On the Boosting Pruning problem

Christino Tamon[1] and Jie Xiang[2]

[1] Clarkson University, Potsdam, NY 13699-5815, U.S.A. email: tino@clarkson.edu
[2] BCL Computers, Inc., U.S.A. email: jiex@bcl-computers.com

Abstract. Boosting is a powerful method for improving the predictive accuracy of classifiers. The AdaBoost algorithm of Freund and Schapire has been successfully applied to many domains [2, 10, 12] and the combination of AdaBoost with the C4.5 decision tree algorithm has been called the *best off-the-shelf* learning algorithm in practice. Unfortunately, in some applications, the number of decision trees required by AdaBoost to achieve a reasonable accuracy is enormously large and hence is very space consuming. This problem was first studied by Margineantu and Dietterich [7], where they proposed an empirical method called Kappa pruning to *prune* the boosting ensemble of decision trees. The Kappa method did this without sacrificing too much accuracy. In this work-in-progress we propose a potential improvement to the Kappa pruning method and also study the *boosting pruning* problem from a theoretical perspective. We point out that the boosting pruning problem is intractable even to approximate. Finally, we suggest a *margin*-based theoretical heuristic for this problem.

1 Introduction

Boosting is a method for combining classifiers to improve prediction accuracy. The idea of boosting is to alter repeatedly the distribution on the training data so that the learning algorithm is forced to focus on harder examples. A boosting algorithm called AdaBoost (Freund and Schapire [1]) has been extensively studied both theoretically and empirically. The algorithm is proven to be theoretically sound and shown to be empirically appealing because of its simplicity and superior performance in many domains.

Many research have focused on boosting decision trees, notably using Quinlan's C4.5 [9] as the tree induction algorithm. The AdaBoost-C4.5 combination has been called the best off-the-shelf learning algorithm in practice because of its superior performance on many benchmark datasets [10, 2]. Despite its good performance, Margineantu and Dietterich [7] observed that, in some domains, boosting needs to combine a large number of trees to lower the prediction error. More specifically, they observed that in the `letter` dataset, AdaBoost requires about 200 iterations of C4.5 to achieve a reasonable accuracy. So the final classifier is a weighted ensemble of about 200 decision trees (each being a nontrivially large tree). They asked if all 200 decision trees are necessary: is there a way of *pruning* some of these trees from the final ensemble without deteriorating the performance.

Margineantu and Dietterich then proposed an interesting method of pruning the boosting ensemble using a statistic called the Kappa measure (see [7] and the references therein). Their heuristic idea is based on the assumption that boosting works by building *diversity* in its ensemble. The Kappa statistic is a measure of agreement between two classifiers. They create their pruned ensemble by greedily selecting pairs of decision trees with very diverse behavior until they reached the required pruning rate. Up to certain rates of pruning, the performance of the pruned ensemble is quite close to the original ensemble. In this paper we propose a slight modification to the Kappa method called *weight shifting*. Viewing the pruning process as a *clustering*-like process, we shifted the voting weights of pruned trees onto its unpruned neighbors. We conducted some preliminary experiments and observed some encouraging although mixed results.

Next we study some theoretical aspects of the boosting pruning problem. We show that the boosting pruning problem is NP-complete and is even hard to approximate. Then we propose a pruning scheme that is *margin*-based. Recent work by Schapire et al. [11] has shown that boosting achieves good generalization error by maximizing the minimum margin on the training sample. We suggest a theoretical heuristic derived using tools from the area of approximation algorithms, where a trade-off between the margin and the size of the pruned boosting ensemble is made explicit.

2 Boosting Decision Trees

Quinlan's C4.5 algorithm is a well-studied method for inducing decision trees from data (see [9]). It is a top-down method that continually splits the training data using the best attribute under an entropic measure. Several works have studied boosting decision trees by combining AdaBoost with C4.5 (including [10, 2]). We follow Quinlan's boosting experiments [10] by making use of C4.5's ability to assign fractional weights to data items. This will be important in how we do boosting.

The AdaBoost algorithm (Freund and Schapire [1]) works by repeatedly calling the weak learning algorithm (in this case C4.5) on a newly reweighted training data. The reweightings are done so as to focus the weak learner's attention to examples where mistakes are still being made. This cycle repeats until all training data are correctly classified.

We introduce some notation before we describe the AdaBoost algorithm formally. Let X be the example domain and let Y be the label domain. A *labeled sample* S is a sequence of pairs $(x, y) \in X \times Y$. We assume that S is drawn according to some fixed but unknown distribution D over X and that the labels satisfy $y = f(x)$, for some unknown target function f. The *training error* of a function h with respect to sample S is defined as $\epsilon_S(h) = \frac{1}{|S|}\sum_{(x,y)\in S}[\![h(x) \neq y]\!]$, where $[\![\pi]\!]$ is 1 if the statement π is true and 0 otherwise. The *generalization error* of a function h is defined as $\epsilon_D(h) = \Pr_{(x,y)\sim D}[h(x) \neq y]$. The AdaBoost algorithm is shown in Figure 1. In this paper we adopt Quinlan's strategy of boosting by *reweighting* [10] (instead of *resampling* [2]).

Input: A training sample $S = \{(x_i, y_i) \mid 1 \leq i \leq m\}$, where $x_i \in X$ and $y_i \in Y$.
Output: A classifier $H : X \to Y$ with small training error on S.

```
1.  $D_1(x_i) = 1/m$, for all $1 \leq i \leq m$.
2.  for  $t = 1, 2, \ldots, T$ do
3.    call C4.5 on input $S$ and $D_t$
4.    get weak hypothesis $h_t : X \to Y$
5.    $\epsilon_t = \sum_{i=1}^{m} D_t(i)[\![h_t(x_i) \neq y_i]\!]$
6.    if $\epsilon_t \geq 0.5$ then set $T = t - 1$ and abort loop.
7.    $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
8.    reweight distribution: $D_{t+1}(x_i) = D_t(x_i)\beta_t^{[\![h_t(x_i)=y_i]\!]}/Z_t$, where $Z_t$ is a normal-
    ization constant.
9.  end for
10. output $H(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \ln(1/\beta_t)$.
```

Fig. 1. The AdaBoost.C4.5 algorithm

3 Kappa Pruning

The boosting pruning heuristic of Margineantu and Dietterich [7] proceeds as follows. First we define the Kappa measure between two classifiers h_i and h_j, where $h_i, h_j : X \to Y$. Consider the following $|Y| \times |Y|$ contingency table or matrix M: for $a, b \in Y$, define $M_{a,b}$ to be the fraction of examples $x \in S$ where $h_i(x) = a$ and $h_j(x) = b$. Let $\Theta_1 = \sum_{a \in Y} M_{a,a}$ and $\Theta_2 = \sum_{a \in Y} M_{a,*}M_{*,a}$, where $M_{a,*} = \sum_{b \in Y} M_{a,b}$ and $M_{*,a} = \sum_{b \in Y} M_{b,a}$. The parameter Θ_1 is a measure of $\Pr_S[h_i = h_j]$ and Θ_2 is a measure of $\sum_{a \in Y} \Pr_S[h_i = a] \Pr_S[h_j = a]$. Then the Kappa measure of agreement between h_i and h_j is defined as $\kappa(h_i, h_j) = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}$. A value of $\kappa = 0$ implies that $\Theta_1 = \Theta_2$ and the two classifiers are considered to be different (or independent). A value of $\kappa = 1$ implies that $\Theta_1 = 1$ which means total agreement between the two classifiers. It is possible for κ to be negative although it was noted that this rarely occurs [7].

Using this distance measure, the Kappa pruning method [7] proceeds as follows. It computes all pairwise Kappa distances between the decision trees in the boosting ensemble. After sorting these distance values, the algorithm greedily includes the pairs of hypotheses that correspond to small Kappa distances. This continues until a certain pruning rate is achieved. The resulting boosting ensemble consists of all decision trees included from the greedy selection stage. In effect, the Kappa pruning algorithm sets to zero all the voting weight of the pruned decision trees (the α's in the final hypothesis of AdaBoost).

3.1 Weight Shifting

Here we propose an alternative heuristic for performing Kappa pruning based on a *weight shifting* strategy. While Kappa pruning sets to zero the weights of all pruned decision trees in the boosting ensemble, we propose the following variant: transfer the voting weight of a pruned decision tree to the unpruned ones. This strategy views the pruning process as a clustering process whereby a collection of diverse classifiers are selected to represent the original ensemble. We adopt the following *soft assignment* method of shifting the weight of a pruned hypothesis onto the collection of unpruned ones: each unpruned hypothesis receives a fraction of weight proportional to its *similarity* to the pruned hypothesis. So, in the soft assignment, each pruned classifier computes the set of distances from itself to the collection of unpruned classifiers. The pruned classifier then distributes its voting weight using the distribution of distances (after normalization). More weight is given to classifiers that are *closer* (similar or $\kappa \sim 1$) to the pruned classifier.

We conjecture that the weight shifting process helps produce a more faithful final ensemble, especially when the pruning rate is high. We conducted some preliminary experiments on the effectiveness of Kappa pruning with weight shifting using soft assignment. We report our findings in the next section.

3.2 Experiments

The real-world datasets that we used in our experiments were obtained from the University of California at Irvine (UCI) Machine Learning Repository [8]. Some information about the datasets are given in Table 1.

Table 1. UCI datasets.

	examples			attributes		
name	train	test	classes	disc	cont	missing
auto	205	0	7	11	15	yes
crx	490	200	2	9	6	yes
letter	20000	0	26	0	16	none
monk1	124	432	2	6	0	none
monk2	169	432	2	6	0	none
promoter	106	0	2	57	0	none
soybean	316	0	19	35	0	yes
waveform	5000	0	3	0	21	no

In Table 2 we report a 10-fold cross validation estimate of the generalization error for plain C4.5, AdaBoost and C4.5 with no pruning, and AdaBoost and C4.5 with the two pruning options. We have used the conservative choice of using 30 boosting iterations[1]. Plots of these comparisons are omitted from this abstract due to lack of space.

The basic Kappa pruning algorithm is denoted `kp` and the weight-shifted version is denoted `ws`. The pruning rates that we used are $0.9, 0.8, 0.7, 0.6, 0.5$.

[1] We plan to run further experiments using higher number of boosting iterations (e.g., Margineantu and Dietterich [7] used 50 iterations in their experiments).

Here a pruning rate of α means that we eliminate at least $1 - \alpha$ fraction of the ensemble. So a pruning rate of 0.9 eliminates 10% of the ensemble.

We focused on some UCI datasets where boosting (with 30 rounds) showed a definite improvement upon C4.5 alone. The datasets we used are `auto`, `crx`, `letter`, `monk1`, `monk2`, `promoter`, `soybean`, and `waveform`. We will seek those pruning rates where error rates are still lower than the case without pruning. Our future plans include making comparisons between ensembles of the same size (obtained with and without pruning).

Table 2. 10x-val comparison of C4.5, AdaBoost, Kappa, and weight shifting.

	C4.5	AdaBoost	.9		.8		.7		.6		.5	
name	pruned	T=30	kp	ws	kp	ws	kp	ws	kp	ws	kp	ws
auto	22.4	17.4	18.4	18.4	19.4	19.4	19.4	18.9	21.9	20.9	22.4	22.4
crx	16.5	13.5	13.0	13.2	13.3	13.6	13.2	13.0	12.9	12.3	13.6	13.6
letter	12.22	4.43	4.5	4.51	4.69	4.66	5.0	4.96	5.45	5.47	5.91	5.86
monk1	3.8	0.0	25.5	24.9	25.5	25.5	25.5	25.5	25.5	25.5	26.0	25.5
monk2	33.6	32.1	31.6	31.6	32.6	32.6	32.8	32.4	31.9	32.3	32.4	31.4
promoter	25.0	21.0	21.0	21.0	22.0	22.0	23.0	24.0	28.0	27.0	31.0	27.0
soybean	7.2	5.46	5.7	5.9	5.7	5.7	5.6	5.6	5.9	5.9	6.5	6.6
waveform	25.28	19.50	19.50	19.50	19.64	19.62	20.3	20.26	20.54	20.42	21.82	21.74

The comparison on the datasets `auto`, `crx`, `letter`, and `waveform` showed that weight shifting could help improve the Kappa method in certain pruning rates (mainly for aggressive rates). However, the performance of both methods on `letter` is too similar and hence the improvement is perhaps too negligible. We would like to see if an increased number of boosting iterations might improve this situation.

Furthermore, pruning seemed to cause erratic behavior in the `monk` datasets. We are not sure if this is caused by the special form of the `monk` datasets or a subtle error in our experiment. In `monk1`, pruning caused a marked increase in the error rate. In `monk2`, the improvement of weight shifting is a bit erratic after pruning showed an encouraging promise at low pruning rates. Both methods of pruning also do not seem to work well on `promoter` and `soybean` (although in the former case, weight shifting was better than Kappa on high pruning rates).

4 The Abstract Boosting Pruning Problem

In this section we turn to theoretical considerations of the boosting pruning problem. A *boosting ensemble* H is a collection of hypotheses $h : X \to \{-1, +1\}$ from a known class C of classifiers (for instance, decision trees) where each h has an associated weight $\alpha \in \mathbb{R}$. So let $H = \{\langle \alpha_i, h_i \rangle \mid 1 \leq i \leq T\}$ be a boosting ensemble of size T. We identify the ensemble H with the function $H(x) =$

$sgn\left(\sum_{i=1}^{m} \alpha_i h_i(x)\right)$, where $sgn(x) = +1$ if $x \geq 0$ and $sgn(x) = -1$ otherwise. We also identify any subset A of H with the function $H_A(x) = sgn\left(\sum_{i \in A} \alpha_i h_i(x)\right)$.

We will first make the assumption that minimizing training error leads to the minimization of generalization error (or true error). Under this assumption, we formalize the boosting pruning problem as follows. Assume that the example domain X and the label domain Y are fixed.

ENSEMBLE PRUNING
`input:` A boosting ensemble $H = \{\langle \alpha_i, h_i \rangle \mid 1 \leq i \leq T\}$, where, for each $i = 1, 2, \ldots, T$, $\alpha_i \in \mathbb{R}$ and $h_i : X \to \{-1, +1\}$, and a sample set $S = \{\langle x_i, y_i \rangle \in X \times Y \mid 1 \leq i \leq m\}$.
`output:` A subset A of H minimizing the training error of $H_A(x)$ on S.

For simplicity, we consider an associated problem called MATRIX COVER. Associate with each boosting set of T hypotheses and each sample set of m points, a matrix M of size $T \times m$ where $M_{i,j} = -1$ if $h_i(x_j) = y_j$, and $M_{i,j} = -1$ if $h_i(x_j) \neq y_j$. Assume that M satisfies the *positive column-sum* property, i.e., for all $j \in [m]$, $\sum_{i=1}^{T} M_{i,j} > 0$. This last property means that the boosting ensemble associated with the T rows of M is perfect on the m training points. The question now is to find the smallest subset of the rows of M so that the positive column-sum property is maintained.

MATRIX COVER
`input:` An integral matrix M of size $T \times m$ such that, for all $j \in [m]$, $\sum_{i=1}^{T} M_{i,j} > 0$.
`output:` A minimal subset A of the rows of M such that, i.e., for all $j \in [m]$, $\sum_{i \in A} M_{i,j} > 0$.

Claim. MATRIX COVER is NP-complete.

Proof. Reduction from SET COVER (see [3]). □

Given the NP-completeness of MATRIX COVER, it is natural to ask for the next best solution: an approximation algorithm. For $\alpha > 0$, we say that an algorithm is an approximation algorithm for MATRIX COVER if for any input M to MATRIX COVER it outputs a subset B so that $|B| \leq \alpha OPT(M)$, where $OPT(M)$ is the value of the optimal solution. A very strong hardness result can be proven about approximating MATRIX COVER.

Claim. MATRIX COVER is unapproximable to within n^{ϵ}, $\epsilon > 0$, unless $P = NP$.

Proof. Reduction using the MINIMUM PB 0-1 PROGRAMMING (see [6]). □

4.1 A Margin-Based Heuristic

Although MATRIX COVER is highly intractable to approximate, we suggest in this section a theoretical heuristic for the boosting pruning problem. Note that MATRIX COVER imposes the condition that the resulting final hypothesis must have zero error on the training data. Implicitly, the performance of the boosting hypothesis is measured in terms of the number of mistakes. A recent work by Schapire et al. [11] has shown that an alternative measure called *margin* is a better indicator of the generalization error (or true error) of the boosting hypothesis.

Let us assume now that we have a binary prediction problem, where $Y = \{-1, +1\}$, but that each weak hypothesis can use confidence-rated predictions (as in Schapire and Singer's work [12]), i.e., $h : X \to \mathbb{R}$. Here the sign of h reflects its prediction while its magnitude reflects its confidence in that prediction. Note that the final boosting hypothesis (before thresholding) is $H(x) = \sum_i \alpha_i h_i(x)$. The margin of H on the example $(x, y) \in X \times \{-1, +1\}$ is defined as $m(x) = yH(x)$. A positive margin on an example means that H predicts correctly on that example and the magnitude of the margin reflects the magnitude of its correctness. Schapire et al [11] proved that a a hypothesis with large positive margin on all training examples is a hypothesis with low generalization error.

Using margin theory, we suggest a different heuristic to ENSEMBLE PRUNING. In defining the matrix in our MATRIX COVER instance, let $M_{i,j} = y_j h_i(x_j)$ be the margin of the i-th hypothesis h_i on the j-th example (x_j, y_j). Now the j-th column-sum of M is the margin of H on the j-th example (x_j, y_j).

MATRIX COVER
`input:` A positive constant $\theta > 0$ and a real-valued matrix M of size $T \times m$ such that, for all $j \in [m]$, $\sum_{i=1}^{T} M_{i,j} > \theta$.
`output:` A minimal subset A of the rows of M such that, for all $j \in [m]$, $\sum_{i \in A} M_{i,j} > \theta$.

We now attempt to design a heuristic for this new MATRIX COVER problem. Borrowing some ideas from the approximation algorithms literature [4], here is a well-known approach using mathematical programming: (a) express the problem as an integer program; (b) relax the integer program as a linear program and solve it using a polynomial-time algorithm; (c) (randomly) round the linear programming solution to get an integral solution. The integer program (IP) associated with MATRIX COVER is given as: MINIMIZE $\sum_{i=1}^{T} z_i$ SUBJECT TO $\sum_{i=1}^{T} m_{i,j} z_i \geq \theta$, for $j \in [m]$, and $z_i \in \{0, 1\}$, for $i \in [T]$. The linear programming relaxation (LP) is obtained by letting $z_i \in [0, 1]$, for $i \in [T]$.

Letting $Z \in [0, 1]^T$ be the optimal LP solution and $Z^* \in \{0, 1\}^T$ be the optimal IP solution. Denote the value of the optimal solutions by $z = \sum_i Z_i$ and $z^* = \sum_i Z_i^*$, respectively. Note that z is a lower bound to z^*. We apply a method called *randomized rounding* to obtain an integral solution from the LP solution. Given Z, let $\hat{Z}$ be the integral solution as follows: for each i, let $\hat{Z}_i = 1$, with probability Z_i, and $\hat{Z}_i = 0$, with probability $1 - Z_i$. Note that the expected value of this integral solution equals to the value of the LP solution: $E[\hat{Z}] = E[\sum_i \hat{Z}_i] = \sum_i Z_i = Z$. Moreover, the constraints are satisfied *on average*: for all j, $E[\sum_i m_{i,j} \hat{Z}_i] > \theta$. Using standard large deviation inequalities [5], we claim that $\hat{Z}$ is concentrated near Z and that the constraints are somewhat satisfied. More specifically, $\Pr[|\sum_i \hat{Z}_i - \sum_i Z_i| \geq c\sqrt{T}] \leq 1/4$, whenever $c > 0.6$, and

$\Pr[(\exists j)(\sum_i m_{i,j}\hat{Z}_i \leq \alpha\theta)] \leq 1/4$, by a judicious choice of dependence between α and θ. Note that α represents a *slackness* parameter on the constraints whereas θ is related to the margin of the boosting ensemble.

So with non-negligible probability, a semi-feasible solution is obtained and $\hat{Z}$ will be within an additive factor of $\mathcal{O}(\sqrt{T})$ from the optimal LP solution. This approach allows us to trade optimality (smallness of the boosting ensemble) with feasiblity (goodness of its margin).

5 Conclusion and Future Work

In this paper we revisited the boosting pruning problem [7]. We proposed a minor modification of the powerful Kappa pruning method and reported some preliminary observations of our *weight-shifting* variant. We plan to conduct further and more extensive experiments on this problem. In addition, we have also considered the boosting pruning problem theoretically, proving that the problem is highly intractable, even to approximate. Using ideas from approximation algorithms, we proposed a theoretical heuristic. This heuristic differs from the Kappa method in that it is driven by margin considerations (instead of discrete error). This approach allows one to trade the *size* of the boosting ensemble and the *margin* of the ensemble. We plan to carry out experimental work on this margin-based algorithm.

References

1. Y. Freund and R.E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *J. Comp. System Sciences*, 55(1):119-139, 1997.
2. Y. Freund and R.E. Schapire. Experiments with a New Boosting Algorithm. *Proc. 13th Int. Conf. on Machine Learning*, 148-156, 1996.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
4. D. Hochbaum. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1997.
5. W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *J. American Stat. Assoc.*, 58:13-30, 1963.
6. V. Kann. Polynomially bounded minimization problems that are hard to approximate. *Nordic Journal of Computing*, 1:317-331, 1994.
7. D. Margineantu and T.G. Dietterich. Pruning Adaptive Boosting. *Proc. 14th Int. Conf. Machine Learning*, 211-218, 1997.
8. C.J. Merz and P.M. Murphy. UCI Repository of Machine Learning Databases. Tech. Report, U.C. Irvine, CA.
9. J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
10. J.R. Quinlan. Bagging, Boosting, and C4.5. *Proc. 13th Nat. Conf. Artificial Intelligence*, 725-730, 1996.
11. R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the Margin: a new explanation of the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651-1686, 1998.
12. R.E. Schapire and Y. Singer. Improved Boosting Algorithms using Confidence-rated Predictions. *Proc. 11th Ann. Conf. Comp. Learning Theory*, 80-91, 1998.