CS5339 Lecture Notes #7: Boosting

Jonathan Scarlett

March 13, 2021

Useful references:

- Blog posts by Jeremy Kun on boosting[1] and why it doesn't overfit[2]
- MIT lecture notes,[3] lectures 12 and 13
- Section 14.3 of Bishop's "Pattern Recognition and Machine Learning" book
- Chapter 10 of "Understanding Machine Learning" book
- The original paper on boosting: "A decision-theoretic generalization of on-line learning and an application to boosting" (Freund and Schapire, 1997)

1 Introduction

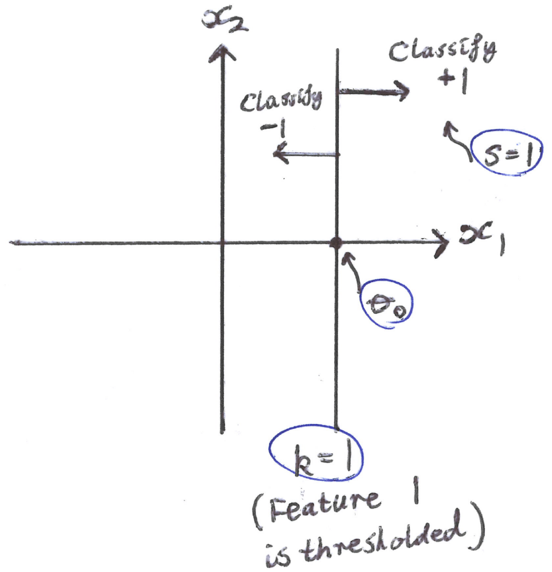
- **The key idea to be explored in this lecture:** Combine several simple classifiers to produce a powerful/complex classifier.
  - This is complementary to the idea of kernels, which can make a "simple" (linear) classifier more "complex" (non-linear) by replacing inner products by kernel evaluations
- We will look at the famous AdaBoost algorithm, which is used extensively in practical scenarios.
- There are many possible choices for the "simple classifier" that we seek to take combinations of. We will focus on arguably the simplest class of all – decision stumps:

$$h(\mathbf{x};\boldsymbol{\theta}) = \mathrm{sign}(s(x_k - \theta_0)),$$

  where $\boldsymbol{\theta} = \{s, k, \theta_0\}$. (*Note:* Don't confuse this with $\boldsymbol{\theta} \in \mathbb{R}^d$ from previous lectures. A consistent way of thinking about the two is "$\boldsymbol{\theta}$ is a vector of parameters", but here the parameters are quite different.)
  - k = index of the (only) feature that the decision is made based on
  - s = sign (to allow both combinations of $+$ on one side and $-$ on the other)
  - θ_0 = offset/threshold (classify to $+$ if above this value and $-$ if below, or vice versa)

[1] http://jeremykun.com/2015/05/18/boosting-census/
[2] http://jeremykun.com/2015/09/21/the-boosting-margin-or-why-boosting-doesnt-overfit/
[3] http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-867-machine-learning-fall-2006/lecture-notes/

In words: Just classify based on whether a single feature's value is above or below a threshold. A visual illustration:



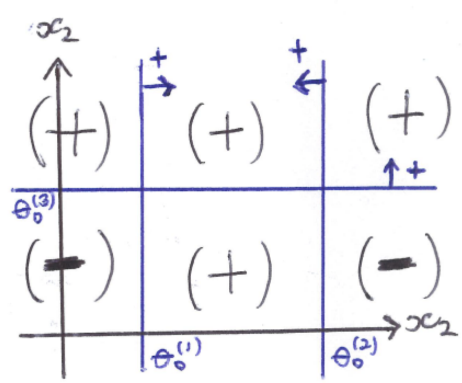
- Even though a single stump $h(\mathbf{x};\boldsymbol{\theta})$ is extremely simple (and unlikely to classify well), a weighted decision function of the form

$$f_M(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h(\mathbf{x};\boldsymbol{\theta}_m), \tag{1}$$

where $\boldsymbol{\theta}_m = (s_m, k_m, \theta_{0,m})$ for each $m = 1, \dots, M$, can perform more complex decision rules.

  - The individual $h(\mathbf{x};\boldsymbol{\theta}_m)$ are called *weak learners* or *base learners*.
  - We can interpret α_m as the "vote" of the m-th weak learner.
  - The AdaBoost algorithm provides a means for finding good $\{(\boldsymbol{\theta}_m, \alpha_m)\}_{m=1}^{M}$.

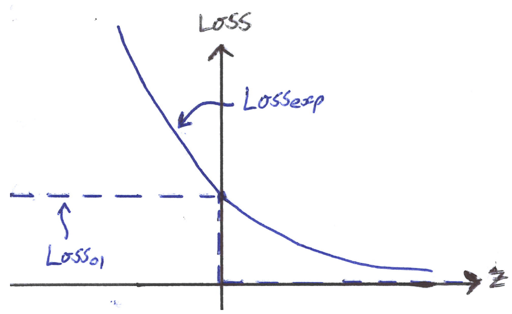
- Example. Even combining just three stumps (with equal weights), we get classifier regions that start to look quite different from a single stump:



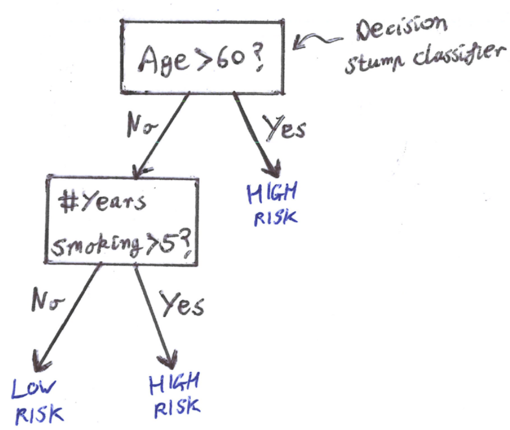
- The analysis of boosting makes use of the *exponential loss*:

$$\mathrm{Loss}_{\mathrm{exp}}(y, f(\mathbf{x})) = \exp(-yf(\mathbf{x})). \tag{2}$$

This is an upper bound to the 0-1 loss $\mathrm{Loss}_{01}(y, f(\mathbf{x})) = \mathbb{1}\{y \neq \mathrm{sign}(f(\mathbf{x}))\}$. (Here $f(\mathbf{x})$ can be thought of as representing $\boldsymbol{\theta}^T\mathbf{x}$ from earlier lectures, but now we will use more general choices.) The proof can be seen visually be interpreting the two losses as being functions of $z = yf(\mathbf{x})$:



- Side note: Decision stumps are often used to construct *decision trees* (not covered in this course):[4]



Sometimes boosting is used with decisions trees playing the role of the weak learner. Decisions trees are also a key building block in another popular technique called *random forests* (and the associated concept of *bagging*), which we will not cover.

[4]See `https://jeremykun.com/2012/10/08/decision-trees-and-political-party-classification/` for an introduction if you are interested.

2 AdaBoost

The algorithm:

- Input: Data set $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^n$ with $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, 1\}$, number of iterations[5] M
- Steps:
  1. Initialize weights $w_0(t) = \frac{1}{n}$ for $t = 1, \ldots, n$.
  2. For $m = 1, \ldots, M$, do the following:
     (a) Choose the next base learner $h(\cdot; \hat{\boldsymbol{\theta}}_m)$ as follows:
     $$\hat{\boldsymbol{\theta}}_m = \arg\min_{\boldsymbol{\theta}} \sum_{t\,:\, y_t \neq h(\mathbf{x}_t; \boldsymbol{\theta})} w_{m-1}(t). \tag{3}$$
     (b) Set $\hat{\alpha}_m = \frac{1}{2} \log \frac{1-\hat{\epsilon}_m}{\hat{\epsilon}_m}$, where $\hat{\epsilon}_m = \sum_{t\,:\, y_t \neq h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)} w_{m-1}(t)$ is the minimal value attained in (3).
     (c) Update the weights:
     $$w_m(t) = \frac{1}{Z_m} w_{m-1}(t) e^{-y_t h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m) \hat{\alpha}_m} \tag{4}$$
     for each $t = 1, \ldots, n$, where Z_m is defined so that the weights sum to one:
     $$Z_m = \sum_{t=1}^n w_{m-1}(t) e^{-y_t h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m) \hat{\alpha}_m}. \tag{5}$$
  3. Output: $f_M(\mathbf{x}) = \sum_{m=1}^M \hat{\alpha}_m h(\mathbf{x}; \hat{\boldsymbol{\theta}}_m)$, corresponding to the classifier $\hat{y} = \operatorname{sign}(f_M(\mathbf{x}))$

Intuition behind the base learner (3):

- The quantity
  $$\epsilon_m = \sum_{t\,:\, y_t \neq h(\mathbf{x}_t; \boldsymbol{\theta})} w_{m-1}(t)$$
  is called the *weighted training error*. Step (3) is choosing a base learner that "classifies best" when certain samples are treated as more important than others (as dictated by the weights $w_{m-1}(\cdot)$).
- Sometimes you might see the selection rule (3) written as
  $$\hat{\boldsymbol{\theta}}_m = \arg\min_{\boldsymbol{\theta}} \sum_{t=1}^n w_{m-1}(t) \cdot \big(- y_t h(\mathbf{x}_t; \boldsymbol{\theta}) \big) \tag{6}$$
  To see that the two are equivalent, first note that since both y_t and $h(\cdot)$ take values in ± 1, we have
  $$- y_t h(\mathbf{x}_t; \boldsymbol{\theta}) = 2\mathbb{1}\{y_t \neq h(\mathbf{x}_t; \boldsymbol{\theta})\} - 1 \tag{7}$$
  (just check the cases $y_t = h$ and $y_t \neq h$ separately). Summing over the samples $t = 1, \ldots, n$ gives
  $$\sum_{t=1}^n w_{m-1}(t) \big(- y_t h(\mathbf{x}_t; \boldsymbol{\theta}) \big) = 2\epsilon_m - 1,$$

[5] Or some other stopping criterion could be used (e.g., stop when the error on $\mathcal{D}$ is small)

so that the two rules are minimizing the same thing (up to $\times 2$ and -1 that have no effect).

Intuition behind the update (4)**:**

- Again exploiting the fact that $-y_t h(\mathbf{x}_t; \boldsymbol{\theta})$ only takes values $+1$ or -1, we can rewrite (4) as

$$w_m(t) = \frac{1}{Z_m} w_{m-1}(t) \times \begin{cases} e^{\hat{\alpha}_m} & y_t \neq h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m) \\ e^{-\hat{\alpha}_m} & y_t = h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m). \end{cases} \tag{8}$$

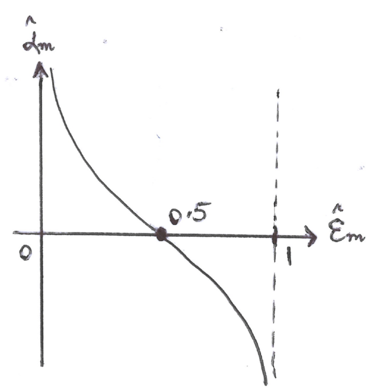
- Therefore we are doing the following:
  - If the newly chosen base learner classifies $\mathbf{x}_t$ incorrectly, increase the t-th weight
  - If the newly chosen base learner classifies $\mathbf{x}_t$ correctly, decrease the t-th weight.

  In other words, future decisions put more importance on inputs that were previously classified wrongly. (Analogy: Teacher places more teaching emphasis on topics that students scored poorly on)

- The update rule is a form of *multiplicative weights update*, which is a widespread tool that was developed independently in multiple research communities.

Intuition behind the choice of $\hat{\alpha}_m$:

- The choice $\hat{\alpha}_m = \frac{1}{2} \log \frac{1-\hat{\epsilon}_m}{\hat{\epsilon}_m}$ as a function of the training error looks like the following:

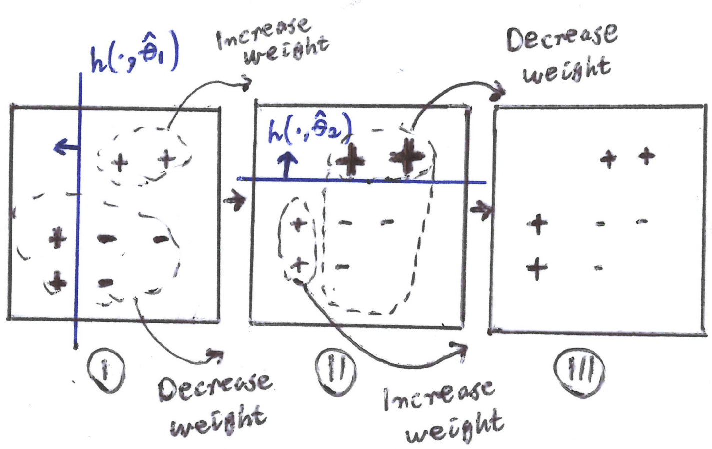


- It satisfies some intuitive properties:
  - It is a decreasing function of $\hat{\epsilon}_m$ (the better the weighted training error this base learner gives, the higher vote it is given)
  - If $\hat{\epsilon}_m = 0$, then $\hat{\alpha}_m = \infty$ (if this base learner gives classifies perfectly, put all the weight on it)
  - If $\hat{\epsilon}_m = \frac{1}{2}$, then $\hat{\alpha}_m = 0$ (if this base learner gives only classifies 50/50, put no weight on it)

- In the next section, we will see more formally how the choice of $\hat{\epsilon}_m$ arises

- Note that the regime $\hat{\epsilon}_m > \frac{1}{2}$ (which would give $\hat{\alpha}_m < 0$) is not relevant, because such a base learner cannot minimize $\hat{\epsilon}_m$ (Proof: Just consider the same base learner with s replaced by $-s$)

Illustration of the algorithm:



- A simple example with $n = 7$ samples:
- Initially $w_0 = \left(\frac{1}{7}, \dots, \frac{1}{7}\right)$.
- Then $\hat{\epsilon}_1 = \frac{2}{7}$ and hence $\hat{\alpha}_1 = \frac{1}{2}\log\frac{5}{2}$
- The updated weight is

$$w_1(t) = \frac{1}{Z_1}\begin{cases} \frac{1}{7}e^{-\frac{1}{2}\log\frac{5}{2}} & \text{correct samples} \\ \frac{1}{7}e^{\frac{1}{2}\log\frac{5}{2}} & \text{incorrect samples,} \end{cases}$$

  and a bit of analysis shows that after choosing Z_1 to satisfy $\sum_t w_1(t) = 1$, the weights simplify to $w_1(t) \in \left\{\frac{1}{10}, \frac{1}{4}\right\}$ (five values of $\frac{1}{10}$, two values of $\frac{1}{4}$)
- On the next iteration, a base classifier is chosen that classifies those with weight $\frac{1}{4}$ correctly.
- **[Code & Class Exercise]**

(Optional) Application: See Section 10.4 of the "Understanding Machine Learning" book for an example of a real-world application of AdaBoost, in the context of face recognition (Viola-Jones method).

3 Analysis of Training Error

In the following, "training error" simply refers to the proportion of mis-classified samples on $\mathcal{D}$.

Formal statement:

- **Theorem.** After M iterations, the training error of AdaBoost satisfies

$$\frac{1}{n}\sum_{t=1}^{n}\mathbb{1}\{y_t f_M(\mathbf{x}_t) \le 0\} \le \exp\left(-2\sum_{m=1}^{M}\left(\frac{1}{2} - \hat{\epsilon}_m\right)^2\right).$$

  In particular, if $\hat{\epsilon}_m \le \frac{1}{2} - \gamma$ for all m and some $\gamma > 0$, then

$$\frac{1}{n}\sum_{t=1}^{n}\mathbb{1}\{y_t f_M(\mathbf{x}_t) \le 0\} \le e^{-2M\gamma^2}.$$

- Since the left-hand side only takes values in $\left\{0, \frac{1}{n}, \dots\right\}$, the training error is zero for $M > \frac{\log n}{2\gamma^2}$.
  - The condition $\hat{\epsilon}_m \le \frac{1}{2} - \gamma$ can be interpreted as "we slightly beat random guessing" (since randomly guessing a label gives a 50% chance of success)
  - There is no assumption here of linear separability.
  - The result sounds like overfitting, but surprisingly it is not – see the following section.
- The proof proceeds in several steps as follows.

Step 1 (Convert to exponential loss):

- As stated following (2), the exponential loss upper bounds the 0-1 loss, so

$$\frac{1}{n}\sum_{t=1}^{n} \mathbb{1}\{y_t \ne \operatorname{sign}(f_M(\mathbf{x}_t))\} \le \frac{1}{n}\sum_{t=1}^{n} e^{-y_t f_M(\mathbf{x}_t)}.$$

Step 2 (An unusual equality):

- We claim that

$$\frac{1}{n}\sum_{t=1}^{n} e^{-y_t f_M(\mathbf{x}_t)} = \prod_{m=1}^{M} Z_m,$$

  where Z_m is the normalizing constant in (5).
- To see this, we can write out the updates recursively:

$$\begin{aligned} w_0(t) &= \frac{1}{n} \\ w_1(t) &= \frac{1}{n}\frac{\exp(-\hat{\alpha}_1 y_t h(\mathbf{x}_t; \boldsymbol{\theta}_1))}{Z_1} \\ w_2(t) &= \frac{1}{n}\frac{\exp(-\hat{\alpha}_1 y_t h(\mathbf{x}_t; \boldsymbol{\theta}_1))}{Z_1}\frac{\exp(-\hat{\alpha}_2 y_t h(\mathbf{x}_t; \boldsymbol{\theta}_2))}{Z_2} \\ &\vdots \\ w_M(t) &= \frac{1}{n}\frac{\exp\big(-\sum_{m=1}^{M}\hat{\alpha}_m y_t h(\mathbf{x}_t; \boldsymbol{\theta}_m)\big)}{\prod_{m=1}^{M} Z_m} = \frac{1}{n}\cdot\frac{\exp\big(-y_t f_M(\mathbf{x}_t)\big)}{\prod_{m=1}^{M} Z_m}, \end{aligned}$$

  where in the last step we substituted (1). The desired claim follows since $\sum_{t=1}^{M} w_M(t) = 1$ by construction (i.e., by the definition of the weights in the algorithm).

Step 3 (Rewriting Z_m):

- Recall the definition of Z_m in (5). We can split the sum over t into two cases: If $y_t = h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)$ then $y_t h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m) = 1$, whereas if $y_t \ne h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)$ then $y_t h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m) = -1$. Therefore, (5) simplifies to

$$\begin{aligned} Z_m &= \sum_{t\,:\,y_t \ne h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)} e^{\hat{\alpha}_m} w_{m-1}(t) + \sum_{t\,:\,y_t = h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)} e^{-\hat{\alpha}_m} w_{m-1}(t) \\ &= e^{\hat{\alpha}_m}\hat{\epsilon}_m + e^{-\hat{\alpha}_m}(1 - \hat{\epsilon}_m), \end{aligned} \tag{9}$$

  where we have used the fact that $\sum_{t\,:\,y_t \ne h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)} w_{m-1}(t) = \hat{\epsilon}_m$ by definition, and similarly $\sum_{t\,:\,y_t = h(\mathbf{x}_t; \hat{\boldsymbol{\theta}}_m)} w_{m-1}(t) = 1 - \hat{\epsilon}_m$ since the weights sum to one by definition.

- Note that $\frac{\partial Z_m}{\partial \hat{\alpha}_m} = \hat{\epsilon}_m e^{\hat{\alpha}_m} - e^{-\hat{\alpha}_m}(1-\hat{\epsilon}_m)$; setting this to zero and solving gives $\hat{\alpha}_m = \frac{1}{2}\log\frac{1-\hat{\epsilon}_m}{\hat{\epsilon}_m}$. (It is easy to check that it is a minimum and not a maximum)
  - The choice of $\hat{\alpha}_m$ is now more well-motivated: It is the one that minimizes Z_m, and therefore minimizes the upper bound on training error from Steps 1 and 2.

Step 4 (Substitute $\hat{\alpha}_m$):

- Substituting the choice of $\hat{\alpha}_m$ into (9) gives

$$\begin{aligned} Z_m &= \sqrt{\frac{1-\hat{\epsilon}_m}{\hat{\epsilon}_m}}\hat{\epsilon}_m + \sqrt{\frac{\hat{\epsilon}_m}{1-\hat{\epsilon}_m}}(1-\hat{\epsilon}_m) \\ &= 2\sqrt{\hat{\epsilon}_m(1-\hat{\epsilon}_m)} \\ &= \sqrt{1-(1-2\hat{\epsilon}_m)^2}, \end{aligned}$$

  where the last step can be verified by expanding the square.

- The last line allows us to use the convenient inequality $\sqrt{1-c^2} = \exp\big(\frac{1}{2}\log(1-c^2)\big) \le \exp\big(-\frac{1}{2}c^2\big)$ (where the last step uses $\log(1+a) \le a$). This may seem a bit mysterious – it is a mathematical trick for getting to $\prod \exp(\cdot)$; products of exponentials are convenient, because they simplify to $\exp\big(\sum \dots\big)$.

- As a result, we get $Z_m \le \exp\big(-\frac{1}{2}(1-2\hat{\epsilon}_m)^2\big)$, and combined with Steps 1 and 2, it follows that

$$\begin{aligned} \frac{1}{n}\sum_{t=1}^{n} \mathbb{1}\{y_t \ne \mathrm{sign}(f_M(\mathbf{x}_t))\} &\le \prod_{m=1}^{M} Z_m \\ &\le \prod_{m=1}^{M} \exp\bigg(-\frac{1}{2}(1-2\hat{\epsilon}_m)^2\bigg) \\ &= \exp\bigg(-2\sum_{m=1}^{M}\Big(\frac{1}{2}-\hat{\epsilon}_m\Big)^2\bigg), \end{aligned}$$

  which proves the theorem.

4 Other types of error

Weighted error relative to updated weights:

- <u>Claim.</u> The updated weights $w_m(t)$ are such that

$$\sum_{t=1}^{n} w_m(t)\mathbb{1}\{y_t \ne h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)\} = \frac{1}{2}.$$

  Hence, the base learner chosen at iteration m has the "worst possible" weighted training error with respect to the updated weights (a value $\epsilon > 0.5$ is technically worse, but it could be replaced by $1-\epsilon < 0.5$ by just flipping the sign, i.e., changing positive labels to negative and vice versa).
  - Consequence: The same base learner will never be chosen on two consecutive rounds.

- The proof:

- Since $\mathbb{1}\{y_t \neq h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)\} = \frac{1}{2}\big(1 - y_t h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)\big)$ (see (7)) and $\sum_{t=1}^n w_m(t) = 1$ (by definition), it suffices to prove that

$$\sum_{t=1}^n w_m(t) y_t h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m) = 0.$$

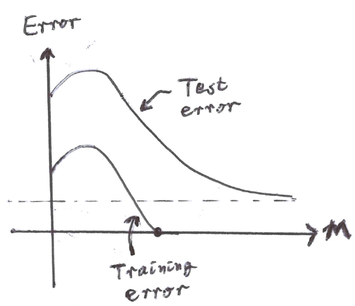
- To show this, we do the usual trick of splitting the summation into two:

$$\begin{aligned}
\sum_{t=1}^n w_m(t) y_t h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m) &= \sum_{t\,:\,y_t = h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)} w_m(t) - \sum_{t\,:\,y_t \neq h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)} w_m(t) \\
&= \sum_{t\,:\,y_t = h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)} \frac{1}{Z_m} w_{m-1}(t) e^{-\hat{\alpha}_m} - \sum_{t\,:\,y_t \neq h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)} \frac{1}{Z_m} w_{m-1}(t) e^{\hat{\alpha}_m} \\
&= \frac{1}{Z_m}(1-\hat{\epsilon}_m) e^{-\hat{\alpha}_m} - \frac{1}{Z_m}\hat{\epsilon}_m e^{\hat{\alpha}_m} \\
&= 0,
\end{aligned}$$

where the second line substitutes the update equation (8), the third line applies the definition $\hat{\epsilon}_m = \sum_{t\,:\,y_t \neq h(\mathbf{x}_t;\hat{\boldsymbol{\theta}}_m)} w_{m-1}(t)$, and the last line uses the fact that $\hat{\alpha}_m$ was chosen by equating $\hat{\epsilon}_m e^{\hat{\alpha}_m} - e^{-\hat{\alpha}_m}(1-\hat{\epsilon}_m)$ with zero (see Step 3 above).

Test error (non-examinable advanced material):

- For typical learning algorithms, making the training error too small makes the *test error* (i.e., the error rate on *unseen* data) large.
  - This is known as *overfitting*, and will be explored more in the coming lectures.
- For boosting, we typically observe the following surprising phenomenon:



  - Test error continues decreasing even after getting zero training error!
- Let's try to get an intuitive understanding of this. Define

$$\tilde{f}_M(\mathbf{x}) = \frac{\sum_{m=1}^M \hat{\alpha}_m h(\mathbf{x};\hat{\boldsymbol{\theta}}_m)}{\sum_{m=1}^M \hat{\alpha}_m} \in [-1,1], \qquad \text{margin}(t) = y_t \tilde{f}_M(\mathbf{x}_t) \in [-1,1].$$

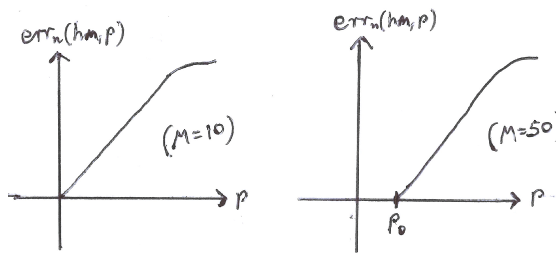
We have $\text{margin}(t) > 0$ if and only if $\mathbf{x}_t$ is classified correctly. But the closer $\text{margin}(t)$ is to one, the "further away" the classifier is from incorrectly classifying it.

- Now define the following stricter notion of error:

$$\mathrm{err}_n(f_M;\rho) = \frac{1}{n}\sum_{t=1}^{n} \mathbb{1}\{\mathrm{margin}(t) \le \rho\}.$$

Note that the normal training error is simply $\mathrm{err}_n(f_M;0)$.

- *Implicitly, AdaBoost is minimizing* $e^{-\mathrm{margin}(t)}$. This means that even after achieving $\mathrm{err}_n(f_M;0) = 0$, the algorithm continues to decrease $\mathrm{err}_n(f_M;\rho)$ for $\rho > 0$:



  - A higher margin leads to better generalization, and boosting naturally increases the margin.

- A formal statement on the test error of AdaBoost is given in the 1998 paper by Schapire, Freund, Bartlett, and Lee – the final author is a professor at NUS.