Training-Time Optimization of a Budgeted Booster

Yi Huang, Brian Powers, Lev Reyzin
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
Chicago, IL 60607
{yhuang,bpower6,lreyzin}@math.uic.edu

Abstract

We consider the problem of feature-efficient prediction – a setting where features have costs and the learner is limited by a budget constraint on the total cost of the features it can examine in test time. We focus on solving this problem with boosting by optimizing the choice of base learners in the training phase and stopping the boosting process when the learner's budget runs out. We experimentally show that our method improves upon the boosting approach `AdaBoostRS` [Reyzin, 2011] and in many cases also outperforms the recent algorithm `SpeedBoost` [Grubb and Bagnell, 2012]. We provide a theoretical justication for our optimization method via the margin bound. We also experimentally show that our method outperforms pruned decision trees, a natural budgeted classifier.

1 Introduction

The problem of budgeted learning centers on questions of resource constraints imposed on a traditional supervised learning algorithm. Here, we focus on the setting where a learner has ample resources during training time but is constrained by resources in predicting on new examples. In particular, we assume that accessing the features of new examples is costly (with each feature having its own access cost), and predictions must be made without running over a given budget. This budget may or may not be known to the learner. Learners that adhere to such budget constraints are sometimes called **feature-efficient**.

A classic motivation for this problem is the medical testing setting where features correspond to the results of tests that are often costly or even dangerous to perform. Diagnoses often need to be made on incomplete information and doctors must order tests thoughtfully in order to stay within whatever budgets the world imposes. In internet-scale applications this problem also comes up. The cost to access certain features of a document or website is often used as a proxy for computing time which is crucially important to minimize. To address this issue, Yahoo! has recently released datasets which include feature costs [Xu *et al.*, 2012].

Here, we focus on boosting methods, in particular `AdaBoost`, to make them feature-efficient predictors. This line of work was started by Reyzin [2011], who introduced the algorithm `AdaBoostRS`, a feature-efficient version of `AdaBoost`. While `AdaBoostRS` provably converges to the behavior of `AdaBoost` as the feature budget is increased, it only considers feature costs and budget at test time. Reyzin left open the problem of whether optimizing during training can improve performance. Here, we answer this question with a resounding yes, giving algorithms that clearly outperform `AdaBoostRS`, especially when costs vary and budget limits are small.

Our approach relies mainly on two observations. The first is that when all features have equal costs, stopping the training of `AdaBoost` early, once the budget runs out, will outperform `AdaBoostRS`. Second, when features have different costs, which is the setting that chiefly concerned Reyzin, one can still run `AdaBoost`but choose weak learners as to better trade-off their cost against contribution to the performance of the ensemble.

2 Past work

Research on this problem goes back at least to Wald [1947], who considered the problem of running a clinical trial sequentially, only testing future patients if the validity of the hypothesis in question is still sufficiently uncertain. This question belongs to the area of sequential analysis [Chernoff, 1972].

Ben-David and Dichterman [1993] examined the theory behind learning using random partial information from examples and discussed conditions for learning in their model. Greiner et al. [2002] also considered the problem of feature-efficient prediction, where a classifier must choose which features to examine before predicting. They showed that a variant of PAC-learnability is still possible even without access to the full feature set.

In related settings, Globerson and Roweis [2006] looked at building robust predictors that are resilient to corrupted or missing features. Cesa-Bianchi et al. [2010] studied how to efficiently learn a linear predictor in the setting where the learner can access only a few features per example. In the multi-class setting, Gao and Koller [2011] used a parameter-tuned value-theoretic computation to create efficient instance-specific decision paths. In a similar vein, Schwing et al. [2011] trained a random forest to adaptively select experts at test-time via a tradeoff parameter. He et al. [2012] trained an MDP for this task, casting it as dynamic feature selection – their model is a variant of ours, except that they attempt to jointly optimize feature costs and errors, whereas our model has a strict bound on the budget. Finally, Xu *et al.* [2012] tackled a related a feature-efficient regression problem by training CART decision trees with feature costs incorporated as part of the impurity function.

In the area of boosting, Pelossof et al. [2010] analyzed how to speed up margin-based learning algorithms by stopping evaluation when the outcome is close to certain. Sun and Zhou [2013] also considered how to order base learner evaluations so as to save prediction time.

However, our main motivation is the work of Reyzin [2011], who tackled the feature-efficient learning problem using ensemble predictors. He showed that sampling from a weights distribution of an ensemble yields a budgeted learner with similar properties to the original ensemble, and he tested this idea experimentally on `AdaBoost`. The goal of this paper is to improve on Reyzin's approach by considering the feature budget during training.

We also compare to the recent work of Grubb and Bagnell [2012], who also focused on this setting. Their algorithm, `SpeedBoost`, works by sequentially choosing weak learners and voting weight α as to greedily optimize the improvement of a loss function (e.g. exponential loss) per unit cost, until the budget runs out.

3 AdaBoost and early stopping

Our goal in this paper is to produce an accurate classifier given a budget B and a set of m training examples, each with n features, and each feature with a cost via cost function $C : [n] \to \mathbb{R}^+$.

Reyzin's `AdaBoostRS` [2011] takes the approach of ignoring feature cost during training and then randomly selecting hypotheses from the ensemble produced by `AdaBoost` until the budget is reached.

Here we look at a different approach – to optimize the cost efficiency of boosting during training, so the ensemble classifier that results is both relatively accurate and affordable.

One straightforward approach is to run `AdaBoost`, paying for the features of the weak learners chosen every round, bookkeeping expenditures and the features used, until we cannot afford to continue. In this case we are simply stopping `AdaBoost` early. We call this algorithm the "basic" `AdaBoostBT` for Budgeted Training. Surprisingly, this albeit simple methodology produces results that are significantly better than `AdaBoostRS` for both features with a uniform cost and features with random cost across a plethora of datasets.

We note that, in `AdaBoost`, since training error is upper bounded by

$$\widehat{\Pr}[H(x) \neq y] \;\leq\; \prod_{t=1}^{T} Z_t \;=\; \prod_{t=1}^{T} \sqrt{1-\gamma_t^2},$$

at each round t of boosting one typically greedily chooses the base learner that minimizes the quantity Z_t, which is equivalent to choosing the base learner that maximizes γ_t. This is done in order to bound the generalization error, which was

Algorithm 1 `AdaBoostBT(S,B,C)`, where: $S \subset X \times \{-1,+1\}$, $B > 0$, $C : [i \ldots n] \to \mathbb{R}^+$

1: given: $(x_1, y_1), ..., (x_m, y_m) \in S$
2: initialize $D_1(i) = \frac{1}{m}$, $B_1 = B$

3: **for** $t = 1, \ldots, T$ **do**
4: train base learner using distribution D_t, get

$$\boxed{h_t \in \mathcal{H} : X \to \{-1,+1\}}$$

5: **if** the total cost of the unpaid features of h_t exceeds B_t **then**
6: set $T = t - 1$ and **end for**
7: **else** set B_{t+1} as B_t minus the total cost of the unpaid features of h_t, marking them as paid
8: set

$$\alpha_t = \frac{1}{2} \ln \frac{1+\gamma_t}{1-\gamma_t},$$

where $\gamma_t = \sum_i D_t(i) y_i h_t(x_i)$
9: update

$$D_{t+1}(i) = D_t(i) \exp(\alpha_t y_i h_t(x_i))/Z_t,$$

where Z_t is the normalization factor
10: **end for**

11: output the final classifier

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

shown by Freund and Schapire [1997] to be bounded by

$$\Pr[H(x) \neq y] \leq \widehat{\Pr}[H(x) \neq y] + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right).$$

In these bounds $\widehat{\Pr}$ refers to the probability with respect to the training sample, and d is the VC-dimension of $\mathcal{H}$.

Hence, one can simply choose h_t in step 4 of `AdaBoostBT` according to this rule, which amounts to stopping `AdaBoost` early if its budget runs out. As we show in Section 6, this already yields an improvement over `AdaBoostRS`. This is unsurprising, especially when the budget or number of allowed rounds is low, as `AdaBoost` aggressively drives down the training error (and therefore generalization error), which `AdaBoostRS` does not do as aggressively. A similar observation will explain why the methods we will introduce presently also outperform `AdaBoostRS`.

However, this approach turns out to be suboptimal when costs are not uniform. Namely, it may sometimes be better to choose a worse-performing hypothesis if its cost is lower. Doing so may hurt the algorithm on that current round, but allow it to afford to boost for longer, more than compensating for the locally suboptimal choice.

4 A better trade-off

Here we focus on the problem of choosing a weak learner when feature costs vary. Clearly, higher values of γ_t are still preferable, but so are lower feature costs. Both contribute to minimizing the quantity $\prod_{t=1}^{T} Z_t$, which upper bounds the training error. High γ_ts make the product small term-wise. Lower costs, on the other hand, allow for more terms in the product.[1] The goal is to strike the proper balance between the two.

One problem is that it is difficult to know exactly how many future rounds of boosting can be afforded under most strategies. If we make the assumption that we expect the costs of base learners selected in future rounds to be similar to the cost c in this round, we could afford B_t/c additional rounds of boosting. Assuming that future rounds will incur the same cost and achieve the same γ_t as in the current round, minimizing the quantity $\prod_{t=1}^{T} Z_t$ is equivalent to minimizing the quantity

$$\left(1-\gamma_t(h)^2\right)^{T/2} = \left(1-\gamma_t(h)^2\right)^{B_t/2c}.$$

Since $B_t/2$ is common to all hypotheses, h_t is chosen using the following rule:

$$h_t = \operatorname{argmin}_{h \in \mathcal{H}} \left((1-\gamma_t(h)^2)^{\frac{1}{c(h)}} \right), \qquad (1)$$

where

$$\gamma_t(h) = \sum_i D_t(i) y_i h(x_i),$$

and $c(h)$ is the cost of the features used by h. This is our first algorithm criterion for modifying base learner selection in step 4 of AdaBoostBT (boxed for emphasis). We call this algorithm AdaBoostBT_Greedy.

There is a potential pitfall with this approach: if we mark every used feature down to cost 0 (since we don't re-pay for features), then the optimization will collapse since every base learner with cost 0 will be favored over all other base learners, no matter how uninformative it is. We can obviate this problem by considering the original cost during the selection, but not paying for used features again while updating B_t, as is done in our Algorithm.

As optimizing according to Equation 1 makes a very aggressive assumption of future costs, we consider a smoother optimization for our second approach. If in round t, we were to select h_t with cost c, the average cost per round thus far is then

$$\frac{(B-B_t)+c}{t}.$$

If we expect future rounds to have this average cost, we get a different estimate of the number of additional rounds we are able to afford. Specifically, in step 4 of AdaBoostBT, we select a base learner according to

$$h_t = \operatorname{argmin}_{h \in \mathcal{H}} \left(\left(1-\gamma_t(h)^2\right)^{\frac{1}{(B-B_t)+c(h)}} \right). \qquad (2)$$

[1] This also creates a slightly more complex classifier, which factors into the generalization error bound, but this effect has been observed to not be very significant in practice [Reyzin and Schapire, 2006; Schapire *et al.*, 1998].

Selecting according to Equation 2 is less aggressive, because as more of the budget is used, current costs matter less and less. Hence, we call this second algorithm AdaBoostBT_Smoothed. While some other feature-efficient algorithms require a non-heuristic tuning parameter to control trade-off, this equation continually revises the trade-off as the budget is spent. Our experimental results show that it pays to be less greedy for larger budgets. To further customize this trade-off, however, a parameter $\tau \in (0,1)$ may be used to scale $(B - B_t)$ allowing greed to drive the decision for more rounds of boosting.

One could even implement a hybrid approach, in which the algorithm begins by using AdaBoostBT_Greedy to select weak learners and switches to AdaBoostBT_Smoothed when the quality (a function of γ and c) of the unused features drops below a certain threshold. Exploring this hybrid idea, however, is outside the scope of this paper.

5 Additional theoretical justification

While the theory behind optimizing the bound of $\prod_{t=1}^{T} Z_t$ on the training error is clear, we can borrow from the theory of margin bounds [Schapire *et al.*, 1998] to understand why this optimization yields improved results for generalization error.

One might be concerned that in finding low cost hypotheses, we will be building too complex a classifier, which will not generalize well. In particular, this is the behavior that the Freund and Schapire [1997] generalization bound would predict. Fortunately, the margin bound theory can be used to alleviate these concerns.

The following bound, known as the margin bound, bounds the probability of error as a sum of two terms.

$$\Pr[yf(x) \le 0] \quad \le \quad \widehat{\Pr}[yf(x) \le \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right),$$

where $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$, as in Algorithm 1. The first term is the fraction of training examples whose margin is below a given value and the second term is independent of the number of weak learners.

It can be shown [Schapire and Freund, 2012] that the first term can be bounded as follows

$$\widehat{\Pr}[yf(x) \le \theta] \le e^{\theta \sum \alpha_i} \prod_{t=1}^{T} Z_t,$$

where α_i is defined in Algorithm 1. For small θ this tends to

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1-\gamma_t^2}.$$

This well-known viewpoint provides additional justification for optimizing $\prod_{t=1}^{T} Z_t$, as is done in the preceding section.

6 Experimental results

Although there are a number of feature-efficient classification methods [Gao and Koller, 2011; Schwing *et al.*, 2011; Xu *et al.*, 2012], we directly compare the performance of AdaBoostBT, AdaBoostBT_Greedy

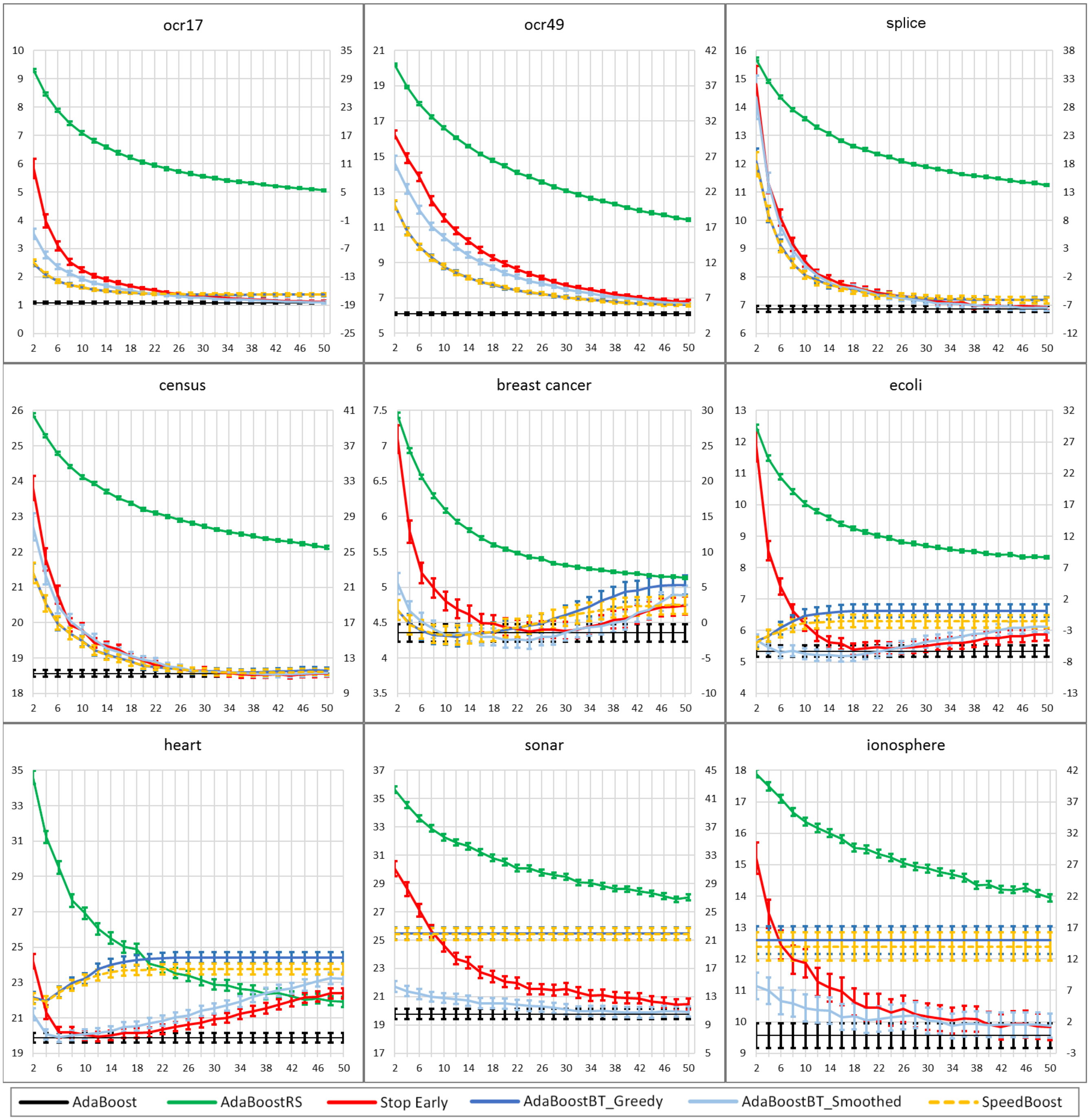

Figure 1: Experimental results comparing our approaches to `AdaBoostRS` [Reyzin, 2011] and `SpeedBoost` [Grubb and Bagnell, 2012]. Test error is calculated at budget increments of 2. The feature costs are uniformly distributed in the interval [0,2]. The horizontal axis has the budget, and the vertical axis has the test error rate. `AdaBoostRS` test error rate uses the secondary vertical axis (on the right hand side) for all data sets except for heart. Error bars represent a 95% confidence interval.

and `AdaBoostBT_Smoothed` to `AdaBoostRS` and `SpeedBoost` as both are feature-efficient boosting methods which allow for any class of weak learners.

For our experiments we first used datasets from the UCI repository, as shown in Table 1. The features and labels were collapsed into binary categories, and decision stumps were used for the hypothesis space.

Experimental results, given in Figure 1, compare average generalization error rates over multiple trials, each with a random selection of training examples. Features are given costs

	num features	training size	test size	AdaBoost rounds (optimized)	trials
ocr17	403	1000	5000	400	100
ocr49	403	1000	5000	200	100
splice	240	1000	2175	75	100
census	131	1000	5000	880	100
breast cancer	82	500	199	500	400
ecoli	356	200	136	50	400
sonar	11196	100	108	99	400
heart	371	100	170	15	400
ionosphere	8114	300	51	400	400
webscope set2	519	7000	15439	500	20

Table 1: Dataset sizes, numbers of features for training and test, and number of rounds when running the `AdaBoost` predictor.

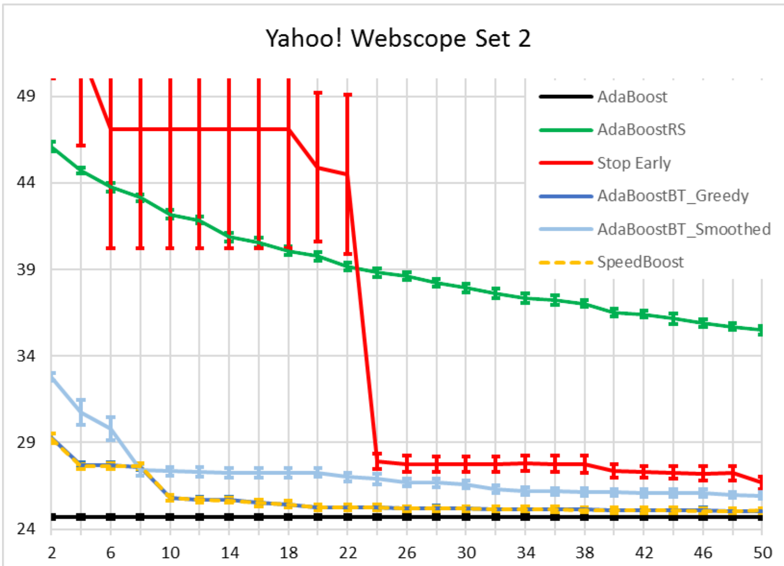

Figure 2: Experimental results comparing our approaches to `AdaBoostRS` and `SpeedBoost` on the Yahoo! Webscope data set 2. Test error is calculated at budget increments of 2. Error bars represent a 95% confidence interval.

uniformly at random on the interval $[0, 2]$. For comparison, `AdaBoost` was run for a number of rounds that gave lowest test error, irrespective of budget. This setup was chosen to compare directly against the results of Reyzin [2011] who also used random costs. We test on all the datasets Reyzin used, plus others.

Then, to study our algorithms on real-world data, we used the Yahoo! Webscope dataset 2, which includes feature costs [Xu *et al.*, 2012]. The data set contains 519 features, whose costs we rescaled to costs to the set $\{.1, .5, 1, 2, 5, 10, 15, 20\}$. Examples are query results labeled 0 (irrelevant) to 5 (perfectly relevant). We chose to collapse labels 1-5 to be binary label 1 (relevant) to test our algorithms. Results are given in Figure 2.

The most apparent conclusion from our experiments is that it is not only possible to improve upon `AdaBoostRS` by optimizing base learner selection during training, but that the improvement is dramatic. Further modifications of the basic `AdaBoostBT` tend to yield additional improvements.

`AdaBoostBT_Greedy` often performs better than the basic `AdaBoostBT` for small budgets, but it chooses base learners quite aggressively – a low cost base learner is extremely attractive at all rounds of boosting. This makes it possible that the algorithm falls into a trap, as we see in the sonar and ionosphere datasets where a huge number of features (11,196 and 8,114 respectively) lead to many features with costs close to zero (due to the cost distribution). Even after 500 rounds of boosting on the sonar dataset, this approach still does not spend the budget of 2 because the same set of cheap features are re-used round after round leading to a deficient classifier. Similar behavior is seen for the ecoli (356) and heart (371) datasets which also have relatively small training sizes, leading to over-fitting on small budgets.

`AdaBoostBT_Smoothed` avoids this trap by considering the average cost instead. The appeal of cheap base learners is dampened as the boosting round increases, with its limiting behavior to choose weak learners that maximize γ. Thus, we can see that using `AdaBoostBT_Smoothed`, while tending to perform worse than `AdaBoostBT_Greedy` for low budgets, tends to exceed its accuracy for larger budgets.

In cases when `AdaBoost` will noticeably over-fit with larger budgets (breast cancer, ecoli, heart - note the behavior of Stopping Early) we note that `AdaBoostBT_Smoothed` achieves the same (or better) error rate as `AdaBoost` at much lower budgets. For example, on the ecoli data set, `AdaBoost` needs a budget of 18 to achieve what `AdaBoostBT_Smoothed` does with a budget of 6.

On the Yahoo! Webscope data set, we see a dramatic difference between `AdaBoostBT` and our other optimizations. However, this is understandable because `AdaBoost` typically includes the expensive feature (cost of 20) in early rounds of boosting thus failing to produce a feature-efficient classifier for small budgets. Both the `Greedy` and `Smoothed` optimizations, however, effectively select from the less expensive features to create powerful low-budget classifiers.

Comparing to `Speedboost`

We also compare to the classification algorithm `SpeedBoost` [Grubb and Bagnell, 2012] on all data sets, which was recently designed for tackling the same issue

`SpeedBoost` works by choosing weak learners (together with a voting weight α) so as to greedily optimize the improvement of a loss function per unit cost until the budget runs out. To mirror Grubb and Bagnell [2012], as well as `AdaBoost`, we use exponential loss.

In our experiments, `SpeedBoost` performs almost identically to `AdaBoostBT_Greedy`. This phenomenon can be explained as follows. In `AdaBoostBT_Greedy` we find $\mathrm{argmin}_{h\in\mathcal{H}}\left(1-\gamma(h)^2\right)^{\frac{1}{c(h)}}$, while in `SpeedBoost`, implicitly we find $\mathrm{argmax}_{h\in\mathcal{H}}\frac{1-\sqrt{1-\gamma(h)^2}}{c(h)}$ (See Appendix A). Since

$$\min_{h\in\mathcal{H}}\left(1-\gamma(h)^2\right)^{\frac{1}{c(h)}} = \max_{h\in\mathcal{H}}\frac{-\ln\sqrt{1-\gamma(h)^2}}{c(h)},$$

and the Taylor series of $-\ln(x)$ is

$$(1-x)+\frac{1}{2}(1-x)^2 - o((1-x)^2),$$

we have when $\gamma(h)$ is close to 0 (the value toward which boosting drives edges by making hard distributions), the two functions `SpeedBoost` and `AdaBoostBT_Greedy` seek to optimize are very similar.

Moreover, both algorithms greedily optimize an objective function without considering the impact on future rounds. Hence, `SpeedBoost` falls into the same trap of copious cheap hypotheses as `AdaBoostBT_Greedy`. Note: the lines for `SpeedBoost` are dashed because they overlap with `AdaBoostBT_Greedy`.

Yet, our approach offers a number of benefits over `SpeedBoost`. First, we have flexibility of explicitly considering future rounds, as `AdaBoostBT_Smoothed` does, in many cases outperforming `SpeedBoost`– e.g. on the ecoli, heart, sonar, and ionosphere data. Second, computational issues (for a discussion, see [Grubb and Bagnell, 2012]) surrounding choosing the best α is completely avoided in our weak learner selection – in fact, we show in Appendix A that the double optimization of `SpeedBoost` can also be avoided. Hence, our approach is simple, yet powerful at the same time.

7 A note on decision trees

One straightforward method for making a budgeted predictor is to use decision trees, and we consider this approach in this section. Decision trees are a natural choice for budgeted predictor since after a tree is constructed, each test example only needs to go through one path of the tree in order the receive a label. Hence, it incurs a cost using features only on that particular path. One may even be tempted to think that simply using decision trees would be optimal for this problem. We experimentally show that this is not the case, especially for larger budgets.

The problem with decision trees is that when one has more budget at hand and is able to grow larger trees, the trees begin to overfit, and this occured on all datasets (Figure 3). In contrast, `AdaBoostBT` performs better with more budget on most datasets. Cost optimization methods, both `Greedy` and `Smoothed`, tend to exploit large budgets to an even greater extent. Budgeted boosting algorithms continue to drive error rates down with higher budgets; decision trees do not.

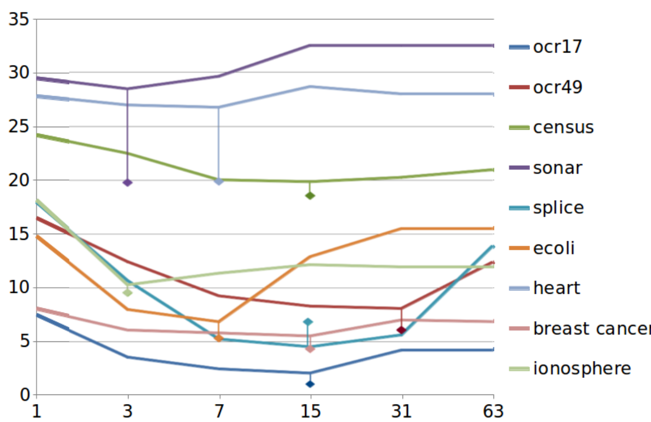

Figure 3: Error Rates of decision trees. The horizontal axis is these number of nodes (log scale in number of nodes, linear in expected tree depth). The vertical axis is percent error. Diamonds show the `AdaBoost` error rate for easy comparison.

8 Future work

One direction for future work is to improve optimization for cost distributions with few cheap features. In addition, one could consider an adversarial cost model where cost and feature performance are strongly positively correlated, and analyze the extent to which optimization could help.

Other potentially promising approaches within our framework would be to boost weak learners other than decision stumps - for instance, boosting decision trees with the optimizations we suggest in Section 3 would likely outperform stumps, especially because decision trees show highest gains at small budgets (see Figure 3). There remains the open question of how to efficiently optimize Equation 1 or 2 for certain exponentially large or infinite classes of weak learners.

Finally, it would be worthwhile to study making other machine learning algorithms feature-efficient by incorporating budgets in their training.

References

Shai Ben-David and Eli Dichterman. Learning with restricted focus of attention. In *COLT*, pages 287–296, New York, NY, USA, 1993. ACM.

Nicolò Cesa-Bianchi, Shai Shalev-Shwartz, and Ohad Shamir. Efficient learning with partially observed attributes. *CoRR*, abs/1004.4421, 2010.

Herman Chernoff. *Sequential Analysis and Optimal Design*. SIAM, 1972.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

Tianshi Gao and Daphne Koller. Active classification based on value of classifier. In *Advances in Neural Information Processing Systems*, pages 1062–1070, 2011.

Amir Globerson and Sam T. Roweis. Nightmare at test time: robust learning by feature deletion. In *ICML*, pages 353–360, 2006.

Russell Greiner, Adam J. Grove, and Dan Roth. Learning cost-sensitive active classifiers. *Artif. Intell.*, 139(2):137–174, 2002.

Alexander Grubb and Drew Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, pages 458–466, 2012.

He He, Hal Daumé III, and Jason Eisner. Imitation learning by coaching. In *NIPS*, pages 3158–3166, 2012.

Raphael Pelossof, Michael Jones, and Zhiliyang Ying. Speeding-up margin based learning via stochastic curtailment. In *ICML/COLT Budgeted Learning Workshop*, Haifa, Israel, June 25 2010.

Lev Reyzin and Robert E. Schapire. How boosting the margin can also boost classifier complexity. In *ICML*, pages 753–760, 2006.

Lev Reyzin. Boosting on a budget: Sampling for feature-efficient prediction. In *ICML*, pages 529–536, 2011.

R.E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. Adaptive computation and machine learning. MIT Press, 2012.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *the Annals of Statistics*, 26(5):1651–1686, 1998.

Alexander G Schwing, Christopher Zach, Yefeng Zheng, and Marc Pollefeys. Adaptive random forest–how many "experts" to ask before making a decision? In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1377–1384. IEEE, 2011.

Peng Sun and Jie Zhou. Saving evaluation time for the decision function in boosting: Representation and reordering base learner. In *ICML*, 2013.

Abraham Wald. *Sequential Analysis*. Wiley, 1947.

Zhixiang Xu, Kilian Weinberger, and Olivier Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, pages 1175–1182. ACM, July 2012.

APPENDIX

A Optimizing α, h in `SpeedBoost`

Here we show that the double optimization of h and α in `SpeedBoost` is unnecessary computationally. As we show presently, the optimization problem can be written solely as a function of h and α can then be computed directly. Let $H_t(x_i) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)$, `SpeedBoost` tries to find

$$\operatorname*{argmax}_{\alpha\in\mathbb{R}^+, h\in\mathcal{H}} \frac{\sum_{i=1}^m e^{-y_i H_t(x_i)} - \sum_{i=1}^m e^{-y_i(H_t(x_i)+\alpha h(x_i))}}{c(h)}$$

For a fixed $h \in \mathcal{H}$, let

$$I(\alpha) = \sum_{i=1}^{m} e^{-y_i H_t(x_i)} - \sum_{i=1}^{m} e^{-y_i(H_t(x_i)+\alpha h(x_i))},$$

we have

$$\begin{aligned} I'(\alpha) &= -\sum_{i=1}^{m} e^{-y_i H_t(x_i)}(-y_i h(x_i)) e^{-\alpha y_i h(x_i)} \\ &= \sum_{\{i:y_i=h(x_i)\}} e^{-y_i H_t(x_i)} e^{-\alpha} - \sum_{\{i:y_i\neq h(x_i)\}} e^{-y_i H_t(x_i)} e^{\alpha} \end{aligned}$$

which is equal to zero if and only if

$$\alpha = \frac{1}{2}\ln \frac{\sum_{\{i:y_i=h(x_i)\}} e^{-y_i H_t(x_i)}}{\sum_{\{i:y_i\neq h(x_i)\}} e^{-y_i H_t(x_i)}}. \tag{3}$$

Plugging in the α above, we get

$$\max_{\alpha\in\mathbb{R}^+, h\in\mathcal{H}} \frac{\sum_{i=1}^m e^{-y_i H_t(x_i)} - \sum_{i=1}^m e^{-y_i(H_t(x_i)+\alpha h(x_i))}}{c(h)}$$

$$=$$

$$\max_{h\in\mathcal{H}} \frac{1}{c(h)} \left(\sum_{i=1}^{m} \ell_t(i) - 2\left(\sum_{i:y_i=h(x_i)} \ell_t(i) \sum_{i:y_i\neq h(x_i)} \ell_t(i) \right)^{\frac{1}{2}} \right).$$

Where $\ell_t(i) = e^{-y_i H_t(x_i)}$.

We have the optimization problem above is equal to

$$\operatorname*{argmax}_{h\in\mathcal{H}} \frac{1-\sqrt{1-\gamma(h)^2}}{c(h)}. \tag{4}$$

Note that Equation 4 does not depend on α, which, after choosing h, can be set according to Equation 3.