the tree to scale linearly with the training set size. 3. We demonstrate on several real-world datasets that ASTC matches CSTC's cost/accuracy trade-offs, yet it is significantly quicker to implement, requires no additional hyperparameters for the optimization and is up to *two orders of magnitude* faster to train.

## Resource-efficient learning

In this section we formalize the general setting of resource-efficient learning and introduce our notation. We are given a training dataset with inputs and class labels $\{(\mathbf{x}_i, y_i)\}_{i=1}^n = (\mathbf{X}, \mathbf{y}) \in \mathcal{R}^{(n,d)} \times \mathcal{Y}^n$ (in this paper we limit our analysis to the regression case). Without loss of generality we assume for each feature vector $\mathbf{x}_f \in \mathcal{R}^n$ that $\|\mathbf{x}_\alpha\|_2 = 1$, for all $\alpha = 1, \ldots, d$, and that $\|\mathbf{y}\|_2 = 1$, all with zero mean. Our aim is to learn a linear classifier $h_{\boldsymbol{\beta}}(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\beta}$ that generalizes well to unseen data *and* uses the available resources as effectively as possible. We assume that each feature $\alpha$ incurs a cost of

$c(\alpha)$ during *extraction*. Additionally, let $\ell(\cdot)$ be a convex loss function and let $B$ be a budget on its feature cost. Formally, we want to solve the following optimization problem,

$$\min_{\boldsymbol{\beta}} \ \ell(\mathbf{y}; \mathbf{X}, \boldsymbol{\beta}) \ \text{ subject to } \sum_{\alpha:|\boldsymbol{\beta}|>0} c(\alpha) \leq B, \quad (1)$$

where $\sum_{\alpha:|\boldsymbol{\beta}|>0} c(\alpha)$ sums over the (used) features with non-zero weight in $\boldsymbol{\beta}$.

### Cost-sensitive tree of classifiers (CSTC)

Recent work in resource-budgeted learning (Xu et al. 2014) (CSTC) shows impressive results by learning multiple classifiers from eq. (1) which are arranged in a tree (depth $D$) $\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^{2^D-1}$. The CSTC model is shown in figure 1 (throughout the paper we consider the linear classifier version of CSTC). Each node $v^k$ is a classifier whose predictions $\mathbf{x}^\top \boldsymbol{\beta}^k$ for an input $\mathbf{x}$ are thresholded by $\theta^k$. The threshold decides whether to send $\mathbf{x}$ to the upper or lower child of $v^k$. An input continues through the tree in this way until arriving at a leaf node, which predicts its label.

**Combinatorial optimization.** There are two road-blocks to learning the CSTC classifiers $\boldsymbol{\beta}^k$. First, because instances traverse different paths through the tree, the optimization is a complex combinatorial problem. In (Xu et al. 2014), they fix this by probabilistic tree traversal. Specifically, each classifier $\mathbf{x}^\top \boldsymbol{\beta}^k$ is trained using *all* instances, weighted by the probability that instances reach $v^k$. This probability is derived by squashing node predictions using the sigmoid function: $\sigma(\mathbf{x}^\top \boldsymbol{\beta}) = 1/(1+\exp(-\mathbf{x}^\top \boldsymbol{\beta}))$.

To make the optimization more amenable to gradient-based methods, (Xu et al. 2014) convert the constrained optimization problem in eq. (1) to the Lagrange equivalent and minimize the *expected classifier loss* plus the *expected feature cost*, where the expectation is taken over the probability of an input reaching node $v^k$,

$$\min_{\boldsymbol{\beta}^k} \underbrace{\frac{1}{n}\sum_{i=1}^{n} p_i^k (y_i - \mathbf{x}_i^\top \boldsymbol{\beta}^k)^2}_{\text{exp. squared loss}} + \rho\|\boldsymbol{\beta}^k\|_1 + \lambda \underbrace{\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)]}_{\text{exp. feature cost}}. \quad (2)$$

Here $p_i^k$ is the probability that instance $\mathbf{x}_i$ traverses to $v^k$ and $\rho$ is the regularization constant to control overfitting. The last term is the expected feature cost of $\boldsymbol{\beta}^k$,

$$\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)] = \sum_{v^l \in \mathcal{P}^k} p^l \left[ \sum_{\alpha} c(\alpha) \left\| \sum_{v^j \in \pi^l} |\beta_\alpha^j| \right\|_0 \right]. \quad (3)$$

The outer-most sum is over all leaf nodes $\mathcal{P}^k$ affected by $\boldsymbol{\beta}^k$ and $p^l$ is the probability of any input reaching such a leaf node $v^l$. The remaining terms describe the cost incurred for an input traversing to that leaf node. CSTC makes the assumption that, once a feature is extracted for an instance *it is free for future requests*. Therefore, CSTC sums over all features $\alpha$ and if any classifier along the path to leaf node $v^l$ uses feature $\alpha$, it is paid for exactly once ($\pi^l$ is the set of nodes on the path to $v^l$).

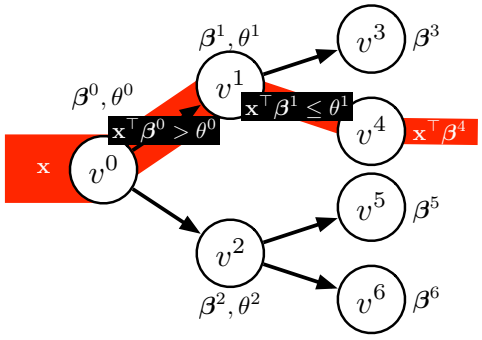


Figure 1: The CSTC tree (depth 3). Instances $\mathbf{x}$ are sent along a path through the tree (*e.g.*, in red) based on the predictions of node classifiers $\boldsymbol{\beta}^k$. If predictions are above a threshold $\theta^k$, $\mathbf{x}$ is sent to an upper child node, otherwise it is sent to a lower child. The leaf nodes predict the class of $\mathbf{x}$.

**$\ell_0$ norm and differentiability.** The second optimization road-block is that this feature cost term is non-continuous, and is thus hard to optimize. Their solution is to derive a continuous relaxation of the $\ell_0$ norm using the mixed-norm (Kowalski 2009). The final optimization is non-covex and not differentiable and the authors present a variational approach, introducing auxiliary variables for both $\ell_0$ and $\ell_1$ norms so that the optimization can be solved with cyclic block-coordinate descent.

There are a number of practical difficulties that arise when using CSTC for a given dataset. First, optimizing a non-leaf node in the CSTC tree affects all descendant nodes via the instance probabilities. This slows the optimization and is difficult to implement. Second, the optimization is sensitive to gradient learning rates and convergence thresholds, which require careful tuning. In the same vein, selecting appropriate ranges for hyperparameters $\lambda$ and $\rho$ may take repeated trial runs. Third, because CSTC needs to reoptimize all classifier nodes the training time is non-trivial for large datasets, making hyperparameter tuning on a validation set time-consuming. Additionally, the stationary point reached by block coordinate descent is initialization dependent. These difficulties may serve as significant barriers to entry, potentially preventing practitioners from using CSTC.

## Model

In this paper we propose a vastly simplified variant of the CSTC classifier, called Approximately Submodular Tree of Classifiers (*ASTC*). Instead of relaxing the expected cost term into a continuous function, we reformulate the entire optimization as an *approximately submodular set function optimization problem*.

**ASTC nodes.** We begin by considering an individual classifier $\boldsymbol{\beta}^k$ in the CSTC tree, optimized using eq. (2). If we ignore the effect of $\boldsymbol{\beta}^k$ on descendant leaf nodes $\mathcal{P}^k$ and previous nodes on its path $\pi^k$, the feature cost changes:

$$\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)] = \sum_{\alpha} c(\alpha)\|\beta_\alpha^k\|_0. \quad (4)$$

This combined with the loss term is simply a weighted classifier with cost-weighted $\ell_0$-regularization. We propose to *greedily* select features based on their performance/cost trade-off and to build the tree of classifiers top-down, starting from the root. We will solve one node at a time and set features 'free' that are used by parent nodes (as they need not be extracted twice). Figure 2 shows a schematic of the difference between the optimization of ASTC and the reoptimization of CSTC.

**Resource-constrained submodular optimization.** An alternative way to look at the optimization of a single CSTC node is as an optimization over sets of features. Let $\Omega$ be the set of all features. Define the loss function for node $v^k$, $\ell_k(A)$, over a set of features $A \subseteq \Omega$ as such,

$$\ell_k(A) = \min_{\boldsymbol{\beta}^k} \frac{1}{n} \sum_{i=1}^{n} p_i^k (y_i - \delta_A(\mathbf{x}_i)^\top \boldsymbol{\beta}^k)^2 \qquad (5)$$

where we treat probabilities $p_i^k$ as indicator weights: $p_i^k = 1$ if input $\mathbf{x}_i$ is sent to $v^k$, and to 0 otherwise. Define $\delta_A(\mathbf{x})$ as an element-wise feature indicator function that returns feature $x_a$ if $a \in A$ and 0 otherwise. Thus, $\ell_k(\cdot)$ is the squared loss of the optimal model using only (a) inputs that reach $v^k$ and (b) the features in set $A$. Our goal is to select a set of features $A$ that have low cost, and simultaneously have a low optimal loss $\ell_k(A)$.

Certain problems in constrained set function optimization have very nice properties. Particularly, a class of set functions, called *submodular* set functions, have been shown to admit simple near-optimal greedy algorithms (Nemhauser, Wolsey, and Fisher 1978). For the resource-constrained case, each feature (set element) $a$ has a certain resource cost $c(a)$, and we would like to ensure that the cost of selected features fall under some resource budget $B$. For a submodular function $s$ that is non-decreasing and non-negative the resource-constrained set function optimization,

$$\max_{A \subseteq \Omega} s(A) \;\text{ subject to }\; \sum_{a \in A} c(a) \leq B \qquad (6)$$

can be solved near-optimally by greedily selecting set elements $a \in \Omega$ that maximize $s$ as such,

$$g_j = \operatorname*{argmax}_{a \in \Omega} \left[ \frac{s(G_{j-1} \cup a) - s(G_{j-1})}{c(a)} \right]. \qquad (7)$$

Where we define the greedy ordering $G_{j-1} = (g_1, g_2, \ldots, g_{j-1})$. To find $g_j$ we evaluate all remaining set elements $a \in \Omega \setminus G_{j-1}$ and pick the element $g_j = \hat{a}$ for which $s(G_{j-1} \cup a)$ increases the most over $s(G_{j-1})$ *per cost*. Let $G_{\langle T \rangle} = (g_1, \ldots, g_m)$ be the largest feasible greedy set, having total cost $T$ (*i.e.*, $\sum_{j=1}^{m} c(g_j) = T \leq B$ and $T + c(g_{m+1}) > B$). Streeter and Golovin (2007) prove that for any non-decreasing and non-negative submodular function $s$ and some budget $B$, eq. (7) gives an approximation ratio of $(1 - e^{-1}) \approx 0.63$ with respect to the optimal set with cost $T \leq B$. Call this set $\mathcal{C}^*_{\langle T \rangle}$. Then, $s(G_{\langle T \rangle}) \geq (1 - e^{-1}) s(\mathcal{C}^*_{\langle T \rangle})$ for the optimization in eq. (6).

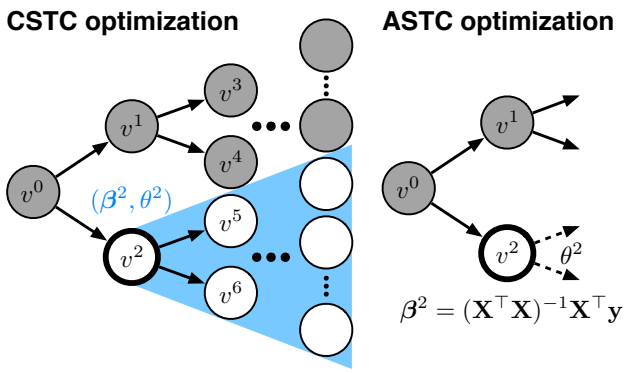


Figure 2: The optimization schemes of CSTC and ASTC. *Left:* When optimizing the classifier and threshold of node $v^2$, $(\boldsymbol{\beta}^2, \theta^2)$ in CSTC, it affects all of the descendant nodes (highlighted in blue). If the depth of the tree is large (*i.e.*, larger than 3), this results in a complex and expensive gradient computation. *Right:* ASTC on the other hand optimizes each node greedily using the familiar ordinary least squares closed form solution (shown above). $\theta^2$ is set by binary search to send half of the inputs to each child node.

## Optimization

In this section we demonstrate that optimizing a single CSTC node, and hence the CSTC tree, greedily is *approximately submodular*. We begin by introducing a modification to the set function (5). We then connect this to the approximation ratio of the greedy cost-aware algorithm eq. (7), demonstrating that it produces near-optimal solutions. The resulting optimization is very simple to implement and is described in Algorithm 1.

**Approximate submodularity.** To make $\ell_k(\cdot)$ amenable to resource-constrained set function optimization (6) we convert the loss minimization problem into an equivalent label 'fit' maximization problem. Define the set function $z_k$,

$$z_k(\mathcal{A}) = \frac{\text{Var}(\mathbf{y}; p^k) - \ell_k(A)}{\text{Var}(\mathbf{y}; p^k)} \qquad (8)$$

where $\text{Var}(\mathbf{y}; p^k) = \sum_i p_i^k (y_i - \bar{y})^2$ is the variance of the training label vector $\mathbf{y}$ multiplied by 0/1 probabilities $p_i^k$ ($\bar{y}$ is the mean predictor). It is straightforward to show that maximizing $z_k(\cdot)$ is equivalent to minimizing $\ell_k(\cdot)$. In fact, the following approximation guarantees hold for $z_k(\cdot)$ constructed from a wide range of loss functions (via a modification of Grubb and Bagnell (2012)). As we are interested in developing a new method for CSTC training, we focus purely on the squared loss. Note that $z_k(\cdot)$ is always non-negative (as the mean predictor is a worse training set predictor than the OLS solution using one feature, assuming it takes on more than one value). To see that it is also non-decreasing note that $z_k(\cdot)$ is precisely the squared multiple correlation $R^2$ (Diekhoff 1992), (Johnson and Wichern 2002), which is known to be non-decreasing.

If the features are orthogonal then $z_k(\cdot)$ is submodular (Krause and Cevher 2010). However, if this is not the case

it can be shown that $z_k(\cdot)$ is *approximately submodular* and has a *submodularity ratio*, defined as such:

**Definition 2.1 (Submodularity Ratio)** (Das and Kempe 2011). *Any non-negative set function $z(\cdot)$ has a submodularity ratio $\gamma_{U,i}$ for a set $U \subseteq \Omega$ and $i \geq 1$,*

$$\sum_{s \in S} \Big[ z(L \cup \{s\}) - z(L) \Big] \geq \gamma_{U,i} \Big[ z(L \cup S) - z(L) \Big],$$

*for all $L, S \subset U$, such that $|S| \leq i$ and $S \cap L = \emptyset$.*

The submodularity ratio ranges from 0 ($z(\cdot)$ is not submodular) to 1 ($z(\cdot)$ is submodular) and measures how close a function $z(\cdot)$ is to being submodular. As is done in Grubb and Bagnell (2012), for simplicity we will only consider the smallest submodularity ratio $\gamma \leq \gamma_{U,i}$ for all $U$ and $i$.

The submodularity ratio in general is non-trivial to compute. However, we can take advantage of work by Das and Kempe (2011) who show that the submodularity ratio of $z_k(\cdot)$, defined in eq. (8), is further bounded. Define $\mathbf{C}_A^k$ as the covariance matrix of $\tilde{\mathbf{X}}$, where $\tilde{\mathbf{x}}_i = p_i^k \delta_A(\mathbf{x}_i)$ (inputs weighted by the probability of reaching $v^k$, using only the features in $A$). Das and Kempe (2011) show in Lemma 2.4 that for $z_k(\cdot)$, it holds that $\gamma \geq \lambda_{min}(\mathbf{C}_A^k)$, where $\lambda_{min}(\mathbf{C}_A^k)$ is the minimum eigenvalue of $\mathbf{C}_A^k$.

**Approximation ratio.** As in the submodular case, we can optimize $z_k(\cdot)$ subject to the resource constraint that the cost of selected features must total less than a resource budget $B$. This optimization can be done greedily using the rule described in eq. (7). The following theorem—which is proved for any non-decreasing, non-negative, approximately submodular set function in Grubb and Bagnell (2012)—gives an approximation ratio for this greedy rule.

**Theorem 2.2** (Grubb and Bagnell 2012). *The greedy algorithm selects an ordering $G_{\langle T \rangle}$ such that,*

$$z_k(G_{\langle T \rangle}) > (1 - e^{-\gamma}) z_k(\mathcal{C}^*_{\langle T \rangle})$$

*where $G_{\langle T \rangle} = (g_1, g_2, \ldots, g_m)$ is the greedy sequence truncated at cost $T$, such that $\sum_{i=1}^m c(g_i) = T \leq B$ and $\mathcal{C}^*_{\langle T \rangle}$ is the set of optimal features having cost $T$.*

Thus, the approximation ratio depends directly on the submodularity ratio of $z_k(\cdot)$. For each node in the CSTC tree we greedily select features using the rule described in (7). If we are not at the root node, we set the cost of features used by the parent of $v^k$ to 0, and select them immediately (as we have already paid their cost). We fix a new-feature budget $B$—identical for each node in the tree—and then greedily select new features up to cost $B$ for each node. By setting probabilities $p_i^k$ to 0 or 1 depending on if $\mathbf{x}_i$ traverses to $v^k$, learning each node is like solving a unique approximately submodular optimization problem, using only the inputs sent to that node. Finally, we set node thresholds $\theta^k$ to send half of the training inputs to each child node.

We call our approach Approximately Submodular Tree of Classifiers (ASTC), which is shown in Algorithm 1. The optimization is much simpler than CSTC.

**Algorithm 1** ASTC in pseudo-code.

Inputs: $\{\mathbf{X}, \mathbf{y}\}$; tree depth $D$; node budget $B$, costs $c$
Set the initial costs $c^1 = c$
**for** $k = 1$ **to** $2^D - 1$ nodes **do**
  $G^k = \emptyset$
  **while** budget not exceeded: $\sum_{g \in G^k} c(g) \leq B$ **do**
    Select feature $a \in \Omega$ via eq. (7)
    $G^k = G^k \cup \{a\}$
  **end while**
  Solve $\boldsymbol{\beta}^k$ using weighted ordinary least squares
  **if** $v^k$ is not a leaf node, with children $v^l$ and $v^u$ **then**
    Set child probabilities:

$$p_i^u = \begin{cases} 1 & \text{if } p_i^k > \theta^k \\ 0 & \text{otherwise} \end{cases} \quad p_i^l = \begin{cases} 1 & \text{if } p_i^k \leq \theta^k \\ 0 & \text{otherwise} \end{cases}$$

    Set new feature costs: $c^u = c^l = c^k$
    Free used features: $c^u(G^k) = c^l(G^k) = 0$
  **end if**
**end for**
Return $\{\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots \boldsymbol{\beta}^{2^D - 1}\}$

## Fast greedy selection

Equation (7) requires solving an ordinary least squares problem, eq. (5), when selecting the feature that improves $z_k(\cdot)$ the most. This requires a matrix inversion which typically takes $\mathrm{O}(d^3)$ time. However, because we only consider selecting one feature at a time we can avoid the inversion for $z_k(\cdot)$ altogether using the QR decomposition. Let $G_j = (g_1, g_2, \ldots, g_j)$ be our current set of greedily-selected features. For simplicity let $\mathbf{X}_{G_j} = \delta_{G_j}(\mathbf{X})$, the data masked so that only features in $G_j$ are non-zero. Computing $z_k(\cdot)$ requires computing the weighted squared loss, eq. (5), which, after the QR decomposition requires no inverse. Redefine $\mathbf{x}_i = \sqrt{\frac{p_i^k}{n}} \mathbf{x}_i$ and $y_i = \sqrt{\frac{p_i^k}{n}} y_i$, then we have,

$$\ell_k(G_j) = \min_{\boldsymbol{\beta}^k} (\mathbf{y} - \mathbf{X}_{G_j} \boldsymbol{\beta}^k)^\top (\mathbf{y} - \mathbf{X}_{G_j} \boldsymbol{\beta}^k). \qquad (9)$$

Let $\mathbf{X}_{G_j} = \mathbf{QR}$ be the QR decomposition of $\mathbf{X}_{G_j}$. Plugging in this decomposition, taking the gradient of $\ell_k(G_j)$ with respect to $\boldsymbol{\beta}^k$, and solving at 0 yields (Hastie, Tibshirani, and Friedman 2001),

$$\boldsymbol{\beta}^k = \mathbf{R}^{-1} \mathbf{Q}^\top \mathbf{y}$$

The squared loss for the optimal $\boldsymbol{\beta}^k$ is,

$$\begin{aligned} \ell_k(G_j) &= (\mathbf{y} - \mathbf{QRR}^{-1}\mathbf{Q}^\top \mathbf{y})^\top (\mathbf{y} - \mathbf{QRR}^{-1}\mathbf{Q}^\top \mathbf{y}) \\ &= (\mathbf{y} - \mathbf{QQ}^\top \mathbf{y})^\top (\mathbf{y} - \mathbf{QQ}^\top \mathbf{y}) \\ &= \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{QQ}^\top \mathbf{y}. \end{aligned} \qquad (10)$$

Imagine we have extracted $j$ features and we are considering selecting a new feature $a$. The immediate approach would be to recompute $\mathbf{Q}$ including this feature and then recompute the squared loss (10). However, computing $\mathbf{q}_{j+1}$ (the column corresponding to feature $a$) can be done incrementally

using the Gram–Schmidt process:

$$\mathbf{q}_{j+1} = \frac{\mathbf{X}_a - \sum_{m=1}^{j}(\mathbf{X}_a^\top \mathbf{q}_m)\mathbf{q}_m}{\|\mathbf{X}_a - \sum_{m=1}^{j}(\mathbf{X}_a^\top \mathbf{q}_m)\mathbf{q}_m\|_2} = \frac{\mathbf{X}_a - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}_a}{\|\mathbf{X}_a - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}_a\|_2}$$

where $\mathbf{q}_1 = \mathbf{X}_{g_1}/\|\mathbf{X}_{g_1}\|_2$ (recall $g_1$ is the first greedily-selected feature). Finally, in order to select the best next feature using eq. (7), for each feature $a$ we must compute,

$$\frac{z_k(G_j \cup a) - z_k(G_j)}{c(a)} = \frac{-\ell_k(G_j \cup a) + \ell_k(G_j)}{\mathrm{Var}(\mathbf{y}; \mathbf{p}^k)c(a)} = \frac{\mathbf{y}^\top \mathbf{Q}_{1:j+1}\mathbf{Q}_{1:j+1}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{Q}\mathbf{Q}^\top \mathbf{y}}{\mathrm{Var}(\mathbf{y}; \mathbf{p}^k)c(a)} = \frac{(\mathbf{q}_{j+1}^\top \mathbf{y})^2}{\mathrm{Var}(\mathbf{y}; \mathbf{p}^k)c(a)} \quad (11)$$

where $\mathbf{Q}_{1:j+1} = \big[\mathbf{Q}, \mathbf{q}_{j+1}\big]$. The first two equalities follow from the definitions of $z_k(\cdot)$ and $\ell_k(\cdot)$. The third equality follows because $\mathbf{Q}$ and $\mathbf{Q}_{1:j+1}$ are orthogonal matrices.

We can compute all of the possible $\mathbf{q}_{j+1}$ columns, corresponding to all of the remaining features $a$ in parallel, call this matrix $\mathbf{Q}_{\text{remain}}$. Then we can compute eq. (11) vector-wise on $\mathbf{Q}_{\text{remain}}$ and select the feature with the largest corresponding value of $z_k(\cdot)$.

**Complexity.** Computing the ordinary least squares solution the naive way for the $(j+1)^{th}$ feature: $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$ requires $\mathrm{O}\big(n(j+1)^2 + (j+1)^3\big)$ for the covariance multiplication and inversion. This must be done $d-j$ times to compute $z_k(\cdot)$ for every remaining feature. Using the QR decomposition, computing $\mathbf{q}_{j+1}$ requires $\mathrm{O}\big(nj\big)$ time and computing eq. (11) takes $\mathrm{O}\big(n\big)$ time. As before, this must be done $d-j$ times for all remaining features, but as mentioned above both steps can be done in parallel.

## Results

In this section, we evaluate our approach on a real-world feature-cost sensitive ranking dataset: the Yahoo! Learning to Rank Challenge dataset. We begin by describing the dataset and show Precision@5 per cost compared against CSTC (Xu et al. 2014) and another cost-sensitive baseline. We then present results on a diverse set of non-cost sensitive datasets, demonstrating the flexibility of our approach. For all datasets we evaluate the training times of our approach compared to CSTC for varying tree budgets.

**Yahoo! learning to rank.** To judge how well our approach performs in a particular real-world setting, we test ASTC on the binary Yahoo! Learning to Rank Challenge data set (Chen et al. 2012). The dataset consists of $473,134$ web documents and $19,944$ queries. Each input $\mathbf{x}_i$ is a query-document pair containing $519$ features, each with extraction costs in the set $\{1, 5, 20, 50, 100, 150, 200\}$. The unit of cost is in weak-learner evaluations (i.e., the most expensive feature takes time equivalent to 200 weak-learner evaluations). We remove the mean and normalize the features by their $\ell_2$ norm, as is assumed by the submodularity ratio bound analysis. We use the Precision@5 metric, which is often used for binary ranking datasets.

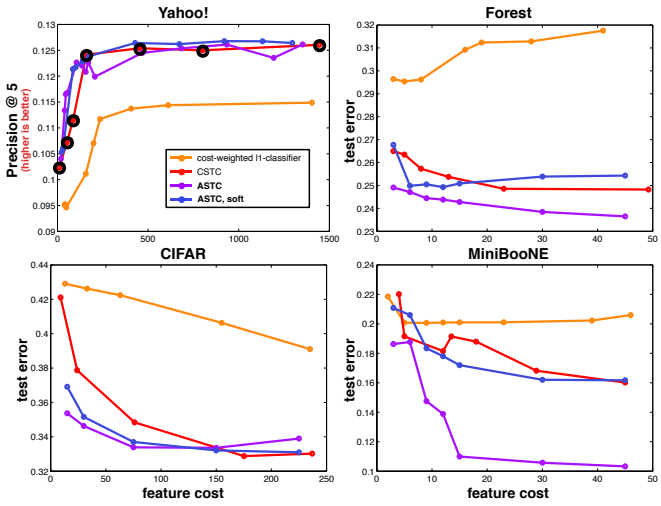


Figure 3: Plot of ASTC, CSTC, and a cost-sensitive baseline on on real-world feature-cost sensitive dataset (Yahoo!) and three non-cost sensitive datasets (Forest, CIFAR, MiniBooNE). ASTC demonstrates roughly the same error/cost trade-off as CSTC, sometime improving upon CSTC. For Yahoo! circles mark the CSTC points that are used for training time comparison, otherwise, all points are compared.

Figure 3 compares the test Precision@5 of CSTC with the greedy algorithm described in Algorithm 1 (*ASTC*). For both algorithms we set a maximum tree depth of 5. We also compare against setting the probabilities $p_i^k$ using the sigmoid function $\sigma(\mathbf{x}^\top \boldsymbol{\beta}^k) = 1/(1+\exp(-\mathbf{x}^\top \boldsymbol{\beta}^k))$ on the node predictions as is done by CSTC (*ASTC, soft*). Specifically, the probability of an input $\mathbf{x}$ traversing from parent node $v^k$ to its upper child $v^u$ is $\sigma(\mathbf{x}^\top \boldsymbol{\beta}^k - \theta^k)$ and to its lower child $v^l$ is $1-\sigma(\mathbf{x}^\top \boldsymbol{\beta}^k - \theta^k)$. Thus, the probability of $\mathbf{x}$ reaching node $v^k$ *from the root* is the product of all such parent-child probabilities from the root to $v^k$. Unlike CSTC, we disregard the effect $\boldsymbol{\beta}^k$ has on descendant node probabilities (see Figure 2). Finally, we also compare against a single cost-weighted $\ell_1$-regularized classifier.

We note that the ASTC methods perform just as good, and sometimes slightly better, than state-of-the-art CSTC. All of the techniques perform better than the single $\ell_1$ classifier, as it must extract features that perform well for all instances. CSTC and ASTC instead may select a small number of expert features to classify small subsets of test inputs.

**Forest, CIFAR, MiniBooNE.** We evaluate ASTC on three very different non-cost sensitive datasets in tree type and image classification (*Forest*, *CIFAR*), as well as particle identification (*MiniBooNE*). As the feature extraction costs are unknown we set the cost of each feature $\alpha$ to $c(\alpha) = 1$. As before, ASTC is able to improve upon CSTC.

**Training time speed-up.** Tables 1 and 2 show the speed-up of our approaches *over* CSTC for various tree budgets.

Table 1: Training speed-up of ASTC over CSTC for different tree budgets on Yahoo! and Forest datasets.

| | YAHOO! | | | | | | | FOREST | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COST BUDGETS | 10 | 52 | 86 | 169 | 468 | 800 | 1495 | 3 | 5 | 8 | 13 | 23 | 50 |
| ASTC | 119X | 52X | 41X | 21X | 15X | 9.2X | 6.6X | 8.4X | 7.0X | 6.3X | 4.9X | 3.1X | 1.4X |
| ASTC, SOFT | 121X | 48X | 46X | 18X | 15X | 8.2X | 6.4X | 8.0X | 6.4X | 5.7X | 4.5X | 2.8X | 1.5X |

Table 2: Training speed-up of ASTC over CSTC for CIFAR and MiniBooNE datasets.

| | CIFAR | | | | | MINIBOONE | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COST BUDGETS | 9 | 24 | 76 | 180 | 239 | 4 | 5 | 12 | 14 | 18 | 33 | 47 |
| ASTC | 5.6X | 2.3X | 0.68X | 0.25X | 0.14X | 7.4X | 7.9X | 5.5X | 5.2X | 4.1X | 3.1X | 2.0X |
| ASTC, SOFT | 5.3X | 2.3X | 0.62X | 0.27X | 0.13X | 7.2X | 6.2X | 5.9X | 4.2X | 4.3X | 2.5X | 1.7X |

For a fair speed comparison, we first learn a CSTC tree for different values of $\lambda$, which controls the allowed feature extraction cost (the timed settings on the Yahoo! dataset are marked with black circles on Figure 3, whereas all points are timed for the other datasets). We then determine the cost of unique features extracted at each node in the learned CSTC tree. We set these unique feature costs as individual node budgets $B^k$ for ASTC methods and greedily learn tree features until reaching the budget for each node. We note that on the real-world feature-cost sensitive dataset Yahoo! the ASTC methods are consistently faster than CSTC. Of the remaining datasets ASTC is faster in all settings except for three parameter settings on CIFAR. One possible explanation for the reduced speed-ups is that the training set of these datasets are much smaller (Forest: $n = 36,603$ $d = 54$; CIFAR: $n = 19,761$ $d = 400$; MiniBooNE: $n = 45,523$ $d = 50$) than Yahoo! ($n = 141,397$ and $d = 519$). Thus, the speed-ups are not as pronounced and the small, higher dimensionality CIFAR dataset trains slightly slower than CSTC.

## Related work

Prior to CSTC (Xu et al. 2014), a natural approach to controlling feature resource cost is to use $\ell_1$-regularization to obtain a sparse set of features (Efron et al. 2004). One downside of these approaches is that certain inputs may only require a small number of cheap features to compute, while other inputs may require a number of expensive features.

This scenario motivated the development of CSTC (Xu et al. 2014). There are a number of models that use similar decision-making schemes to speed-up test-time classification. This includes Dredze, Gevaryahu, and Elias-Bachrach (2007) who build feature-cost sensitive decision trees, Busa-Fekete et al. (2012) who use a Markov decision process to adaptively select features for each instance, Xu et al. (2013b) who build feature-cost sensitive representations, and Wang and Saligrama (2012) who learn subset-specific classifiers.

Feature selection has been tackled by a number of submodular optimization papers (Krause and Cevher 2010; Das and Kempe 2011; Das, Dasgupta, and Kumar 2012; Krause and Guestrin 2012). Surprisingly, until recently, there were relatively few papers addressing resource-efficient learning. Grubb and Bagnell (2012) greedily learn weak learners that are cost-effective using (orthogonal) matching pursuit. Work last year (Zolghadr et al. 2013) considers an online setting in which a learner can purchase features in 'rounds'. Most similar to our work is Golovin and Krause (2011) who learn a policy to adaptively select features to optimize a set function. Differently, their work assumes the set function is fully submodular and every policy action only selects a single element (feature). To our knowledge, this work is the first tree-based model to tackle resource-efficient learning using approximate submodularity.

## Conclusion

We have introduced Approximately Submodular Tree of Classifiers (ASTC), using recent developments in approximate submodular optimization to develop a practical near-optimal greedy method for feature-cost sensitive learning. The resulting optimization yields an efficient update scheme for training ASTC up to 120 times faster than CSTC.

One limitation of this approach is that the approximation guarantee does not hold if features are preprocessed. Specifically, for web-search ranking, it is common to first perform gradient boosting to generate a set of limited-depth decision trees. The predictions of these decision trees can then be used as features, as in non-linear of CSTC (Xu et al. 2013a).

Additionally, a set of features may cost less than the sum of their individual costs. One common example is image descriptors for object detection such as HOG features (Dalal and Triggs 2005). A descriptor can be thought of as a group of features. Once a single feature from the group is selected, the remaining features in the group become 'free', as they were already computed for the descriptor. Extending ASTC to these features would widen the scope of the approach.

Overall, by presenting a simple, near-optimal method for feature-cost sensitive learning we hope to bridge the gap between machine learning models *designed* for real-world industrial settings and those *implemented* in such settings. Without the need for expert tuning and with faster training we believe our approach can be rapidly used in an increasing number of large-scale machine learning applications.

**Acknowledgements.** KQW, MK, ZX, YC, and WC are supported by NIH grant U01 1U01NS073457-01 and NSF grants 1149882, 1137211, CNS-1017701, CCF-1215302, IIS-1343896. QZ is partially supported by Key Technologies Program of China grant 2012BAF01B03 and NSFC grant 61273233. Computations were performed via the Washington University Center for High Performance Computing,

partially through grant NCRR 1S10RR022984-01A1.

## References

Bottou, L.; Peters, J.; Quiñonero-Candela, J.; Charles, D. X.; Chickering, D. M.; Portugaly, E.; Ray, D.; Simard, P.; and Snelson, E. 2013. Counterfactual reasoning and learning systems: the example of computational advertising. *The Journal of Machine Learning Research* 14(1):3207–3260.

Busa-Fekete, R.; Benbouzid, D.; Kégl, B.; et al. 2012. Fast classification using sparse decision dags. In *International Conference on Machine Learning*.

Chen, M.; Weinberger, K. Q.; Chapelle, O.; Kedem, D.; and Xu, Z. 2012. Classifier cascade for minimizing feature evaluation cost. In *International Conference on Artificial Intelligence and Statistics*, 218–226.

Dalal, N., and Triggs, B. 2005. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, volume 1, 886–893.

Das, A., and Kempe, D. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *International Conference on Machine Learning*.

Das, A.; Dasgupta, A.; and Kumar, R. 2012. Selecting diverse features via spectral regularization. In *Advances in Neural Information Processing Systems*, 1592–1600.

Diekhoff, G. 1992. *Statistics for the social and behavioral sciences: univariate, bivariate, multivariate*. Wm. C. Brown Publishers Dubuque, IA.

Dredze, M.; Gevaryahu, R.; and Elias-Bachrach, A. 2007. Learning fast classifiers for image spam. In *Conference on Email and Anti-Spam*.

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

Golovin, D., and Krause, A. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research* 42(1):427–486.

Grubb, A., and Bagnell, J. A. 2012. Speedboost: Anytime prediction with uniform near-optimality. In *International Conference on Artificial Intelligence and Statistics*.

Hastie, T.; Tibshirani, R.; and Friedman, J. 2001. The elements of statistical learning: data mining, inference and prediction. *New York: Springer-Verlag* 1(8):371–406.

Johnson, R. A., and Wichern, D. W. 2002. *Applied multivariate statistical analysis*, volume 5. Prentice hall Upper Saddle River, NJ.

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

Krause, A., and Cevher, V. 2010. Submodular dictionary selection for sparse representation. In *International Conference on Machine Learning*, 567–574.

Krause, A., and Guestrin, C. E. 2012. Near-optimal nonmyopic value of information in graphical models. *arXiv preprint arXiv:1207.1394*.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions i. *Mathematical Programming* 14(1):265–294.

Streeter, M., and Golovin, D. 2007. An online algorithm for maximizing submodular functions. Technical report, DTIC Document.

Wang, J., and Saligrama, V. 2012. Local supervised learning through space partitioning. In *Advances in Neural Information Processing Systems*, 91–99.

Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; and Attenberg, J. 2009. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning*, 1113–1120.

Xu, Z.; Kusner, M. J.; Chen, M.; and Weinberger, K. Q. 2013a. Cost-sensitive tree of classifiers. In *International Conference on Machine Learning*.

Xu, Z.; Kusner, M.; Huang, G.; and Weinberger, K. Q. 2013b. Anytime representation learning. In *International Conference on Machine Learning*.

Xu, Z.; Kusner, M. J.; Weinberger, K. Q.; Chen, M.; and Chapelle, O. 2014. Budgeted learning with trees and cascades. *Journal of Machine Learning Research*.

Zhang, D.; He, J.; Si, L.; and Lawrence, R. 2013. Mileage: Multiple instance learning with global embedding. *International Conference on Machine Learning*.

Zheng, Z.; Zha, H.; Zhang, T.; Chapelle, O.; Chen, K.; and Sun, G. 2008. A general boosting method and its application to learning ranking functions for web search. In *Advances in Neural Information Processing Systems*. 1697–1704.

Zolghadr, N.; Bartók, G.; Greiner, R.; György, A.; and Szepesvári, C. 2013. Online learning with costly features and labels. In *Advances in Neural Information Processing Systems*, 1241–1249.