# A simple cost function for boosting

**Marcus Frean and Tom Downs**
Department of Computer Science and Electrical Engineering
University of Queensland
Brisbane QLD 4072
Australia

## Abstract

For two-class classification problems the boosting algorithm 'AdaBoost' is equivalent to minimizing the cost-function:

$$C = \sum_{\mu} \exp(-t^{\mu}\Phi^{\mu}/2)$$

where $\mu$ indexes training items, $t \in \{\pm 1\}$ is the target and $\Phi$ is the weighted sum from individual classifiers into a composite classifier whose output is $+1$ if $\Phi > 0$ and $-1$ otherwise. Using this we show that the $D$ variables are unnecessary for AdaBoost to work, and that 're-boosting' of previous classifiers is straightforward.

## 1 Overview of Boosting

The basic idea of boosting is illustrated in figure 1. In figure 1A a number of classifiers have been combined through weighted connections to provide a joint classifier. For input pattern $\mu$, the weighted sum of the guesses made by the individual classifiers is $\phi^{\mu}$. Figure 1B depicts the addition of a further classifier whose bipolar outputs $h^{\mu}$ are fed to the joint classifier with weight $w$ reflecting the performance of this additional classifier on the training set. The weighted sum to the joint classifier is now $\Phi^{\mu} = \phi^{\mu} + h^{\mu}w$.

The overall classification is then given by the sign of the summed input to the final classifier, so it effectively acts as a linear threshold unit. Boosting algorithms endeavour to train the classifier $h$ appropriately given the targets $\{t^{\mu}\}$, and to determine the most appropriate weight $w$ to use given $\{h^{\mu}\}$.

In **AdaBoost** (Freund & Schapire 1995) one explicitly keeps track of a series of numbers $D$, one for each training item $\mu$. For the very first classifier, these are set uniformly at $1/M$ where $M$ is the number of items. We will use $D^{\mu}_{\rm old}$ to refer to the values of these numbers prior to a new classifier $h$ being generated. The new

classifier is trained to minimize the probability of errors on the full training set, but with each item being weighted by $D^{\mu}_{\rm old}$. After training, it is connected up to the joint classifier with a weighting of

$$w^{\rm Ada} \;=\; \log \frac{1-\epsilon}{\epsilon} \tag{1}$$

where

$$\epsilon = \sum_{\mu \,:\, h^{\mu}\,\rm wrong} D^{\mu}_{\rm old} \tag{2}$$

and the $D$ values are updated to

$$D^{\mu} \;=\; D^{\mu}_{\rm old} \begin{cases} \exp(-w^{\rm Ada}) & \text{if} \quad h^{\mu} \text{ is correct} \\ \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

before being normalized to sum to unity over all $\mu$. One can think of $D^{\mu}$ as 'focussing attention' on the hard examples of the training set, since it is largest for those patterns that previous classifiers have got wrong.

For future reference, note that equations 1 and 2 allow us to write

$$w^{\rm Ada} \;=\; \log \sum_{\mu:\, h^{\mu}\,\rm right} D^{\mu}_{\rm old} \;\; - \;\; \log \sum_{\mu:\, h^{\mu}\,\rm wrong} D^{\mu}_{\rm old} \tag{4}$$

and equation 3 can be written

$$D^{\mu} \;=\; D^{\mu}_{\rm old} \;\exp(-w^{\rm Ada}/2) \begin{cases} \exp(-w^{\rm Ada}/2) & \text{if } h^{\mu} \text{ is correct} \\ \\ \exp(w^{\rm Ada}/2) & \text{otherwise} \end{cases}$$

implying

$$D^{\mu} \;\propto\; D^{\mu}_{\rm old} \exp(-t^{\mu} h^{\mu} w^{\rm Ada}/2) \tag{5}$$

and since multiplicative scaling of $D$ has no effect on $w^{\rm Ada}$, the normalization of $D$ can be seen to be a non-essential feature of the algorithm.

## 2 A simple cost function for boosting

Consider the following cost function for two-class classification:

$$C \;=\; \sum_{\mu} C^{\mu} \;=\; \sum_{\mu} \exp(-t^{\mu} \Phi^{\mu}/2) \tag{6}$$

where the targets $t^{\mu}$ are either plus or minus one. This choice of cost function means:

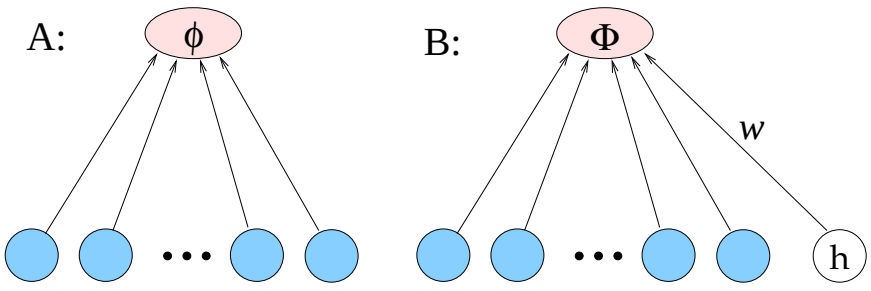


Figure 1: Architecture of boosting algorithms.

- at the outset, $\Phi$ is zero so $C^\mu = 1$ for all patterns, and $C = M$.
- the joint classifier gets $\mu$ correct if and only if $C^\mu < 1$ (*ie.* $C$ is an upper bound on the number of errors, albeit a very loose one).
- the new $C^\mu$ (with the weighted sum $\Phi$) is simply the old one (with $\phi$) multiplied by an exponential factor:

$$C^\mu \;=\; C^\mu_{\rm old} \exp(-t^\mu h^\mu w/2) \tag{7}$$

  where $C^\mu_{\rm old} \;=\; \exp(-t^\mu \phi^\mu)$.
- $C^\mu$ is closely related to the *margin* (Schapire *et. al.* 1997): if all the weights were normalized such that $\sum_i |w_i| = 1$, $C^\mu$ would be $\exp(-\text{margin}^\mu/2)$

We now show that incrementally reducing this cost function, one classifier at a time, yields the AdaBoost algorithm.

### 2.1 Finding $w$

After substituting for $\Phi$ the gradient of equation 6 is

$$\frac{\partial C}{\partial w} \;=\; -\sum_\mu t^\mu\; h^\mu\; C^\mu_{\rm old} \exp(-t^\mu h^\mu w/2)$$

Note that $t^\mu h^\mu$ is 1 if the classifier is correct on the $\mu^{\rm th}$ item and -1 otherwise, so we can write this gradient as

$$-\sum_{\mu:\, h^\mu \text{right}} C^\mu_{\rm old}\; e^{-w/2} \;+\; \sum_{\mu:\, h^\mu \text{wrong}} C^\mu_{\rm old}\; e^{w/2}$$

Setting this to zero, we can solve *exactly* for $w$.

$$w \;=\; \log \sum_{\mu:\, h^\mu \text{right}} C^\mu_{\rm old} \;-\; \log \sum_{\mu:\, h^\mu \text{wrong}} C^\mu_{\rm old} \tag{8}$$

This closed form solution means we don't need to do any iterated gradient descent to find the best weight: just solve for it directly, in one pass through the data set. This is indeed a minimum of $C$, because $\partial^2 C/\partial w^2 \;= C$ so the rate of change of the gradient is positive everywhere (except if $C = 0$ where it is zero, but then this point cannot be reached without infinite weights). Comparison of equation 8 with equation 4, and 7 with 5, confirms that the minimum cost procedure for setting $w$ is equivalent to AdaBoost.

By writing the exponential in equation 7 as $[(1-\epsilon)/\epsilon]^{-h\;t\;/2}$ and rearranging into 'right' and 'wrong' terms, it is not difficult to show that

$$C = C_{\rm old}\; 2\;\sqrt{\epsilon(1-\epsilon)}$$

where $\epsilon$ is exactly as in AdaBoost, namely the sum of $C^\mu$ over patterns $h$ gets wrong divided by $C_{\rm old}$ (because $D^\mu$ is $C^\mu$ normalised). By definition $C_0 = M$, the number of items, so we have

$$\frac{C_i}{M} \;=\; 2^i \prod_{j=1}^{i} \sqrt{\epsilon(1-\epsilon)}$$

which (as Freund & Schapire (1995) have pointed out) is a loose upper bound on the proportion of training items the joint classifier gets wrong.

### 2.2 Training $h$

Suppose $y^\mu$ is the probability that the current component classifier gets the $\mu^{\text{th}}$ instance wrong. In AdaBoost, component classifiers are trained to minimize the expected number of errors, over a warped input distribution[1]: $\sum_\mu D^\mu_{\text{old}}\ y^\mu$. In these terms the expected value of our cost function (using equation 7) is

$$\sum_\mu C^\mu_{\text{old}} \left[ y^\mu e^{w/2} \ + \ (1-y^\mu) e^{-w/2} \right]$$

Ignoring terms which don't depend on $y$ and assuming[2] $w > 0$, this is proportional to $\sum_\mu C^\mu_{\text{old}}\ y^\mu$. Since $D^\mu$ is proportional to $C^\mu$, minimizing $C$ gives the same relative weighting to training instances as AdaBoost does.

## 3 Discussion

We have shown that $D^\mu$ in AdaBoost is a normalised version of the cost function $C^\mu$ being minimized, and that $C^\mu$ effectively tracks the weighted sum into the composite classifier (times the target). AdaBoost can thus be seen as an incremental method for reducing a simple cost function - at least for two-way classifications. This has a number of interesting implications, as described in the following.

### 3.1 AdaBoost does not require the D distribution

There is no need to track the $D^\mu$ distribution explicitly. For a given pattern and a given classifier, $D^\mu$ can be replaced by $C^\mu$ and $C^\mu$ is readily available from the joint classifier: it doesn't need to be stored.

### 3.2 Earlier classifiers can be 're-boosted'

In the usual AdaBoost picture in which $D$ is taken to be a distribution over the examples, it is not obvious whether a classifier's weight to the output remains 'optimal' once other classifiers have been added. Interestingly, if exactly the same weak classifier is found twice in a row, the later one will be given a weight of zero by AdaBoost[3], so in that sense the weight found the first time was as good as possible. Nevertheless the cost-function picture makes it clear that earlier classifiers can indeed be usefully 're-boosted', without being re-trained. That is, the output weight for a given classifier involves the current values of $\phi$ and minimizes the cost function with respect to them. After other classifiers have been added the $\phi$ values are different however, and the output weight for the earlier classifier could now be improved. A simple way to do this in practice is to reset it to zero and recalculate it by one pass through the data exactly as before.

We applied re-boosting to the ionosphere data in the UCI database: there are 351 examples available and for each run we randomly took 90% of these as a training set and tested on the remaining 10%. For the purposes of this illustration, we employed random perceptrons with no training, taking advantage of the fact that AdaBoost

[1] Where it is not possible to weight the individual samples differently, as in decision trees for example, resampling with replacement is used as an approximation (eg. Drucker & Cortes, 1996)

[2] AdaBoost implicitly assumes the weight will be positive, by training individual classifiers to be good on the training set rather than bad!

[3] *eg.* calculate this weight with equation 8 using equation 7 for the 'new' $C^\mu_{\text{old}}$

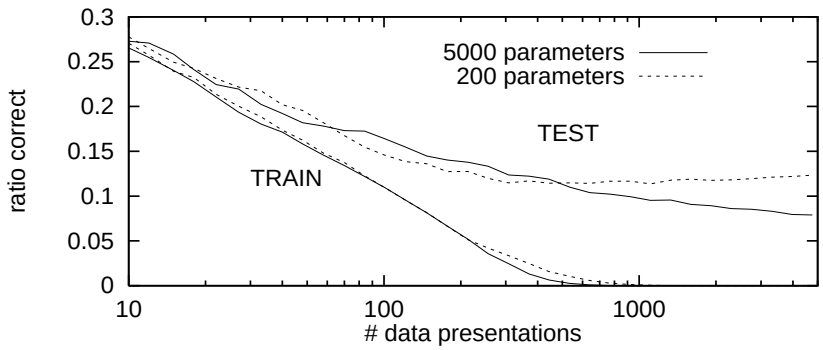


Figure 2: Errors on the training and test sets for the ionosphere data for 5000 presentations of the data set (epochs). There are two cases: in the first, 5000 classifiers are boosted just once in sequence, and in the second there are 200 each of which is revised 25 times. Behaviour on the training set is virtually the same in both cases.

copes equally well with good and bad classifiers by simply assigning a negative output weight to any classifier that achieves less than 50% accuracy. (Since each classifier is a threshold unit, this is equivalent to inverting the value of the classifier output, thereby turning a bad classifier into a good one). Since AdaBoost was used simply to determine the output weights, training was very fast because it only required one pass through the data[4]. Figure 2 shows the fraction of the training and test sets correct (averaged over 100 different runs and partitions of the data) versus the number of boosted random classifiers. Intruigingly, in the re-boosting case performance on the test set reaches a plateau soon after re-boosting begins while singular boosting continues to improve (with 25 times as many parameters). This somewhat surprising behaviour is typical of most datasets we have examined. Figure 3 shows how the margin distribution differs in the two cases.

### 3.3 AdaBoost does not have a simple interpretation in terms of maximum likelihood

If a probabilistic model has a probability $P_{t^\mu}$ of getting the $\mu^{\text{th}}$ item right, then ascent of the model's likelihood (the product of $P_{t^\mu}$ over all items in the training set) is equivalent to descent of its negative log (which is a sum over patterns in the training set, *ie.* a cost function). In our case the cost function (6) is the negative log of a 'likelihood' of the form

$$L_{(H^\mu = t^\mu)} = e^{-\exp(-t^\mu \Phi^\mu/2)} \tag{9}$$

where $H$ denotes the output of the joint classifier. It is interesting that $L_{(H=1)}$ is broadly similar to the sigmoid function more conventionally used in neural networks (see figure 4), since finding the maximum likelihood weight $w$ for a sigmoid unit is in general an iterative procedure requiring more than one pass through the data.

To be a true likelihood, one must have

$$L_{(H^\mu=1)} + L_{(H^\mu=-1)} = 1$$

[4]Note however our aim here is simply to illustrate the effect of re-boosting earlier classifiers, not to compete with other methods on this problem.

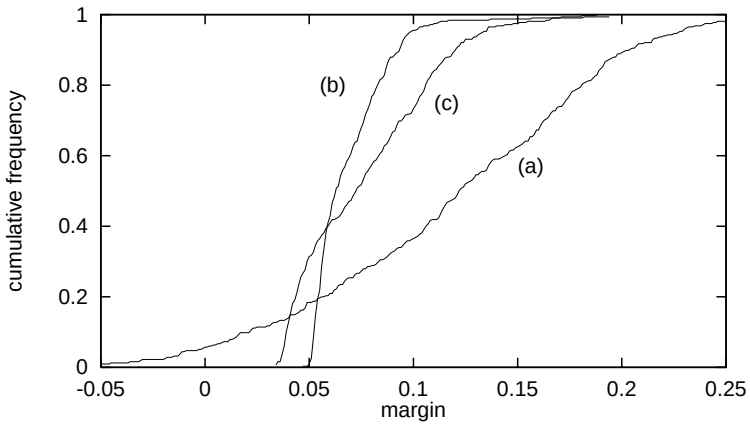


Figure 3: Cumulative distributions of the margin after (a) 200 and (b) 5000 random classifiers have each been boosted once. The steepness of the line indicates the tendency of the algorithm to maximize the margin. The third curve (c) shows the case where 200 random classifiers are each revised 25 times.

This condition is true of the above for large positive or negative $\Phi$, but not for all $\Phi$, as shown in figure 4. If we instead use a normalised version of equation 9 we can claim to be dealing with a true likelihood:

$$P_{(H^\mu=t^\mu)} = \frac{L_{(H^\mu=t^\mu)}}{L_{(H^\mu=1)} + L_{(H^\mu=-1)}} \tag{10}$$

Taking the negative log of this gives us a more appropriate cost function, which is the one described above, less a correction term which is roughly[5] gaussian in $\Phi$:

$$\begin{aligned} & C^\mu \;+\; \log\left[e^{-\exp(-\Phi^\mu/2)} + e^{-\exp(\Phi^\mu/2)}\right] \\ \approx \;\; & C^\mu \;-\; \tfrac{1}{3}\textstyle\sum_\mu e^{-(\Phi^\mu)^2/18} \end{aligned} \tag{11}$$

Minimizing $C$ on its own results in larger and larger values for $\Phi$ as the number of boosted classifiers grows. The second term is minimized by having *small* values for $\Phi$, and its gradient is only significant for "borderline" items (those with small $\Phi$). This suggests that AdaBoost may be doing something strange in the vicinity of the decision surface: the modified cost would seem to lead to smaller weights for borderline patterns than would be found by minimizing $C$ alone.

### 3.4 An analogous situation for 'boosting' regressors

It is interesting to compare section 2.1 with the analogous situation in regression. Take the regression cost function to be the squared error:

$$C = \sum_\mu (t^\mu - \Phi^\mu)^2$$

As before, splitting this $\Phi^\mu$ into $\phi^\mu + h^\mu w$ and setting the derivative (with respect to $w$) to zero gives us the output weight:

$$w = \frac{\sum_\mu (t^\mu - \phi^\mu) h^\mu}{\sum_\mu (h^\mu)^2}$$

[5]This is best seen by simply comparing the two graphs visually.

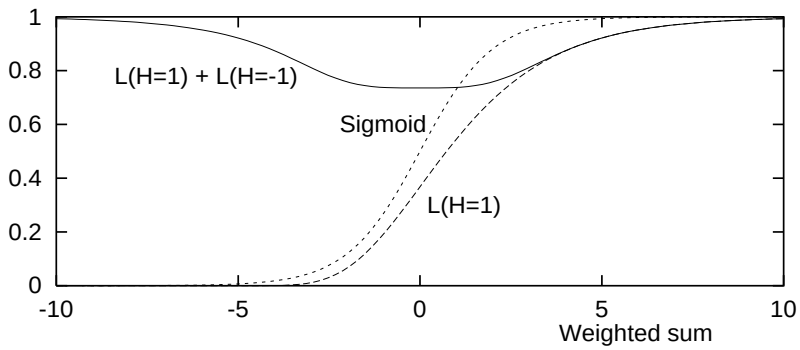


Figure 4: The sum of $L_{(H=1)}$ and $L_{(H=-1)}$, versus $\Phi$. $L_{(H=1)}$ is also shown on its own: compare this to the conventional sigmoid function $(1+\exp(-\Phi))^{-1}$, which *does* sum to unity and hence can be interpreted as a true likelihood.

Again, this is a closed form solution[6] which can be found in a single pass through the training set. This effects a reduction of $C$ from

$$C_{\text{old}} = \sum_{\mu} (t^{\mu} - \phi^{\mu})^2$$

to

$$C \;=\; C_{\text{old}} \;-\; w^2 \sum_{\mu} (h^{\mu})^2$$

Of course we could also train the individual regressors along with their output weights by gradient descent. If one is also allowed to revise earlier regressors, the algorithm then becomes standard LMS ( *ie.* backpropagation with a linear output layer, if the regressors are sigmoid units).

**Acknowledgements**

The authors would like to thank Hugo Navone and Peter Bartlett for helpful discussions in the course of this work.

**References**

Drucker, H. & Cortes, C. (1996) Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, 1996. pp. 479-485. Cambridge, MA: MIT Press.

Freund, Y. & Schapire, R.E. (1996) Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference, 1996.* pp. 148-156.

Schapire, R.E., Freund,Y., Bartlett,P. & Lee,W.S. (1997) Boosting the margin: a new explanation for the effectiveness of voting methods. *Machine Learning: Proceedings of the Fourteenth International Conference, 1997.*

Freund, Y. & Schapire, R.E. (1995) A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory: Second European Conference, EuroCOLT'95*, pp. 23-37, Springer-Verlag, 1995.

[6] Actually this is just the "delta rule", summed over all the items.