# Boosting

## MIT 15.097 Course Notes
## Cynthia Rudin

Credit: Freund, Schapire, Daubechies

Boosting started with a question of Michael Kearns, about whether a "weak learning algorithm" can be made into a "strong learning algorithm." Suppose a learning algorithm is only guaranteed, with high probability, to be slightly more accurate than random guessing. Is it possible, despite how weak this algorithm is, to use it to create a classifier whose error rate is arbitrarily close to 0? (There are some other constraints that made the problem harder, regarding how much calculation is allowed.) The answer to this question was given by Rob Schapire, with an algorithm that could do it, but wasn't very practical. Rob and Yoav Freund developed some more algorithms, e.g., Boost-by-Majority, and then developed AdaBoost.

Given:

1. examples $S = \{x_i, y_i\}_{i=1}^m$, where $y_i \in \{-1, 1\}$
2. easy access to "weak" learning algorithm $A$, producing "weak classifiers" $h \in \mathcal{H}$, $h : \mathcal{X} \to \{-1, 1\}$.
3. $\epsilon > 0$.

Goal: Produce a new classifier $H : \mathcal{X} \to \{-1, 1\}$ with error $\leq \epsilon$. Note: $H$ is not required to be in $\mathcal{H}$.

What we might do is ask the weak learning algorithm $A$ to produce a collection of weak classifiers and figure out how to combine them. But running $A$ with the same input multiple times won't be useful, for instance, if $A$ is deterministic, it will always produce the same weak classifiers over and over again. So we need to modify $A$'s input to give new information each time we ask for a weak classifier. AdaBoost does this by producing a discrete probability distribution over the examples, using that as input to the weak learning algorithm and changing it at each round.

Outline of a generic boosting algorithm:

for $t = 1...T$
construct $\mathbf{d}_t$, where $\mathbf{d}_t$ is a discrete probability distribution over indices $\{1...m\}$.
run $A$ on $\mathbf{d}_t$, producing $h_{(t)} : \mathcal{X} \to \{-1, 1\}$.
calculate

$$\begin{aligned} \epsilon_t &= \text{error}_{\mathbf{d}_t}(h_{(t)}) = Pr_{i \sim \mathbf{d}_t}[h_{(t)}(x_i) \neq y_i] \\ &=: \frac{1}{2} - \gamma_t, \end{aligned}$$

where by the weak learning assumption, $\gamma_t > \gamma_{WLA}$. (Of course, $A$ tries to minimize the error through the choice of $h_{(t)}$.)
end
output $H$

How do we design the $\mathbf{d}_t$'s? How do we create $H$? Let's see what AdaBoost does. AdaBoost (Freund and Schapire 98) is one of the top 10 algorithms in data mining, also boosted decision trees rated #1 in Caruana and Niculescu-Mizil's 2006 empirical survey.

The idea is that at each round, we increase the weight on the examples that are harder to classify - those are the ones that have been previously misclassified at previous iterates. So the weights of an example go up and down, depending on how easy the example was to classify. The easy examples can eventually get tiny weights and the hard examples get all the weight. In our notation, $d_{t,i}$ is the weight of the probability distribution on example $i$. $\mathbf{d}_t$ is called the "weight vector."

$$\begin{aligned} d_{1,i} &= \frac{1}{m} \text{ for all } i \\ d_{t+1,i} &= \frac{d_{t,i}}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_{(t)}(x_i) \text{ (smaller weights for easy examples)} \\ e^{\alpha_t} & \text{if } y_i \neq h_{(t)}(x_i) \text{ (larger weights for hard examples)} \end{cases} \\ & \quad \text{where } Z_t \text{ is a normalization constant for the discrete distribution} \\ & \quad \text{that ensures } \sum_i d_{t+1,i} = 1 \\ &= \frac{d_{t,i}}{Z_t} e^{-y_i \alpha_t h_{(t)}(x_i)} \end{aligned}$$

That's how AdaBoost's weights are defined. It's an exponential weighting scheme, where the easy examples are down-weighted and the hard examples are up-weighted.

$H$ is a linear combination of weak classifiers:

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

It's a weighted vote, where $\alpha_t$ is the coefficient assigned to $h_{(t)}$. Here, $\alpha_t$ is directly related to how well the weak classifier $h_{(t)}$ performed on the weighted training set:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right). \tag{1}$$

where

$$\epsilon_t = P_{i\sim \mathbf{d}_t}[h_{(t)}(x_i) \neq y_i] = \sum_i d_{t,i} \mathbf{1}_{[h_{(t)}(x_i)\neq y_i]} \tag{2}$$

AdaBoost stands for "Adaptive Boosting." It doesn't depend on the weak learning algorithm's assumption, $\gamma_{WLA}$, it adapts to the weak learning algorithm.

Demo

Try to remember these things about the notation: $\mathbf{d}_t$ are the weights on the examples, and $\alpha_t$ are the coefficients for the linear combination that is used to make predictions. These in some sense are both weights, so it's kind of easy to get them confused.

## Statistical View of AdaBoost

I'm going to give the "statistical view" of AdaBoost, which is that it's a coordinate descent algorithm. Coordinate descent is just like gradient descent, except that you can't move along the gradient, you have to choose just one coordinate at a time to move along.

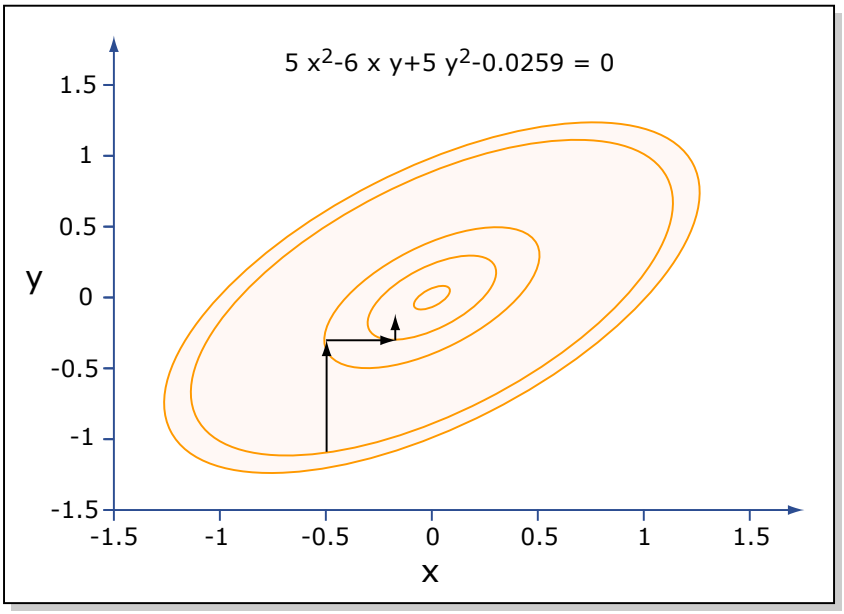



Image by MIT OpenCourseWare.

AdaBoost was designed by Freund and Schapire, but they weren't the ones who came up with the statistical view of boosting, it was 5 different groups simultaneously (Breiman, 1997; Friedman et al., 2000; Rätsch et al., 2001; Duffy and Helmbold, 1999; Mason et al., 2000).

Start again with $\{(x_i, y_i)\}_{i=1}^m$ and weak learning algorithm $A$ that can produce weak classifiers $\{h_j\}_{j=1}^n$, $h_j : \mathcal{X} \to \{-1, 1\}$. Here $n$ can be very large or even infinite. Or, $h_j$ could be very simple and produce the $j^{\text{th}}$ coordinate when $\mathbf{x}_i$ is a binary vector, so $h_j(\mathbf{x}_i) = x_i^{(j)}$.

Consider the misclassification error:

$$\text{Miscl. error} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{[y_i f(x_i) \leq 0]}$$

which is upper bounded by the exponential loss:

$$\frac{1}{m} \sum_{i=1}^{m} e^{-y_i f(x_i)}.$$

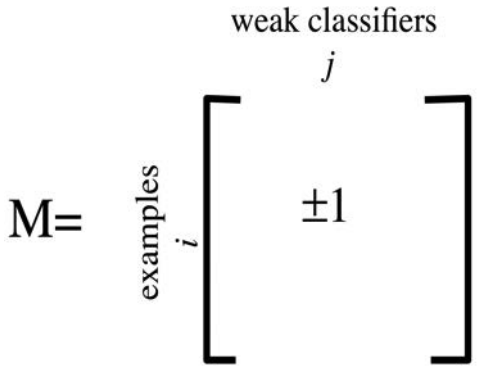
Choose $f$ to be some linear combination of weak classifiers,

$$f(x) = \sum_{j=1}^{n} \lambda_j h_j(x).$$

We're going to end up minimizing the exponential loss with respect to the $\lambda_j$'s.

The notation is going to get complicated from here on in, so try to remember what is what!

Define an $m \times n$ matrix $\mathbf{M}$ so that $M_{ij} = y_i h_j(x_i)$.




So matrix $\mathbf{M}$ encodes all of the training examples and the whole weak learning algorithm. In other words, $\mathbf{M}$ contains all the inputs to AdaBoost. The $ij^{\text{th}}$ entry in the matrix is 1 whenever weak classifier $j$ correctly classifies example $i$. (Note: we might never write out the whole matrix $\mathbf{M}$ in practice!)

Then

$$y_i f(x_i) = \sum_j \lambda_j y_i h_j(x_i) = \sum_j \lambda_j M_{ij} = (\mathbf{M}\boldsymbol{\lambda})_i.$$

Then the exponential loss is:

$$R^{\text{train}}(\boldsymbol{\lambda}) = \frac{1}{m} \sum_i e^{-y_i f(x_i)} = \frac{1}{m} \sum_i e^{-(\mathbf{M}\boldsymbol{\lambda})_i}. \tag{3}$$

Let's do coordinate descent on the exp-loss $R^{\text{train}}(\boldsymbol{\lambda})$. At each iteration we'll choose a coordinate of $\boldsymbol{\lambda}$, called $j_t$, and move $\alpha_t$ in the $j_t^{\text{th}}$ direction. So each weak classifier corresponds to a direction in the space, and $\alpha_t$ corresponds to a distance along that direction. To do coordinate descent, we need to find the direction $j$ in which the directional derivative is the steepest. Let $\mathbf{e}_j$ be a vector that is 1 in the $j^{\text{th}}$ entry and 0 elsewhere. Choose direction:

$$
\begin{aligned}
j_t &\in \text{argmax}_j \left[ -\frac{\partial R^{\text{train}}(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_j)}{\partial \alpha} \bigg|_{\alpha=0} \right] \\
&= \text{argmax}_j \left[ -\frac{\partial}{\partial \alpha} \left[ \frac{1}{m} \sum_{i=1}^{m} e^{-(\mathbf{M}(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_j))_i} \right] \bigg|_{\alpha=0} \right] \\
&= \text{argmax}_j \left[ -\frac{\partial}{\partial \alpha} \left[ \frac{1}{m} \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i - \alpha(\mathbf{M}\mathbf{e}_j)_i} \right] \bigg|_{\alpha=0} \right] \\
&= \text{argmax}_j \left[ -\frac{\partial}{\partial \alpha} \left[ \frac{1}{m} \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i - \alpha M_{ij}} \right] \bigg|_{\alpha=0} \right] \\
&= \text{argmax}_j \left[ \frac{1}{m} \sum_{i=1}^{m} M_{ij} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} \right].
\end{aligned}
$$

Create a discrete probability distribution (we'll see later it's the same as AdaBoost's):

$$
d_{t,i} = e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} / Z_t \text{ where } Z_t = \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} \tag{4}
$$

Multiplying by $Z_t$ doesn't affect the argmax. Same with $\frac{1}{m}$. So from above we have:

$$
j_t \in \text{argmax}_j \sum_{i=1}^{m} M_{ij} d_{t,i} = \text{argmax}(\mathbf{d}_t^T \mathbf{M})_j.
$$

So that's how we choose weak classifier $j_t$. How far should we go along direction $j_t$? Do a linesearch, set derivative to 0.

$$
\begin{aligned}
0 &= \frac{\partial R^{\text{train}}(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t})}{\partial \alpha} \bigg|_{\alpha_t} \\
&= -\frac{1}{m} \sum_{i=1}^{m} M_{ij_t} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i - \alpha_t M_{ij_t}} \\
&= -\frac{1}{m} \sum_{i: M_{ij_t}=1} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} e^{-\alpha_t} - \frac{1}{m} \sum_{i: M_{ij_t}=-1} -e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} e^{\alpha_t}.
\end{aligned}
$$

Get rid of the $-1/m$ and multiply by $1/Z_t$:

$$\begin{aligned} 0 &= \sum_{i:M_{ij_t}=1} d_{t,i}e^{-\alpha_t} - \sum_{i:M_{ij_t}=-1} d_{t,i}e^{\alpha_t} \\ &=: d_+e^{-\alpha_t} - d_-e^{\alpha_t} \\ d_-e^{\alpha_t} &= d_+e^{-\alpha_t} \\ e^{2\alpha_t} &= \frac{d_+}{d_-} \\ \alpha_t &= \frac{1}{2}\ln\frac{d_+}{d_-} = \frac{1}{2}\ln\frac{1-d_-}{d_-}. \end{aligned}$$

So the coordinate descent algorithm is:

$d_{1,i} = 1/m$ `for` $i = 1...m$
$\boldsymbol{\lambda}_1 = \mathbf{0}$
`loop` $t = 1...T$
 $j_t \in \text{argmax}_j(\mathbf{d}_t^T\mathbf{M})_j$
 $d_- = \sum_{M_{ij_t}=-1} d_{t,i}$
 $\alpha_t = \frac{1}{2}\ln\left(\frac{1-d_-}{d_-}\right)$
 $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \alpha_t\mathbf{e}_{j_t}$
 $d_{t+1,i} = e^{-(\mathbf{M}\boldsymbol{\lambda}_{t+1})_i}/Z_{t+1}$ `for each` $i$`, where` $Z_{t+1} = \sum_{i=1}^m e^{-(\mathbf{M}\boldsymbol{\lambda}_{t+1})_i}$
`end`

So that algorithm iteratively minimizes the exp-loss. But how is it AdaBoost? Start with $f(x)$. From coordinate descent, we notice that $\lambda_{t,j}$ is just the sum of the $\alpha_t$'s where our chosen direction $j_t$ is $j$.

$$\lambda_{t,j} = \sum_{t=1}^{T} \alpha_t \mathbf{1}_{[j_t=j]}$$

In other words, it's the total amount we've traveled along direction $j$. Thus

$$f(x) = \sum_{j=1}^{n} \lambda_{t,j}h_j(x) = \sum_{j=1}^{n}\sum_{t=1}^{T} \alpha_t \mathbf{1}_{[j_t=j]}h_j(x) = \sum_{t=1}^{T}\alpha_t \sum_{j=1}^{n} h_j(x)\mathbf{1}_{[j_t=j]} = \sum_{t=1}^{T}\alpha_t h_{j_t}(x).$$

Look familiar? There is a slight difference in notation between this and AdaBoost, but that's it. (You can just set AdaBoost's $h_{(t)}$ to coord descent's $h_{j_t}$.)

Let's look at $\mathbf{d}_t$. AdaBoost has:

$$d_{t+1,i} = \frac{d_{t,i}e^{-M_{ij_t}\alpha_t}}{Z_t} = \frac{\prod_t e^{-M_{ij_t}\alpha_t}}{m\prod_t Z_t} = \frac{e^{-\sum_t M_{ij_t}\alpha_t}}{m\prod_t Z_t} = \frac{1}{m\prod Z_t}e^{-\sum_j M_{ij}\lambda_{t,j}}$$

This means the denominator must be $\sum_i e^{-\sum_j M_{ij}\lambda_{tj}}$ because we know the $\mathbf{d}_{t+1}$ vector is normalized. So the $\mathbf{d}_t$'s for AdaBoost are the same as for coordinate descent (4) as long as $j_t$'s and $\alpha_t$'s are the same.

Let's make sure AdaBoost chooses the same directions $j_t$ as the coordinate descent algorithm. AdaBoost's weak learning algorithm hopefully chooses the weak classifier that has the lowest error. This combined with the definition for the error (2) means:

$$\begin{aligned}
j_t &\in \text{argmin}_j \sum_i d_{t,i}\mathbf{1}_{[h_j(x_i)\neq y_i]} = \text{argmin}_j \sum_{i:M_{ij}=-1} d_{t,i} \\
&= \text{argmax}_j \left[-\sum_{i:M_{ij}=-1} d_{t,i}\right] = \text{argmax}_j \left[1 - 2\sum_{i:M_{ij}=-1} d_{t,i}\right] \\
&= \text{argmax}_j \left[\left[\sum_{i:M_{ij=1}} d_{t,i} + \sum_{i:M_{ij=-1}} d_{t,i}\right] - 2\sum_{i:M_{ij}=-1} d_{t,i}\right] \\
&= \text{argmax}_j \sum_{i:M_{ij=1}} d_{t,i} - \sum_{i:M_{ij=-1}} d_{t,i} = \text{argmax}_j(\mathbf{d}_t^T\mathbf{M})_j.
\end{aligned}$$

Does that also look familiar? So AdaBoost chooses the same directions as coordinate descent. But does it go the same distance?

Look at $\alpha_t$. Start again with the error rate $\epsilon_t$:

$$\epsilon_t = \sum_i d_{t,i}\mathbf{1}_{[h_{j_t}(x_i)\neq y_i]} = \sum_{i:h_{j_t}(x_i)\neq y_i} d_{t,i} = \sum_{i:M_{ij_t}=-1} d_{t,i} = d_-$$

Starting from AdaBoost,

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t} = \frac{1}{2}\ln\frac{1-d_-}{d_-},$$

as in coordinate descent.
So AdaBoost minimizes the exponential loss by coordinate descent.

## Probabilistic Interpretation

This is not quite the same as logistic regression. Since AdaBoost approximately minimizes the expected exponential loss over the whole distribution $D$, we can get a relationship between it's output and the conditional probabilities. As in logistic regression, assume that there's a distribution on $y$ for each $x$.

**Lemma** *(Hastie, Friedman, Tibshirani, 2001)*

$$\mathbf{E}_{Y\sim D(x)}e^{-Yf(x)}$$

*is minimized at:*

$$f(x) = \frac{1}{2}\ln\frac{P(Y=1|x)}{P(Y=-1|x)}.$$

*Proof.*

$$\begin{aligned}
\mathbf{E}e^{-Yf(x)} &= P(Y=1|x)e^{-f(x)} + P(Y=-1|x)e^{f(x)} \\
0 = \frac{d\mathbf{E}(e^{-Yf(x)}|x)}{df(x)} &= -P(Y=1|x)e^{-f(x)} + P(Y=-1|x)e^{f(x)} \\
P(Y=1|x)e^{-f(x)} &= P(Y=-1|x)e^{f(x)} \\
\frac{P(Y=1|x)}{P(Y=-1|x)} &= e^{2f(x)} \Rightarrow f(x) = \frac{1}{2}\ln\frac{P(Y=1|x)}{P(Y=-1|x)}.
\end{aligned}$$

■

In logistic regression, we had

$$f(x) = \ln\frac{P(Y=1|x)}{P(Y=-1|x)}$$

so we're different by just a factor of 2.

From the Lemma, we can get probabilities out of AdaBoost by solving for $P(Y=1|x)$, which we denote by $p$:

$$\begin{aligned}
f(x) &= \frac{1}{2}\ln\frac{P(Y=1|x)}{P(Y=-1|x)} =: \frac{1}{2}\ln\frac{p}{1-p} \\
e^{2f(x)} &= \frac{p}{1-p} \\
e^{2f(x)} - pe^{2f(x)} &= p \\
e^{2f(x)} &= p(1+e^{2f(x)})
\end{aligned}$$

and finally,

$$p = P(Y = 1|x) = \frac{e^{2f(x)}}{1 + e^{2f(x)}}.$$

Recall logistic regression had the same thing, but without the 2's.

So now we can get probabilities out of AdaBoost.

- This is helpful if you want a probability of failure, or probability of spam, or probability of discovering oil.
- Even though logistic regression minimizes the logistic loss and AdaBoost minimizes the exponential loss, the formulas to get probabilities are somehow very similar.

## Training Error Decays Exponentially Fast

**Theorem** *If the weak learning assumption holds, AdaBoost's misclassification error decays exponentially fast:*

$$\frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{[y_i \neq H(x_i)]} \leq e^{-2\gamma_{WLA}^2 T}.$$

*Proof.* Start with

$$\begin{aligned} R^{\text{train}}(\boldsymbol{\lambda}_{t+1}) &= R^{\text{train}}(\boldsymbol{\lambda}_t + \alpha_t \mathbf{e}_{j_t}) = \frac{1}{m}\sum_{i=1}^{m} e^{-[\mathbf{M}(\boldsymbol{\lambda}_t + \alpha_t \mathbf{e}_{j_t})]_i} = \frac{1}{m}\sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i - \alpha_t M_{ij_t}} \\ &= e^{-\alpha_t}\frac{1}{m}\sum_{i:M_{ij_t}=1} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} + e^{\alpha_t}\frac{1}{m}\sum_{i:M_{ij_t}=-1} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i}. \end{aligned} \tag{5}$$

To summarize this proof, we will find a recursive relationship between $R^{\text{train}}(\boldsymbol{\lambda}_{t+1})$ and $R^{\text{train}}(\boldsymbol{\lambda}_t)$, so we see how much the training error is reduced at each iteration. Then we'll uncoil the recursion to get the bound.

Let's create the recursive relationship. Think about the distribution $\mathbf{d}_t$ that we defined from coordinate descent:

$$d_{t,i} = e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i}/Z_t \text{ where } Z_t = \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} \tag{6}$$

So you should think of $e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i}$ as an unnormalized version of the weight $d_{t,i}$. So that means:

$$\frac{Z_t}{m} d_+ = \frac{Z_t}{m} \sum_{i:M_{ij_t}=1} d_{t,i} = \frac{1}{m} \sum_{i:M_{ij_t}=1} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i}.$$

and we could do the same thing to show $\frac{Z_t}{m} d_- = \frac{1}{m} \sum_{i:M_{ij_t}=-1} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i}$. Plugging that into (5),

$$R^{\text{train}}(\boldsymbol{\lambda}_{t+1}) = e^{-\alpha} \frac{Z_t}{m} d_+ + e^{\alpha} \frac{Z_t}{m} d_-.$$

It's kind of neat that $Z_t/m = R^{\text{train}}(\boldsymbol{\lambda}_t)$, which we can see from the definition of $Z_t$ in (6) and the definition of $R^{\text{train}}$ in (3). Let's use that.

$$\begin{aligned} R^{\text{train}}(\boldsymbol{\lambda}_{t+1}) &= R^{\text{train}}(\boldsymbol{\lambda}_t)[e^{-\alpha} d_+ + e^{\alpha} d_-] \\ &= R^{\text{train}}(\boldsymbol{\lambda}_t)[e^{-\alpha}(1 - d_-) + e^{\alpha} d_-]. \end{aligned}$$

Remember:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - d_-}{d_-} \right), \text{ so } e^{\alpha} = \left( \frac{1 - d_-}{d_-} \right)^{1/2} \text{ and } e^{-\alpha} = \left( \frac{d_-}{1 - d_-} \right)^{1/2}.$$

Plugging:

$$\begin{aligned} R^{\text{train}}(\boldsymbol{\lambda}_{t+1}) &= R^{\text{train}}(\boldsymbol{\lambda}_t) \left[ \left( \frac{d_-}{1 - d_-} \right)^{1/2} (1 - d_-) + \left( \frac{1 - d_-}{d_-} \right)^{1/2} d_- \right] \\ &= R^{\text{train}}(\boldsymbol{\lambda}_t) 2 \left[ d_-(1 - d_-) \right]^{1/2} \\ &= R^{\text{train}}(\boldsymbol{\lambda}_t) 2 \left[ \epsilon_t(1 - \epsilon_t) \right]^{1/2}. \end{aligned}$$

Uncoil the recursion, using for the base case $\boldsymbol{\lambda}_1 = \mathbf{0}$, so that $R^{\text{train}}(\mathbf{0}) = 1$, then

$$R^{\text{train}}(\boldsymbol{\lambda}_T) = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1 - \epsilon_t)}.$$

From all the way back at the beginning (on page 2 in the pseudocode) $\epsilon_t = 1/2 - \gamma_t$. So,

$$R^{\text{train}}(\boldsymbol{\lambda}_T) = \prod_{t=1}^{T} 2\sqrt{\left( \frac{1}{2} - \gamma_t \right) \left( \frac{1}{2} + \gamma_t \right)} = \prod_t 2\sqrt{\frac{1}{4} - \gamma_t^2} = \prod_t \sqrt{1 - 4\gamma_t^2}.$$

Using the inequality $1 + x \leq e^x$, which is true for all $x$,

$$\prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \prod_t \sqrt{e^{-4\gamma_t^2}} = \prod_t e^{-2\gamma_t^2} = e^{-2\sum_{t=1}^{T} \gamma_t^2}.$$

Now, we'll use the weak learning assumption, which is that $\gamma_t > \gamma_{WLA}$ for all $t$. We'll also use that the misclassification error is upper bounded by the exponential loss:

$$\frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{[y_i \neq H(x_i)]} \leq R^{\text{train}}(\boldsymbol{\lambda}_T) \leq e^{-2\sum_{t=1}^{T} \gamma_t^2} \leq e^{-2\gamma_{WLA}^2 T}.$$

And that's our bound.

# 1 Interpreting AdaBoost

Some points:

1. AdaBoost can be used in two ways:

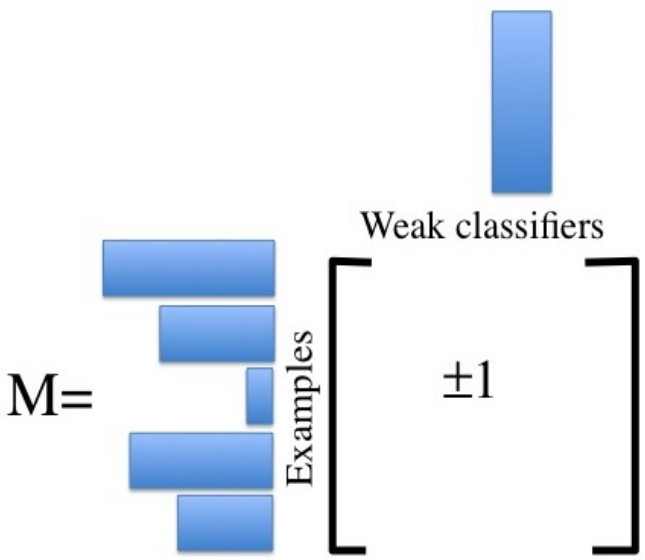
   - where the weak classifiers are truly weak (e.g., $h_j(\mathbf{x}_i) = x_i^{(j)}$).
   - where the weak classifiers are strong, and come from another learning algorithm, like a decision tree. The weak learning algorithm is then, e.g., C4.5 or CART, and each possible tree it produces is one of the $h_j$'s.

     You can think of the decision tree algorithm as a kind of "oracle" where AdaBoost asks it at each round to come up with a tree $h_{j_t}$ with low error.

     So the weak learning algorithm does the $\text{argmax}_j$ step in the algorithm. In reality it might not truly find the argmax, but it will give a good direction $j$ (and that's fine - as long as it chooses a good enough direction, it'll still converge ok).

2. The WLA does not hold in practice, but AdaBoost works anyway. (To see why, you can just think of the statistical view of boosting.)

3. AdaBoost has an interpretation as a 2-player repeated game.




weak learning algorithm chooses $j_t \equiv$ column player chooses a pure strategy

$\mathbf{d}_t \equiv$ mixed strategy for row player

4. Neither logistic regression nor AdaBoost has regularization... but AdaBoost has a tendency not to overfit. There is *lots* of work to explain why this is, and it seems to be completely due to AdaBoost's iterative procedure - another method for optimizing the exponential loss probably wouldn't do as well. There is a "margin" theory for boosting that explains a lot.

MIT OpenCourseWare
http://ocw.mit.edu

15.097 Prediction: Machine Learning and Statistics
Spring 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.