
Adatree 2 : Boosting to build decision trees or Improving Adatree with soft splitting rules

Etienne Grossmann*

Center for Visualization and Virtual Environments,
One, Quality Street, Suite 857
Lexington, KY, 40507, USA.
etienne@cs.uky.edu

Abstract

We extend the framework of Adaboost so that it builds a smoothed decision tree rather than a neural network. The proposed method, “Adatree 2”, is derived from the assumption of a probabilistic observation model. It avoids the problem of over-fitting that appears in other tree-growing methods by reweighing the training examples, rather than splitting the training dataset at each node. It differs from Adaboost by steering the input data towards weak classifiers that are “tuned” to the conditional probability determined by the output of previously evaluated classifiers. As a consequence, Adatree 2 enjoys a lower computation cost than Adaboost. After defining this method, we present some early experimental results and identify some issues left open for future research.

1 Boosting and decision trees

Boosting [1, 2] refers to methods to build weighed combinations of “weak classifiers”. At each training round, a weak classifier is added to the existing combination. These methods reduce the classification error on a training dataset at a geometric rate and have been shown to have good properties [3]. They are known to often not overfit and generalize well even when training is continued way beyond perfect classification of the training dataset. However, the computation cost of a boosted classifier increases linearly with the number of training rounds, so that, in computation-sensitive situations, some adaptations [4, 5, 6, 7, 8] are done to benefit from the good properties of boosting at a reasonable computation cost.

It has been noted [9] that decision tree-growing can also be viewed as a boosting mechanism, in the sense that the training error can be made to decrease geometrically. However, decision trees are notorious for over-fitting [10, Ch. 3.7] and tree “pruning”, “shrinking” [11, cited by [12]] and “smoothing” [11, 13] are techniques developed to address this problem. Even then, over-fitting remains and decision trees have been combined by boosting to improve the generalization ability.

The over-fitting of decision trees can be attributed to the fact that each node is determined by only the examples that reach it during training. Consequently, deeper nodes are built from dwindling training sets that are less and less representative of the general population.

*This is a very slightly modified version of an article submitted to NIPS 2004.

Table 1: Training and classification processes, of Adaboost, decision trees and Adatree 2.

	Classification	Training
Adaboost	<ul style="list-style-type: none"> • All classifiers are evaluated. • Output is linear combination of all classifiers. 	<ul style="list-style-type: none"> • Each classifier is trained with all examples, appropriately weighed.
Adatree 2	<ul style="list-style-type: none"> • Only weak classifiers on path of input are evaluated. • Output is linear combination of evaluated classifiers. 	<ul style="list-style-type: none"> • Each classifier is trained with all examples, appropriately weighed.
Decision Tree	<ul style="list-style-type: none"> • Only weak classifiers on path of input are evaluated. • Output is that of last classifier or (smoothed trees) a linear combination of evaluated classifiers. 	<ul style="list-style-type: none"> • Each classifier is trained with the fraction of examples that reaches it.

The proposed method, “Adatree 2” differs in that respect by diminishing the training weight of examples rather than discarding them altogether.

This re-weighting scheme is derived by assuming a probabilistic observation model that will be introduced in Section 3. Given the observation model, we derive Adatree 2 by minimizing the expected classification error on the training dataset, in a similar manner to the minimization of the classification error in Adaboost.

In Section 4, we will show that different assumptions on the observation model yield variants of Adatree 2 that are equivalent to Adaboost or to our previous method [14], which we shall call “Adatree 1” in the sequel. Table 1 compares the training and classification stages of Adaboost, decision trees and Adatree 2. This section also states the main problem that we leave open for future study, namely the choice of an observation model.

Section 5 presents some preliminary experimental results that suggest that Adatree 2 has a practical potential : its generalization ability appear less good than that of Adaboost while improving over Adatree 1. Also, Adatree 2 improves over Adaboost in terms of computation cost. Finally, some concluding notes are given in Section 6.

2 Classifier model

We consider classifiers which are weighed sums of weak classifiers $h_s()$:

$$H(X) = \text{Sgn} \left(\sum_{t=1}^{T(X)} \alpha_{s(t,X)} h_{s(t,X)}(X) \right), \quad (1)$$

where $h_{s(t,X)}(X) \in \{-1, 1\}$ and $s(t, X)$ is the index of the t^{th} nodes reached in the decision tree by input X . For example, the root node is $s(1, X) = \emptyset$, $s(2, X) = (+)$ if $h_{\emptyset}(X) = +1$, $s(3, X) = (+, -)$ if furthermore $h_{+}(X) = -1$ etc. The weights $\alpha_{s(t,X)}$ are set during training, as detailed in Section 4.

Algorithm 1 Adatree 2

Input Training examples $(X_1, y_1), \dots, (X_N, y_N)$, original weights $D_\emptyset(1), \dots, D_\emptyset(N)$, target error rate and a family of weak classifiers.

Initialization Initialize the set of leaves to $\{\emptyset\}$.

While *stopping condition not met*

1. Choose a leaf \bar{s} .
2. Train a weak learner $h_{\bar{s}}(X)$ using the p.d.f $D_{\bar{s}}(n)$ on the training set.
3. For each descendant leaf $s \in \{(\bar{s}, +), (\bar{s}, -)\}$ choose weights α_s and probabilities

$$q(s, n) = P\{h_{\bar{s}}(O_n(\omega)) = \dot{s} \mid y_n, h_{\bar{s}}(X_n), O_n(\omega) \text{ reaches } \bar{s}\},$$

where \dot{s} is the last element (*head*) of s and may be $+$ or $-$.

4. Define sample weights at each descendant leaf s :

$$D_s(n) = D_{\bar{s}}(n) q(s, n) e^{-y_n \alpha_s \dot{s}} / Z_s,$$

where Z_s normalizes the $D_s(n)$ so that they sum up to one.

5. Split the old leaf \bar{s} into the new leaves $(\bar{s}, +)$ and $(\bar{s}, -)$.

Final classifier

Define

$$h(X) = \sum_{t=1}^{T(X)} \alpha(t, X) h(t, X),$$

where $\alpha(t, X) = \alpha_{s(t+1, X)}$, $h(t, X) = h_{s(t, X)}(X)$ and the $s(t, X)$ are defined recursively by

$$\begin{aligned} s(1, X) &= \emptyset \\ s(t+1, X) &= \begin{cases} (s(t, X), \text{Sgn}(h_{s(t, X)}(X))) & \text{if } h_{s(t, X)} \text{ is defined.} \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned}$$

$T(X)$ is the largest number for which $s(t+1, X)$ is defined.

Note that $H(X)$ only depends on the leaf that is reached by X . If l is that leaf,

$$H(X) = \text{Sgn} \left(\sum_{\emptyset < s \leq l} \alpha_s \dot{s} \right) \triangleq H_l,$$

where the sum is taken over all nodes s between the leaf l (inclusive) and the root \emptyset (exclusive). Throughout this paper, we will call $\dot{s} \in \{-1, 1\}$ the last element (*head*) of a node s and \bar{s} all its previous elements (*tail*). Thus, one writes $s = (\bar{s}, \dot{s})$.

The computational model in Eq. (1) can represent that of any decision tree : a smoothed decision tree [13] will have non-zero coefficients α_s for non-leaf nodes s , while a classical decision tree such as ID3 or C.45 [15] will have $\alpha_s \neq 0$ only if s is a leaf.

This computational model also encompasses the boosting algorithms : in e.g. Adaboost [2], all classifiers $h_s(\cdot)$ at a given tree depth are equal, with weights α_s being either all equal [2, Sec 3] or depending only on whether s is a left- or a right-child [2, Sec. 4][16].

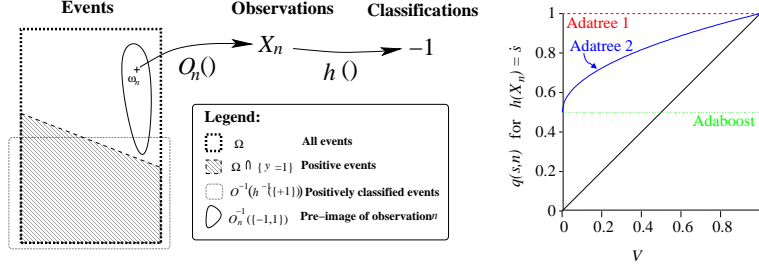


Figure 1: **Left:** Probabilistic observation model. **Right:** Possible choices of $q(s, n) = P\{h(O_n(\omega) = h_{\bar{s}}(X_n) \mid \dots)\}$ as a function of $V = V_{\bar{s}}(h_{\bar{s}}(X_n), y_n)$ - see Sec. 4.1.

3 Probabilistic learning model

With similar intentions as in Logistic Regression [17, 3] and boosting with prior knowledge [18], we assume a probabilistic observation model. Contrarily to [18], we do not consider that the class y_n of an input X_n is stochastic, but instead that X_n is the observation of a random variable $O_n(\omega)$ defined on a probability space (Ω, \mathcal{O}, P) (Figure 1, left). The random variables O_n are themselves assumed to be i.i.d. samples from the class they represent. We will be considering with particular attention the conditional probabilities that $h(O_n(\omega))$ takes a particular value, i.e.

$$P\{h(O_n(\omega)) = h(X_n) \mid y_n, h(X_n)\}$$

for probabilities conditioned on the class y_n and on the observed value of $h(X_n)$ (which may thus differ from $h(O_n(\omega))$).

How this probability is estimated is important because it determines the behavior of the learning algorithm (see Algorithm 1). Moreover, one is “free” to estimate it as one wants, by adopting different assumptions and we will discuss in the next section the consequences of various possible choices.

Given a probabilistic observation model, the natural goal during training is to minimize the expected classification error of an observation:

$$E(\text{Loss}(H(X), y)) \simeq \sum_{n=1} D_{\emptyset}(n) E(\text{Loss}(H(O_n(\omega)), y_n)) \triangleq L, \quad (2)$$

where $E(\cdot)$ is the expectation operator, $D_{\emptyset}(n)$ is the probability of an observation being drawn from $O_n(\omega)$ ¹ and $\text{Loss}(H(X), y)$ is one if $H(X) \neq y$ and zero otherwise - we will write $\text{Loss}(H(X), y) = \mathbb{I}[H(X) \neq y]$, where $\mathbb{I}[\cdot]$ is the indicator function of a predicate. Given the nature of $H(\cdot)$, Eq. (2) can be written

$$L = \sum_n D_{\emptyset}(n) \sum_{l \text{ leaf}} p(l, O_n(\omega)) \mathbb{I}[H_l \neq y_n], \quad (3)$$

where $p(l, O_n(\omega))$ is the probability that the input $O_n(\omega)$ reaches the leaf l . Pushing realism slightly aside, we adopt a practical approach and assume that

$$p(l, O_n(\omega)) = \prod_{\emptyset < s \leq l} q(s, n) \quad (4)$$

where the product is over all non-root nodes s at or below l and

$$q(s, n) = P\{h_{\bar{s}}(O_n(\omega)) = \dot{s} \mid y_n, h_{\bar{s}}(X_n), O_n(\omega) \text{ reaches } s\}$$

¹ $D_{\emptyset}(n) = 1/N$ if the O_n are i.i.d..

is the “transition” probability that $h_{\bar{s}}(O_n(\omega)) = \dot{s}$, knowing $y_n, h_{\bar{s}}(X_n)$ and that $O_n(\omega)$ reaches the node s . “ $O_n(\omega)$ reaches s ” is the event defined by the predicate $\forall u \text{ s.t. } \emptyset < u \leq s, h_{\bar{u}}(O_n(\omega)) = \dot{u}$. What Eq. (4) -the independence of the successive weak classifiers- lacks in theoretical justification is regained in convenience.

This assumption allows to bring in the framework of boosting, as developed in [19], to bound the expected loss

$$\begin{aligned}
L &\leq \sum_n D_{\emptyset}(n) \sum_{l \text{ leaf}} p(l, O_n(\omega)) e^{-\left(\sum_{\emptyset < s \leq l} -\alpha_s \dot{s}\right) y_n} \\
&= \sum_{l \text{ leaf}} \sum_n D_{\emptyset}(n) \prod_{\emptyset < s \leq l} q(s, n) e^{-\alpha_s \dot{s} y_n} \\
&= \sum_{l \text{ leaf}} \sum_n D_{\emptyset}(n) \prod_{\emptyset < s \leq l} \frac{D_s(n)}{D_{\bar{s}}(n)} Z_s \\
&= \sum_{l \text{ leaf}} \prod_{\emptyset < s \leq l} Z_s.
\end{aligned}$$

Given this bound, one is justified to choose the value of α_s that minimize Z_s during training, so as to greedily decrease L . Since $Z_s = \sum_n D_{\bar{s}}(n) q(s, n) e^{-\alpha_s \dot{s} y_n}$, it is easy to see that the optimal α_s is $e^{2\alpha_s} = A_s/B_s$, where $A_s = \sum_{y_n = \dot{s}} D_{\bar{s}}(n) q(s, n)$, $B_s = \sum_{y_n = -\dot{s}} D_{\bar{s}}(n) q(s, n)$ and $\sum_{y_n = \dot{s}}$ is the sum over all indices n s.t. $y_n = \dot{s}$. At the optimum, $Z_s = 2\sqrt{A_s B_s}$.

4 Design choices in Adatree 2

In this section, we discuss the choices that remain to be made, namely that of a probability model for $q(s, n)$ and of a splitting node s in step 1 of Algorithm 1.

4.1 Choice of the transition probability $q(s, n)$

We now consider the consequences of possible choices of $q(s, n)$. First note that, for each \bar{s} , $q((\bar{s}, +), m)$ and $q((\bar{s}, +), n)$ need only be defined for some m, n such that $y_m = 1$ and $y_n = -1$. This is because $q((\bar{s}, -), m)$ and $q((\bar{s}, -), n)$ are given by the relation $q((\bar{s}, +), n) + q((\bar{s}, -), n) = 1$ and all $q((\bar{s}, +), m')$ are equal to either $q((\bar{s}, -), m)$ or $q((\bar{s}, -), n)$.

4.1.1 Constant non-deterministic (Adaboost)

A simple choice is $q(s, n) \triangleq 1/2$, results in the same training weights on both child branches of any node \bar{s} . That is, one has $D_{(\bar{s}, +)}(n) = D_{(\bar{s}, -)}(n)$ for all \bar{s} . As a consequence, the same weak classifier will be trained on both children nodes: $h_{(\bar{s}, +)}() = h_{(\bar{s}, -)}()$. Furthermore, assuming a balanced decision tree of depth T , the classifier $H(X)$ is equal to a boosted classifier $G(X) \triangleq \text{Sgn}\left(\sum_{t=1}^T \alpha_t h_t(X)\right)$, where $\alpha_1 \triangleq \alpha_+$, $\alpha_2 \triangleq \alpha_{++}, \dots$ and $h_1() \triangleq h_{\emptyset}()$, $h_2() \triangleq h_+(), \dots$.

4.1.2 Constant deterministic (Adatree 1)

Another simple choice is $q(s, n) \triangleq \llbracket \dot{s} = h_{\bar{s}}(X_n) \rrbracket$, i.e. $\forall \omega \in \Omega, h_{\bar{s}}(O_n(\omega)) = h_{\bar{s}}(X_n)$, which corresponds to a deterministic observation model. As a result, the training weight

$D_{s+}(n)$ (resp. $D_{s-}(n)$) of all examples X_n such that $h_s(X_n) = -1$ (resp. $h_s(X_n) = 1$) is zero. This is thus equivalent to splitting the training set at each node, as in typical decision tree growing methods. The consequences of this choice, associated with a greedy cost minimization strategy, were explored in our previous work [14] where it was shown that it results in overfitting.

4.1.3 Uniform

Another natural choice is to set $q(s, n)$ to the proportion of training examples that verify $h_{\bar{s}}(X_n) = \dot{s}$:

$$q(s, n) = \frac{\sum_{m=1}^N D_{\emptyset}(m) p(\bar{s}, m) \mathbb{I}[y_m = y_n] \mathbb{I}[h_{\bar{s}}(X_m) = \dot{s}]}{\sum_{m=1}^N D_{\emptyset}(m) p(\bar{s}, m) \mathbb{I}[y_m = y_n]} \triangleq V_{\bar{s}}(\dot{s}, y_n). \quad (5)$$

Note that $q(s, n)$ only depends on $H_{\bar{s}}(X_n)$ through a single term in the numerator.

Although this choice may at first seem compelling, it implies that $D_s(n) = D_{\bar{s}}(n) / \sum_{y_m=y_n} D_{\bar{s}}(m)$, which does *not* increase the weight of misclassified examples. Running Adatrete 2 with this choice of $q(s, n)$ does not result in a decrease of the classification error on the training data.

4.1.4 Non-trivial transition probabilities

We are thus compelled to find transition probabilities that avoid the excesses of the choices proposed above. In this paper, we use a function $q(s, n)$ that depend on $h_{\bar{s}}(X_n)$ and $V_{\bar{s}}(\dot{s}, y_n)$ (Eq. (5)). The probability that $h_{\bar{s}}(O_n(\omega)) = h_{\bar{s}}(X_n)$ is assumed to be:

$$q((\bar{s}, h_{\bar{s}}(X_n)), n) = \frac{1}{2} \left(1 + \sqrt{V_{\bar{s}}(h_{\bar{s}}(X_n), y_n)} \right). \quad (6)$$

This choice (Figure 1, right) ensures that $h_{\bar{s}}(O_n(\omega)) = h_{\bar{s}}(X_n)$ is always more likely than the contrary, so that the path in the tree followed by an example during training is always the most likely path for this example. While Eq. (6) seems adequate, the search for functions $q(s, n)$, preferably that can be shown to yield boosting algorithms, remains open.

4.2 Choice of a splitting criterion

In this section, we discuss the criterion used to select the node s at which a split in the decision tree should be done. In the decision tree literature, it is generally admitted that it is preferable to split according to an entropy gain measure [10] or according to Kearns and Mansour's [9] criterion, rather than according to a classification-cost decrease criterion. Because, in the present work, (a bound on) the expected classification is minimized, a natural question is whether it is best to split a leaf to minimize the expected value of the criterion or its observed value.

One should note that minimizing the observed criterion is feasible only until its value has reached its minimum, e.g. zero error. One is thus in the usual situation of classification trees in which no training is possible once the data is perfectly classified. On the other hand, minimizing the expected criterion does not necessarily decrease the actual (empirical) classification error.

In the present work, we split the node with the highest expected error (zero look-ahead). An alternative could have been to split whichever node yields the greatest decrease in expected error (one look-ahead).

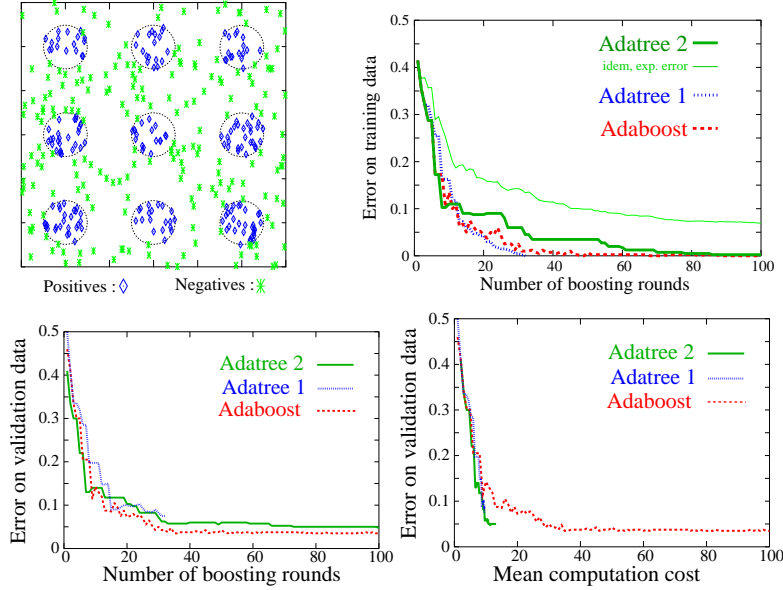


Figure 2: **Top-Left:** Example dataset. **Top-Right:** Error during training. **Bottom-Left:** Error vs. number of training rounds **Bottom-Right:** Error vs. computation cost.

5 Experimental results

This section compares experimentally Adatree 2 with Adaboost [2, Sec. 4] and Adatree 1 [14], using a 2D classification problem.

We consider datasets as shown in Figure 2 (top-left), where positive (resp. negative) examples are uniformly distributed in one of nine circles (resp. in the remaining space). Each class has 200 points.

The class of weak classifiers is the set of signs of 2D affine functions. Adatree 1 is trained until the classification error is zero (it cannot be trained further), while Adaboost and Adatree 2 are trained for 400 rounds. All weak classifiers are selected according to Kearns and Mansour’s criterion [9].

Figure 2 (top-right) shows that Adatree 2 decreases the error during training more slowly than Adaboost. This can be due to 1) less and less examples are affected by training at deeper nodes, and 2) attempting to decrease the expected error does not necessarily affect the true error. The bottom-left curves show how the error on a validation dataset decreases with the number of training rounds. These curves show that the generalization error of Adatree 2 is much better than that of Adatree 1 and is nearly as good as that of Adaboost.

The bottom-right curves of Figure 2 plot the error against the mean computation cost of each classifier. These curves are parameterized by the number of training rounds. In the case of Adaboost, the computation cost coincides with the number of training rounds, while it corresponds to the mean depth reached by input given to the Adatree classifiers. These curves show that the computation cost of Adatree 2 is nearly as low as that of Adatree 1 and that, at the bottom of the curve it has a much lower computation cost than Adaboost, while achieving similar performance.

6 Conclusions and ongoing work

We have proposed a method to grow a smoothed decision that seems to address effectively the problem of over-fitting in decision trees. Moreover, the proposed method is closely related to Adaboost, for which many theoretical results exist [3, 20].

Theoretical results are still needed to show that Adatree is truly a boosting algorithm (i.e. that it decreases the training error to zero under some assumptions on the weak learners), and to guide the choice of a transition probability function $q(s, n)$. Other issues for future research include the choice of a splitting node s and the generalization to real-valued weak classifiers.

We should in brief start benchmarking Adatree 2 on face detection problems.

Acknowledgements

This work was funded by the Kentucky Office of New Economy.

References

- [1] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *proc. International Conference on Machine Learning*, pages 148–156, 1996.
- [2] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
- [3] R.E. Schapire M. Collins and Y. Singer. Logistic regression, Adaboost and Bregman distances. In *COLT*, pages 158–169, 2000.
- [4] P. Viola and M. Jones. Robust real-time object detection. In *proc. ICCV workshop on statistical and computational theories of vision*, 2001.
- [5] S. Li, Z. Zhang, L. Zhu, H.-Y. Shum, and H. Zhang. Floatboost learning for classification. In *NIPS*, 2002.
- [6] B. McCane and K. Novins. On training cascade face detectors. In *Image and Vision Computing New Zealand*, 2003.
- [7] J. Sun, J. M. Rehg, and A. Bobick. Automatic cascade training with perturbation bias. In *proc. CVPR*, 2004.
- [8] E. Grossmann. Automatic design of cascaded classifiers. In *intl. IAPR workshop on statistical pattern recognition, ICPR*, 2004.
- [9] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *ACM Symp. on the Theory of Computing*, pages 459–468, 1996.
- [10] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [11] L. R. Bahl, P. F. Brown, P. V. deSouza, and R. L Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008, 1989.
- [12] J. M. Chambers and T. J. Hastie. *Statistical models in S*. Chapman & Hall, 1993.
- [13] W. Buntine. Learning classification trees. *Statistics and Computing*, 2(63-73), 1992.
- [14] E. Grossmann. AdaTree : boosting a weak classifier into a decision tree. In *Workshop on Learning in Computer Vision and Pattern Recognition, CVPR*, 2004.
- [15] J. R. Quinlan. *C4.5 : Programs for machine learning*. Morgan Kaufmann, 1993.
- [16] J. A. Aslam. Improving algorithms for boosting. In *Annual Conf. on Computational Learning Theory*, pages 200–207. Morgan Kaufmann, San Francisco, 2000.
- [17] J. Friedman, T. Hastie, and R.J. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 2:337–374, 2000.

- [18] R. E. Schapire, M. Rochery, M. Rahim, and N. Gupta. Incorporating prior knowledge into boosting. In *proc. ICML, 2002*.
- [19] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [20] C. Rudin, I. Daubechie, and R. Schapire. The dynamics of AdaBoost: cyclic behavior and convergence margins. *accepted by Journal of Machine Learning Research*, 2004.

A Update rules that don't work

Exercise: show that, with the weight update rule

$$D_s(n) = D_{\bar{s}}(n) e^{\beta_s \dot{s} h_{\bar{s}}(X_n)} e^{-y_n \alpha_s \dot{s}} / Z_s,$$

i.e. $q(s, n) = e^{-\beta_s \dot{s} h_{\bar{s}}(X_n)}$, the choice of β_s that minimizes the expected classifying error is $\beta_s = +\infty$. This is the Adatree1 update rule, which discard entirely examples that do not reach branch s during training.