Boosting

STAT 37710 / CMSC 35300
Rebecca Willett and Yuxin Chen

Introduction

Like with bagging, boosting combines multiple classifiers trained on different "versions" of the training data

- in bagging, different versions of the data correspond to different bootstrap samples and we aggregate by treating each individual classifier $\hat{h}_b$ equally

$$\text{e.g. } \hat{h}(x) = \text{sign}\left(\sum_{b=1}^{B} \hat{h}_b(x)\right)$$

> unlike the bagging lecture, we assume (to keep equations simple) that labels $y_i \in \{-1, 1\}$ instead of $\{0, 1\}$

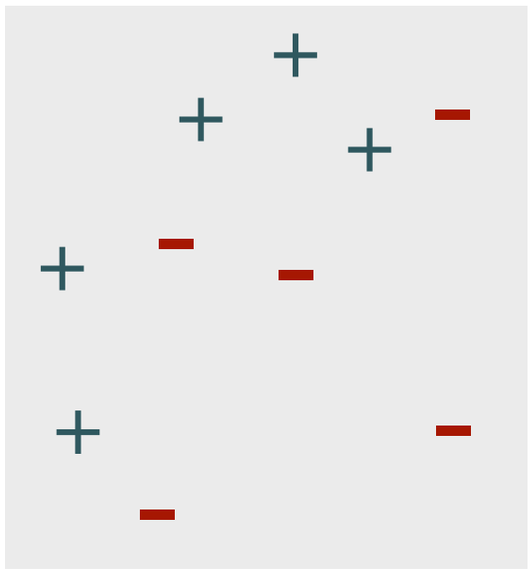
- in boosting, different versions of the data correspond to different weights applied to different samples and we aggregate by computing a weighted sum of the individual classifiers

$$\text{e.g. } \hat{h}(x) = \text{sign}\left(\sum_{b=1}^{B} \alpha_b \hat{h}_b(x)\right)$$

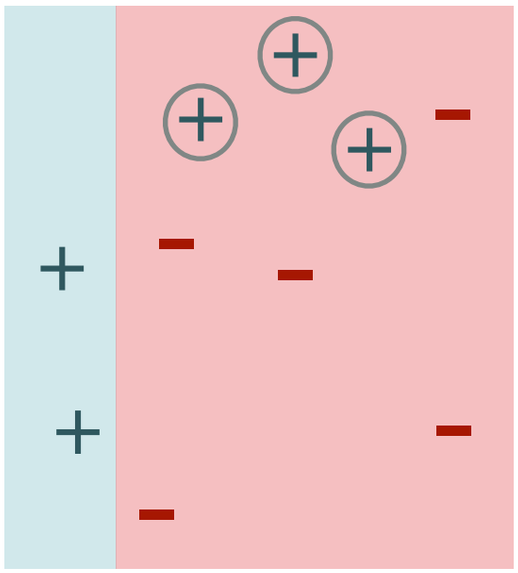
The key idea behind the weights is that if individual classifiers $\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_t$ are regularly misclassifying a sample x_i, then we want to give x_i a large weight when we train $\hat{h}_{t+1}$ to help ensure it is correctly classified.

AdaBoost Illustration

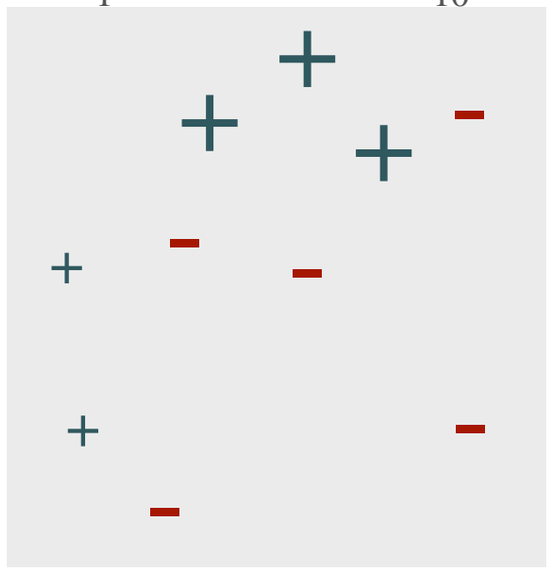
$(x_1, y_1), \ldots, (x_{10}, y_{10})$



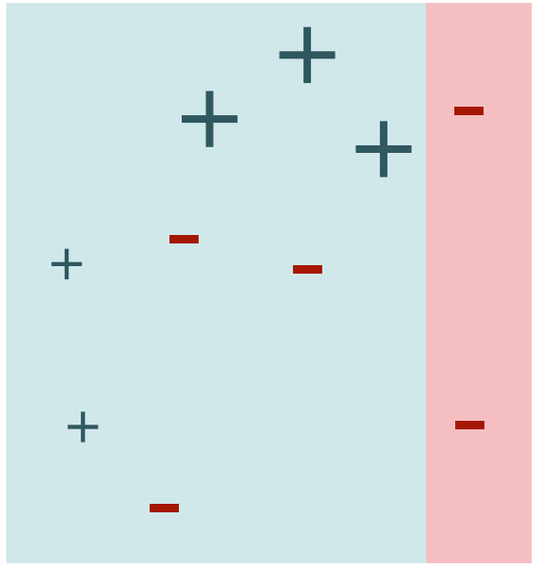
$h_1(x)$



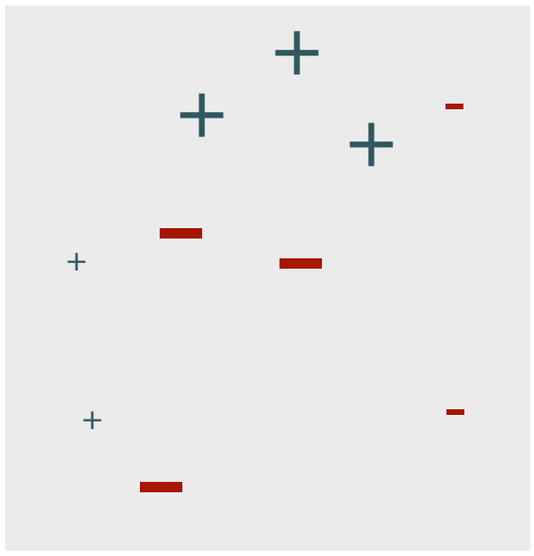
$(x_1, w_1^{(2)} y_1), \ldots, (x_{10}, w_{10}^{(2)} y_{10})$



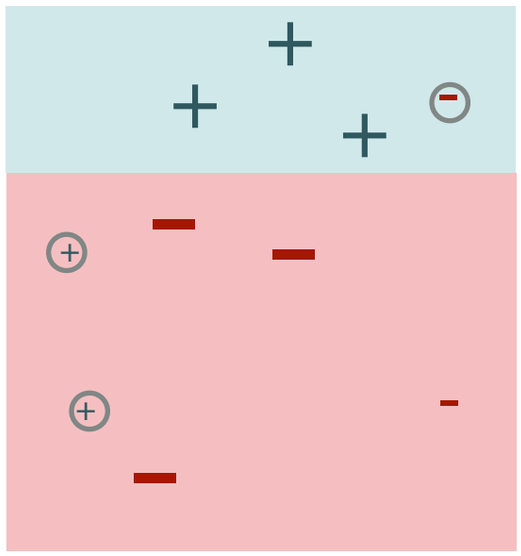
$h_2(x)$



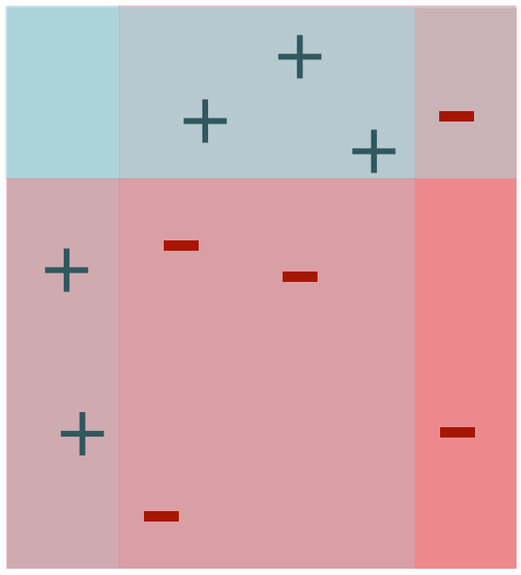
$(x_1, w_1^{(3)} y_1), \ldots, (x_{10}, w_{10}^{(3)} y_{10})$



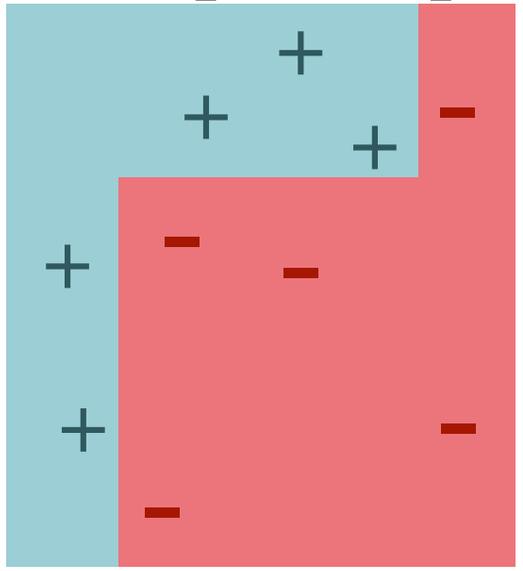
$h_3(x)$



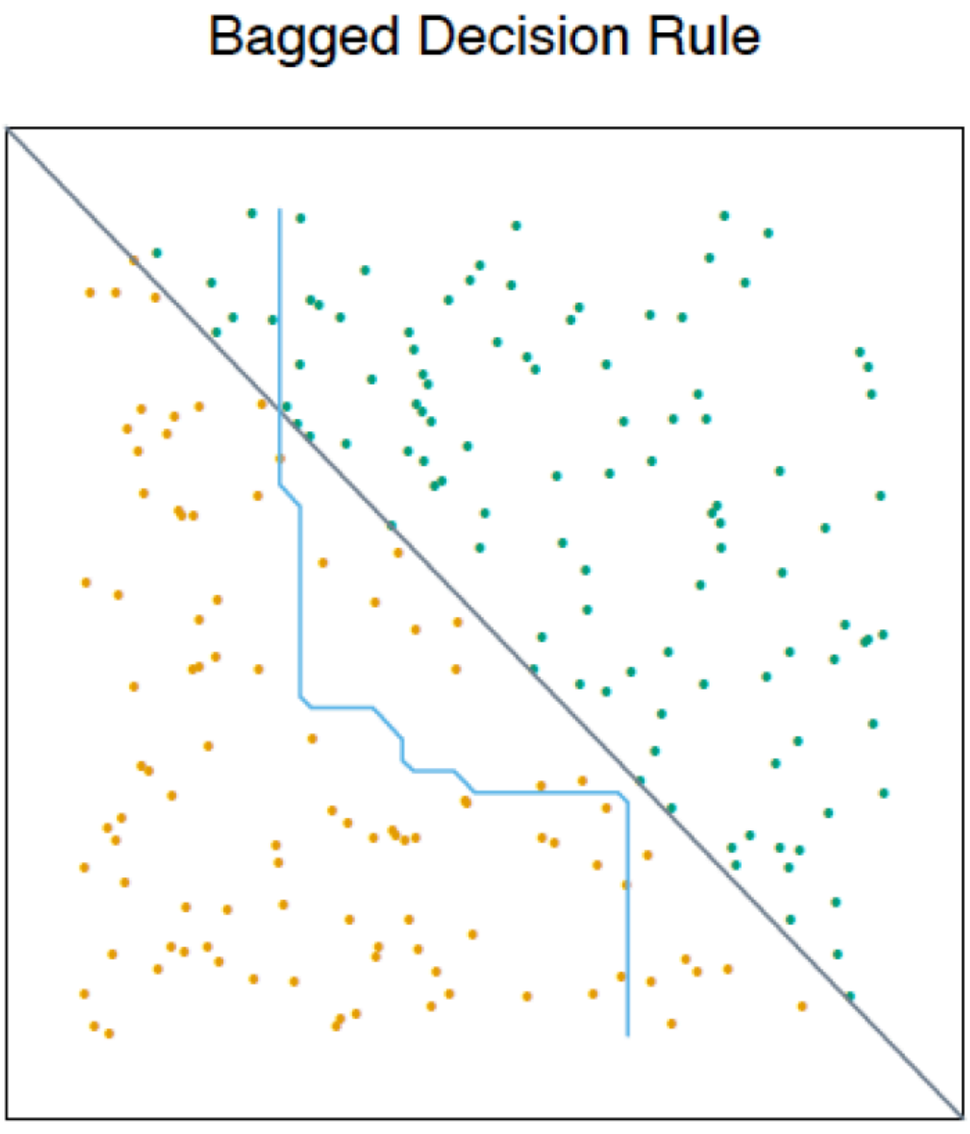
$\sum_{b=1}^{3} h_b(x)$

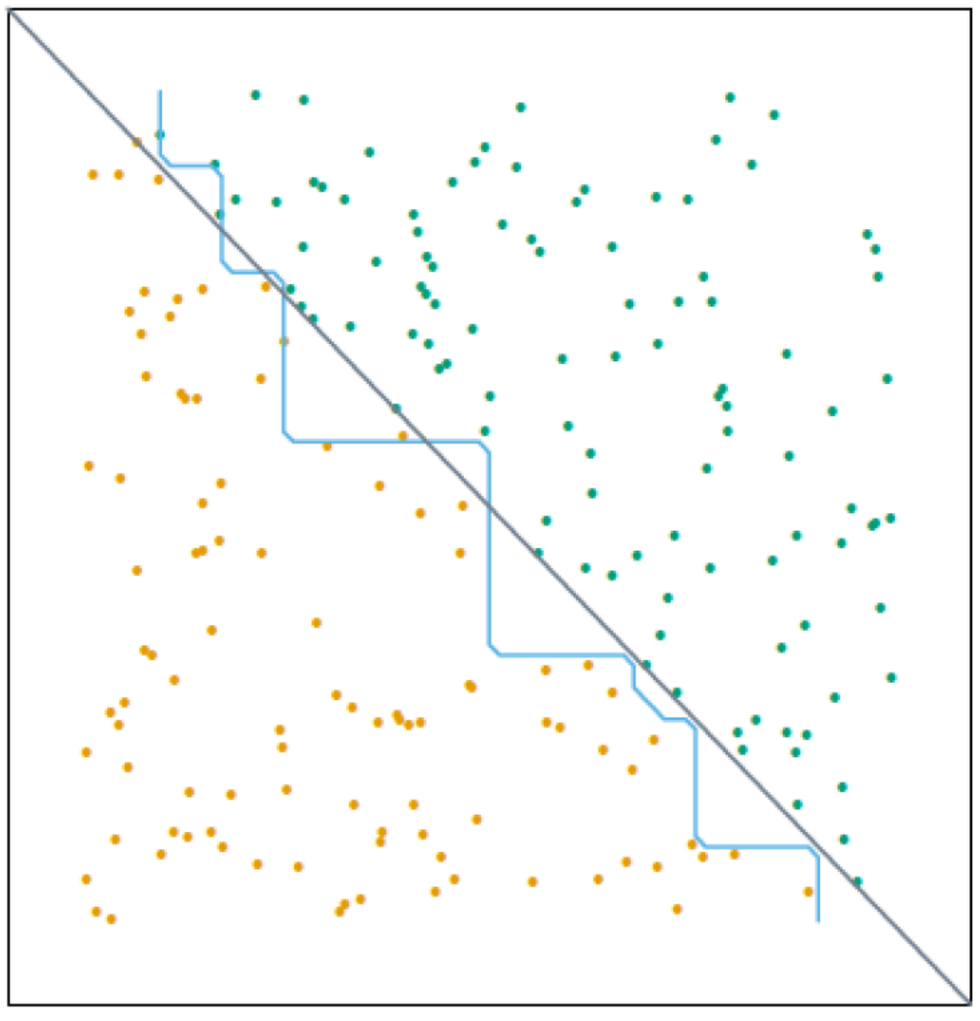


$\text{sign}\left[\sum_{b=1}^{3} h_b(x)\right]$



Bagging vs. Boosting





Weighted training data

It is instructive to think about this idea in the context of decision trees.

For each sample i, $i = 1, 2, \ldots, n$, imagine we have a weight $w_i \geq 0$.

Higher weight $\longrightarrow$ more important that this sample be assigned an accurate label

In a tree, consider a region R_j. Before we computed the fraction of samples in R_j with label = 1 as

$$\hat{p}(R_j) = \frac{\sum_{x_i \in R_j} \mathbb{1}_{\{y_i = 1\}}}{\sum_{x_i \in R_j} 1}$$

} called n_j in previous notes

$$\Longrightarrow \text{ label } c_j = \begin{cases} 1 & \text{if } \hat{p}(R_j) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

empirical misclassification error

$$= \begin{cases} \hat{p}(R_j) & \text{if } c_j = 0 \\ 1 - \hat{p}(R_j) & \text{if } c_j = 1 \end{cases}$$

A WEIGHTED proportion can be computed as

$$\hat{p}(R_j) = \frac{\sum_{x_i \in R_j} w_i \, \mathbb{1}_{\{y_i = 1\}}}{\sum_{x_i \in R_j} w_i}$$

$$\Longrightarrow \text{ label } c_j = \begin{cases} 1 & \text{if } \hat{p}(R_j) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

empirical weighted misclassification error

$$= \begin{cases} \hat{p}(R_j) & \text{if } c_j = 0 \\ 1 - \hat{p}(R_j) & \text{if } c_j = 1 \end{cases}$$

Another way to think about weights: assume we have a loss function $L(y, \hat{y})$.

Instead of minimizing the empirical risk $\hat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))$, we minimize the weighted empirical risk $\hat{R}_n(h) = \frac{\sum_{i=1}^{n} w_i \, L(y_i, h(x_i))}{\sum_{i=1}^{n} w_i}$

Adaboost

Basic tree boosting algorithm (AdaBoost)

Given: training samples (x_i, y_i), $i = 1, \dots, n$, with $x_i \in \mathbb{R}^p$, $y_i \in \{-1, 1\}$

Initialize: $w_i = \frac{1}{n}$ for $i = 1, \dots, n$

For $b = 1, \dots, B$

- Learn classification tree $\hat{h}_b$ using weighted training samples with weights $w_1, \dots, w_n$
- Compute weighted misclassification error

$$e_b = \frac{\sum_{i=1}^{n} w_i \mathbb{1}\{y_i \neq \hat{h}_b(x_i)\}}{\sum_{i=1}^{n} w_i}$$

- Let $\alpha_b = \log \frac{1 - e_b}{e_b}$
- Update weights: $w_i \longleftarrow w_i \cdot \exp\left(\alpha_b \mathbb{1}\{y_i \neq \hat{h}_b(x_i)\}\right)$, $i = 1, \dots, n$

Return $\hat{h}(x) = \text{sign}\left(\sum_{b=1}^{B} \alpha_b \hat{h}_b(x)\right)$

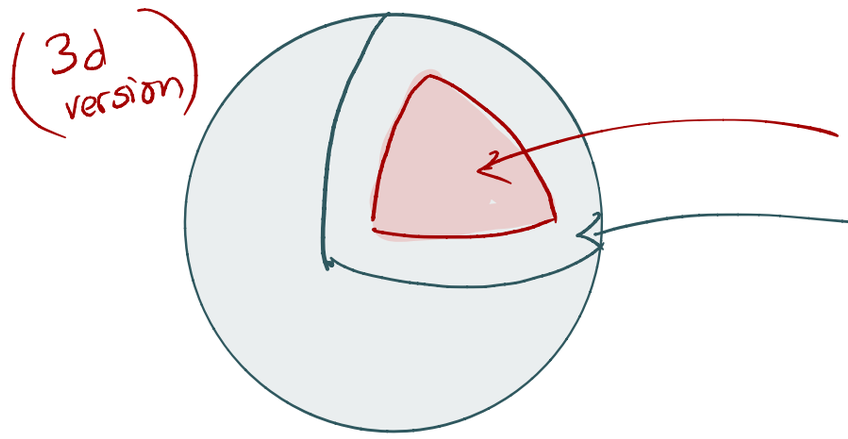
Example

Example: Boosting stumps (ESL p339)

$n = 2000$, $p = 10$. $X_i \sim \mathcal{N}(0, I_p)$,

$$y_i = \begin{cases} 1 & \text{if } \|X_i\|^2 > 9.34 \\ -1 & \text{otherwise} \end{cases}$$

9.34 → median of χ^2 distrib. w/ 10 degrees of freedom



Train a "stump" classifier — classification tree with 1 split (2 leaves) (This is a bad classifier — best stump has misclassification rate 45.8 %)

With boosting, get misclassification error rate of ~ 6% with B = 400

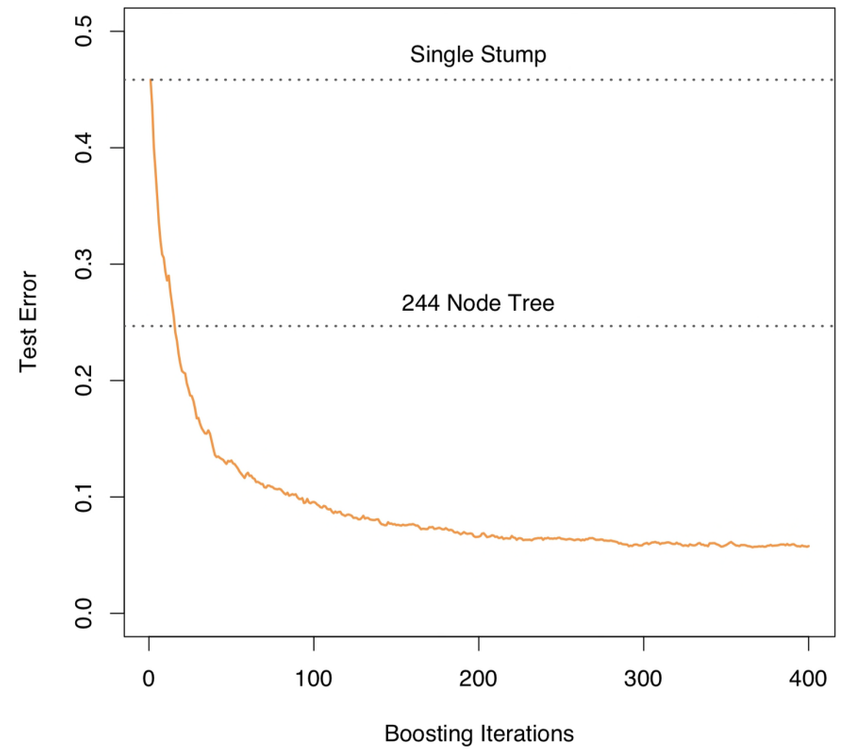


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Why does boosting work?

It's clear that giving points we've misclassified a higher weight, we seek classifiers in future iterations that correctly classify the point.

Consider exponential loss: $L(y_i, h(x_i)) = \exp(-y_i h(x_i))$
Imagine we have a finite collection of possible stump classifier trees $h_1(x), h_2(x), \ldots, h_M(x)$.

We can think of boosting as follows: $\hat{h}$ will be a weighted sum of trees, so we need to select the trees + weights.

- For a given tree h_j, choose tree weight β_j as $\beta_j = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \exp(-y_i \beta h_j(x_i))$

- Choose 1st tree: $j_1 = \underset{j \in [1,\ldots,M]}{\operatorname{argmin}} \sum_{i=1}^{n} \exp(-y_i \beta_j h_j(x_i)) = \underset{j}{\operatorname{argmin}} \sum_i L_i(y_i, \beta_j h_j(x_i))$

  equivalently: $(\beta_1, \hat{h}_1) = \underset{\beta, h}{\operatorname{argmin}} \sum_{i=1}^{n} \exp(-y_i \beta h(x_i)) = \underset{\beta, h}{\operatorname{argmin}} \sum_{i=1}^{n} L_i(y_i, \beta h(x_i))$

- Choose k^{th} tree: Let $\tilde{h}_{k-1}(x) := \sum_{b=1}^{k-1} \beta_b \hat{h}_b(x)$ be our estimate so far

  set $(\beta_b, \hat{h}_b) = \underset{\beta, h}{\operatorname{argmin}} \sum_{i=1}^{n} \exp(-y_i [\tilde{h}_{k-1}(x_i) + \beta h(x_i)])$

  $= \underset{\beta, h}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(k)} \exp(-y_i \beta h(x_i))$ where $w_i^{(k)} := \exp(-y_i \tilde{h}_{k-1}(x_i))$

  does not depend on β or h $\Rightarrow$ think of as weight on samples!

Solving for h

How to solve ⊛ $\underset{\beta, h}{\text{argmin}} \sum_{i=1}^{n} w_i^{(k)} \exp(-y_i \beta h(x_i))$?

1st, consider fixing β and solving for h:

$$\hat{h}_b = \underset{h}{\text{argmin}} \sum_{i=1}^{n} w_i^{(k)} \exp(-y_i \beta h(x_i)) = \underset{h}{\text{argmin}} \sum_{i: y_i = h(x_i)} w_i^{(k)} e^{-\beta} + \sum_{i: y_i \neq h(x_i)} w_i^{(k)} e^{\beta}$$

$$= \underset{h}{\text{argmin}} \; (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\} + e^{-\beta} \sum_{i=1}^{n} w_i^{(k)}$$

$$= \underset{h}{\text{argmin}} \sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\} \quad \text{(independent of } \beta\text{!)}$$

Let $\ell_h^{(k)} :=$ error of $h = \dfrac{\sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\}}{\sum_{i=1}^{n} w_i^{(k)}}$

Now back to ⊛: $\underset{\beta, h}{\text{argmin}} \sum_{i=1}^{n} w_i^{(k)} \exp(-y_i \beta h(x_i)) = \underset{\beta, h}{\text{argmin}} \; (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\} + e^{-\beta} \sum_{i=1}^{n} w_i^{(k)}$

$$= \underset{\beta, h}{\text{argmin}} \; (e^{\beta} - e^{-\beta}) \underbrace{\frac{\sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\}}{\sum_{i=1}^{n} w_i^{(k)}}}_{err_h^{(k)}} + e^{-\beta}$$

Solving for β

now solve for β: $\quad \underset{\beta}{\operatorname{argmin}} \; (e^{\beta} - e^{-\beta})\, err_h + e^{-\beta}$

$$\frac{d}{d\beta} = (e^{\beta} + e^{-\beta})\, err_h - e^{-\beta} = 0 \implies e^{-\beta} = (e^{\beta} + e^{-\beta})\, err_h \implies err_h = \frac{e^{-\beta}}{e^{\beta} + e^{-\beta}}, \; 1 - err_h = \frac{e^{\beta}}{e^{\beta} + e^{-\beta}}$$

$$\implies \frac{1 - err_h}{err_h} = \frac{e^{\beta}}{e^{-\beta}} = e^{2\beta} \implies \beta = \frac{1}{2} \log \frac{1 - err_h}{err_h}$$

All together:

$$\hat{h}_k = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(k)} \mathbb{1}\{y_i \neq h(x_i)\}, \quad \beta_k = \frac{1}{2} \log \frac{1 - err_{\hat{h}_k}}{err_{\hat{h}_k}}$$

Now $\tilde{h}_k(x) = \tilde{h}_{k-1}(x) + \beta_k \hat{h}_k(x)$

This implies weights for next round are $w_i^{(k+1)} = \exp(-y_i \tilde{h}_k(x_i)) = \exp(-y_i \tilde{h}_{k-1}(x)) \exp(-y_i \beta_k \hat{h}_k(x))$

$$= w_i^{(k)} \cdot \exp(-y \beta_k \hat{h}_k(x))$$

Also, note $\beta_k(-y_i \hat{h}_k(x_i)) = (2\,\mathbb{1}\{y_i \neq \hat{h}_k(x_i)\} - 1)\beta_k$

$$\implies w_i^{(k+1)} = w_i^{(k)} \cdot \exp\left(\alpha_k \mathbb{1}\{y_i \neq \hat{h}_k(x_i)\}\right) \cdot \underbrace{\exp^{-\beta_k}}_{\text{constant (doesn't change w/ } i) \implies \text{ignore}}$$

where $\alpha_k := \beta_k = \log \frac{err_{\hat{h}_k}}{1 - err_{\hat{h}_k}}$ ⇐ AdaBoost weight update!

Another perspective

Imaging that instead of exponential loss $L(y,\hat{y}) = e^{-y\hat{y}}$ we were using a generic, differentiable loss $L(y,\hat{y})$. Now given our current predictor $\tilde{h}_{k-1}(x)$, we have empirical risk $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \tilde{h}_{k-1}(x_i))$. We want to find a new predictor $h(x)$ so the sum $\tilde{h}_{k-1}(x) + h(x)$ pushes to loss towards its minimum as quickly as possible.

Key idea: gradient descent: choose h in the direction of the negative gradient of the loss.

Specifically, for each i, compute negative gradient $-g(x_i) = -\frac{\partial L(y_i, \tilde{h}_{k-1}(x_i))}{\partial \tilde{h}_{k-1}(x_i)}$

Loss	Negative Gradient
$L(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$	$\Rightarrow \frac{\partial L}{\partial \hat{y}} = -(y-\hat{y}) \Rightarrow -\frac{\partial L(y_i, \tilde{h}_{k-1}(x_i))}{\partial \tilde{h}_{k-1}(x_i)} = y_i - \tilde{h}_{k-1}(x_i)$
$L(y,\hat{y}) = e^{-y\hat{y}}$	$\Rightarrow \frac{\partial L}{\partial \hat{y}} = -ye^{-y\hat{y}} \Rightarrow -\frac{\partial L(y_i, \tilde{h}_{k-1}(x_i))}{\partial \tilde{h}_{k-1}(x_i)} = y_i \exp\{-y_i \tilde{h}_{k+1}(x_i)\}$

Gradient boosting

Given the negative gradients $-g(x_i)$, we fit a model h to them to update our boosted model.

The resulting method is called

GRADIENT BOOSTING

XGBOOST is a python package for "extreme" gradient boosting.

"Knowing logistic regression and xgboost gets you 95% of the way to a winning Kaggle submission for most competitions"

— Folk Wisdom

Gradient Boosting

Start with an initial model, e.g. $\tilde{h}_1(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$

for $b = 1, 2, \ldots$

calculate negative gradients

$$-g(x_i) = -\frac{\partial L(y_i, \tilde{h}_b(x_i))}{\partial h_b(x_i)}$$

fit a model h_b (e.g. tree) to negative gradients: $h_b = \underset{h}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(-g(x_i), h(x_i))$

$$\tilde{h}_{b+1}(x) = \tilde{h}_b(x) + \beta_b h_b(x)$$

where β_b is a step size parameter we find computationally to minimize the loss.

if $\tilde{h}_{b+1} \approx \tilde{h}_b$, STOP