COMS 4771 Fall 2025
Boosting

Ensemble method

- Ensemble method: method for training several individual predictors so that their combination works together as a good predictor
- Q1: How to combine the individual predictors?
- Q2: How to train the individual predictors?

How to combine predictors?

- Model averaging: final predictor F is average of individual predictors (or majority/plurality vote in the case of classifiers)
- Linear combination: treat predictors as features in linear model …
- …

How to train individual predictors?

- Bagging: bootstrap resampling + model averaging
  - Train individual predictors using bootstrap resampling of training data
  - (In principle, all predictors could be trained in parallel!)
- …

Boosting

- Boosting: type of ensemble method in which individual predictors are trained sequentially (i.e., one after another)
  - (Term "boosting" only really makes sense in original theoretical context)
  - Training objectives for predictors will not all be the same!
  - Objective for t^{th} predictor will depend on previous $t-1$ predictors
- Typically combined in (weighted) majority vote or linear combination

Many different "boosting" methods

- Learn(ϵ, δ, EX)
- Boost-by-Majority
- AdaBoost
- LogitBoost
- MadaBoost
- RankBoost
- MM Boosting
- SmoothBoost
- BrownBoost
- SMartiBoost
- Gradient Boosting
- Stochastic Gradient TreeBoost
- DOOM II
- L2Boost
- Regularized Greedy Forest
- LightGBM
- XGBoost
- …

AdaBoost

AdaBoost ("Adaptive Boosting") [Freund & Schapire, 1997]

- Training data [binary classification]: $\mathcal{S} := ((x^{(i)}, y^{(i)}))_{i=1}^{n}$ from $\mathcal{X} \times \{\pm 1\}$
- Initial "example weights": $D_1(i) = 1/n$ for each $i \in \{1, \dots, n\}$
- For $t = 1, \dots, T$:
  - Run "base learner" on D_t-weighted training data $\mathcal{S}$ to get $h_t : \mathcal{X} \to \{\pm 1\}$
  - Update example weights:

$$z_t := \sum_{i=1}^{n} D_t(i)\, y^{(i)} h_t(x^{(i)}), \qquad \alpha_t := \frac{1}{2} \ln \frac{1 + z_t}{1 - z_t}$$

$$D_{t+1}(i) \propto D_t(i) \exp\left(-\alpha_t\, y^{(i)} h_t(x^{(i)})\right)$$

- Final classifier:

$$H_{\text{final}}(x) := \text{sign}\left(\sum_{t=1}^{T} \alpha_t\, h_t(x)\right)$$

Base learner in AdaBoost

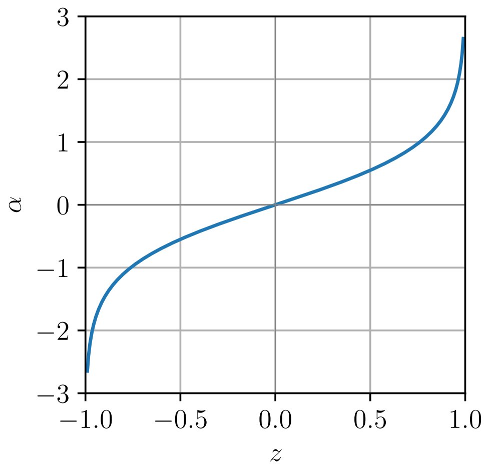
- Base learner: learning algorithm used inside boosting algorithm
  - (Also called "weak learner" in original theoretical context)
- In AdaBoost: in iteration t, base learner is provided training data $\mathcal{S}$ along with "example weights" D_t
  - Assume base learner accounts for example weights in selecting classifier
  - E.g., choose linear classifier based on weight vector w to (try to) minimize
$$\sum_{i=1}^{n} D_t(i)\, \mathrm{loss}\left(w^\top x^{(i)}, y^{(i)}\right)$$
  - E.g., use greedy algorithm to construct decision tree $h_{\mathcal{T}}$ to (try to) minimize
$$\sum_{i=1}^{n} D_t(i)\, \mathbb{I}\left(h_{\mathcal{T}}\left(x^{(i)}\right) \neq y^{(i)}\right)$$

AdaBoost example weights

- How example weights are updated after getting h_t from base learner:

D_t-weighted "correlation" between predictions of h_t and true labels

$$z_t := \sum_{i=1}^{n} D_t(i)\, y^{(i)} h_t\left(x^{(i)}\right)$$
$$\alpha_t := \frac{1}{2} \ln \frac{1 + z_t}{1 - z_t}$$
$$D_{t+1}(i) \propto D_t(i) \exp\left(-\alpha_t\, y^{(i)} h_t\left(x^{(i)}\right)\right)$$



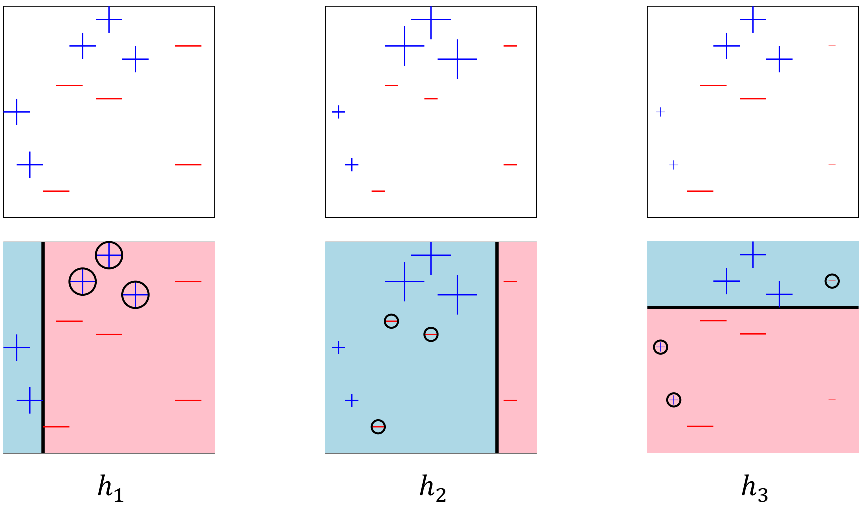
- Updated weights encourage base learner (in next iteration) to focus on training examples where h_t makes mistakes

Sample run of AdaBoost

- Base learner:
  - Choose feature $j \in \{1, \ldots, d\}$ and threshold $\theta \in \mathbb{R}$ such that "decision stump"

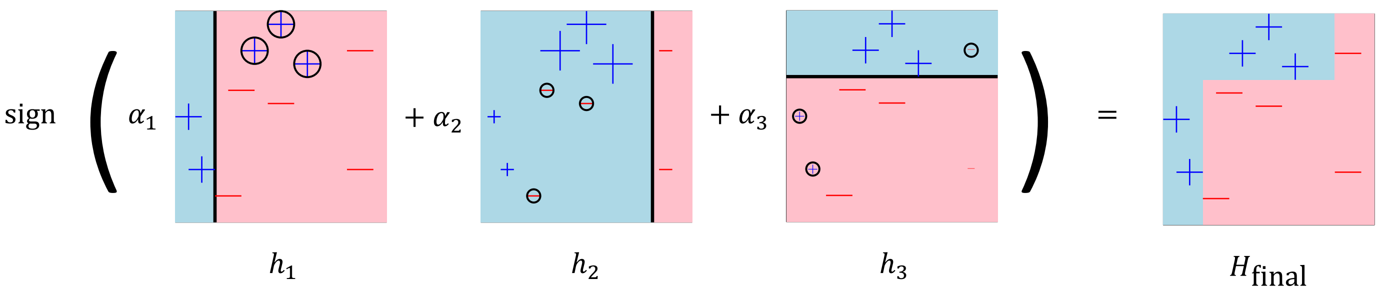
    $$x \mapsto \operatorname{sign}(x_j - \theta) \;\text{ or }\; x \mapsto \operatorname{sign}(-x_j - \theta)$$

    minimizes weighted training error rate



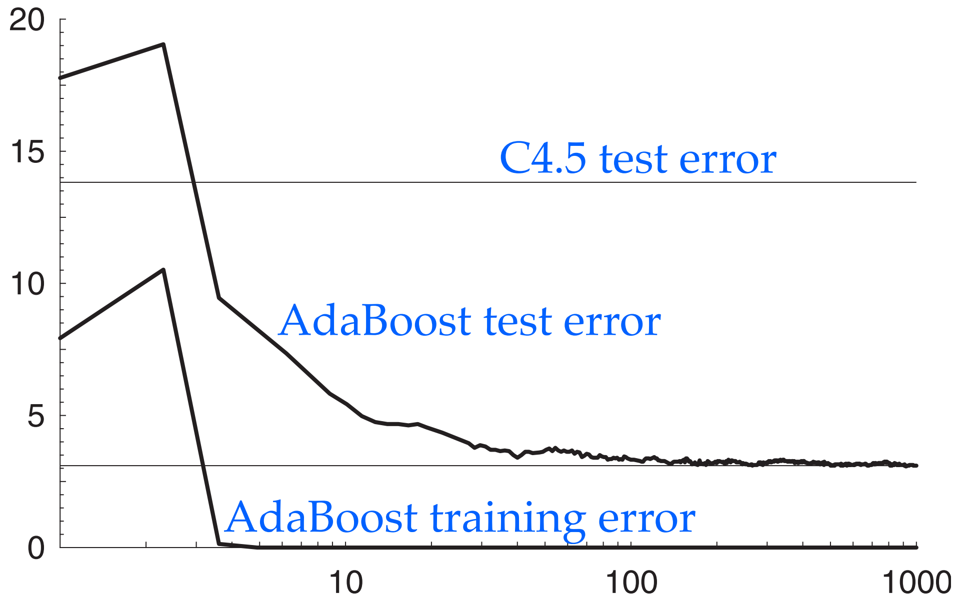
Final classifier from sample run of AdaBoost

- Final classifier:
  - Weighted majority vote of individual classifiers h_1, h_2, h_3
  - Classifier weight α_t based on D_t-weighted correlation z_t between h_t's predictions and labels



Some surprising behavior of AdaBoost (circa 1997)

- AdaBoost + "C4.5 tree learner" as base learner on "letters" dataset



AdaBoost margins [Schapire, Freund, Bartlett, Lee, 1997]

- Margin of H_{final} on example $(x, y) \in \mathcal{X} \times \{\pm 1\}$:

$$\frac{y \sum_{t=1}^{T} \alpha_t \, h_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \in [-1,1]$$

- AdaBoost tries to increase margin on training examples
- On "letters" dataset:

	T=5	T=100	T=1000
Training error rate	0.0%	0.0%	0.0%
Test error rate	8.4%	3.3%	3.1%
% margins ≤ 0.5	7.7%	0.0%	0.0%
Minimum margin	0.14	0.52	0.55

How is it possible to achieve large minimum margins?

- AdaBoost chooses distributions D_t over training examples in each iteration
- Assume base learner always choose h_t from (possibly huge) collection $\mathcal{H}$
- Suppose there is positive number γ such that, for any distribution D over training examples, it is always possible to find $h \in \mathcal{H}$ with

$$\sum_{i=1}^{n} D(i)\, y^{(i)} h\left(x^{(i)}\right) \geq \gamma$$

- Then, there must exist a distribution Q over $\mathcal{H}$ such that

$$\min_{i \in \{1,\ldots,n\}} \sum_{h \in \mathcal{H}} Q(h)\, y^{(i)} h\left(x^{(i)}\right) \geq \gamma$$

Key idea: AdaBoost efficiently solves a zero-sum game

- Zero-sum game between "min" (AdaBoost) and "max" (base learner)
  - First, "min" chooses distribution D over $\{1, \ldots, n\}$
  - Then, "max" chooses distribution Q over $\mathcal{H}$
  - Payoff (= how much "max" wins = how much "min" loses):

$$\mathbb{E}_{(i,h)\sim D\otimes Q}[M(i,h)]$$

  where

$$M(i,h) \coloneqq y^{(i)}\, h\left(x^{(i)}\right) \in \{-1,1\}$$

- Assumption is that

$$\min_{D} \max_{Q} \mathbb{E}_{(i,h)\sim D\otimes Q}[M(i,h)] \geq \gamma$$

- Von Neumann min-max theorem says this is equivalent to

$$\max_{Q} \min_{D} \mathbb{E}_{(i,h)\sim D\otimes Q}[M(i,h)] \geq \gamma$$

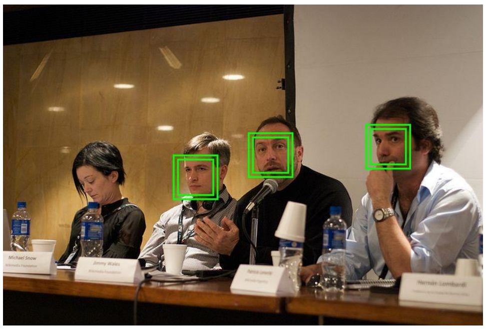
Always achieved by Q that puts all weight on single h

Always achieved by D that puts all weight on single i

Face detection with AdaBoost

Face detection

- Problem: given an image, locate all faces in it



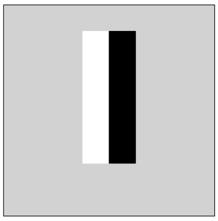
- As classification problem:
  - Divide image into many "patches" of varying sizes (e.g., 24x24, 48x48)
  - Predict whether a given patch x contains a face (binary classification)
- Main challenge: make this fast

Face detection using AdaBoost

- Major achievement by Viola & Jones (2001): Real-time face detector
- Regard image patch ($d \times d$ grayscale image) as vector in $[0,1]^{d^2}$
- Use AdaBoost with base learner that returns linear classifiers

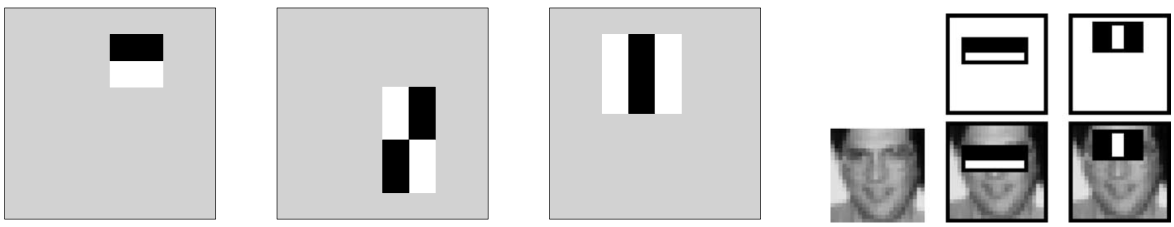
$$h(\vec{x}) = \mathrm{sign}(\langle \vec{w}, \vec{x} \rangle + b)$$

where $\vec{w}$ is specified by a simple pattern such that:



$\langle \vec{w}, \vec{x} \rangle =$ sum of pixel values in black box
$-$ sum of pixel values in white box

Other examples of Viola & Jones base learner classifiers



$\langle \vec{w}, \vec{x} \rangle =$ sum of pixel values in black box
$-$ sum of pixel values in white box

Viola & Jones integral image trick

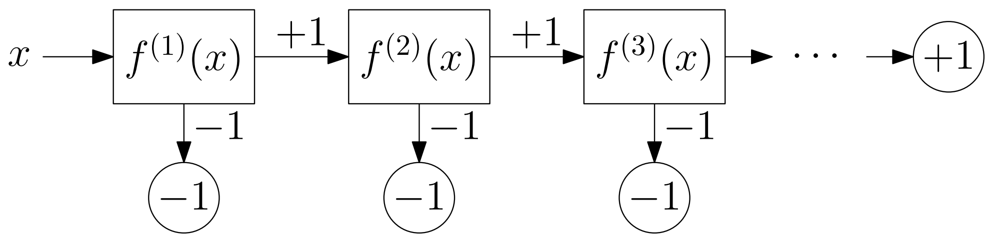
- Fast computation of $\langle \vec{w}, \vec{x} \rangle$:
  - For every image, compute $d \times d$ matrix s, where $s(r, c)$ = sum of pixel values from $(0,0)$ to (r, c)

$\langle \vec{w}, \vec{x} \rangle$ = sum of pixel values in black box
− sum of pixel values in white box

This only requires looking at a few entries of s

Viola & Jones cascade architecture

- Most patches do not contain a face
- Cascade classifier (a.k.a. decision list):
  - Each $f^{(\ell)}$ is based on classifier $F^{(\ell)}$ trained AdaBoost, but adjust "threshold" (inside $\text{sign}(\cdot)$) to minimize False Negative Rate (where $+1$ = "face")
  - Can afford to have $f^{(\ell)}$ at later stages be more "complex" because most patches don't make it to the later parts of the cascade



Example results from Viola & Jones face detector

Gradient boosting

Gradient descent for functions?

- Training data: $((x_i, y_i))_{i=1}^n$ from $\mathcal{X}\times\mathcal{Y}$
- Loss function: $\operatorname{loss}(s, y)$, assumed differentiable w.r.t. $s \in \mathbb{R}$
- Training objective: find $F: \mathcal{X} \to \mathbb{R}$ to minimize

$$J(F) \coloneqq \sum_{i=1}^{n} \operatorname{loss}(F(x_i), y_i)$$

- How about we just worry about predictions on training examples?

$$\vec{s} = (s_1, \dots, s_n) \in \mathbb{R}^n$$

$$\tilde{J}(\vec{s}) \coloneqq \sum_{i=1}^{n} \operatorname{loss}(s_i, y_i)$$

Gradient descent for training data predictions?

- Gradient of $\tilde{J}$:

$$\nabla\tilde{J}(\vec{s}) \coloneqq \left(\frac{\partial \operatorname{loss}(s, y_1)}{\partial s}(s_1), \dots, \frac{\partial \operatorname{loss}(s, y_n)}{\partial s}(s_n)\right)$$

- Can update $\vec{s} \in \mathbb{R}^n$ by subtracting small multiple of $\nabla\tilde{J}(\vec{s})$
- But this only gives predictions on training data! 😥

Functional gradient descent

- Work with functions defined on whole space $\mathcal{X}$, not just training data
- Find function $h: \mathcal{X} \to \mathbb{R}$ such that
$$h(x_i) \approx -\frac{\partial \text{loss}(s, y_i)}{\partial s}\left(F(x_i)\right)$$
for all $i \in \{1, \dots, n\}$ (perhaps just on average)
- Common approach: train regression tree $h_{\mathcal{T}}$ to (try to) minimize
$$\sum_{i=1}^{n}\left(h_{\mathcal{T}}(x_i) - \left(-\frac{\partial \text{loss}(s, y_i)}{\partial s}\left(F(x_i)\right)\right)\right)^2$$
- Then update F to $F + \eta\, h$ for some step size $\eta > 0$

Gradient boosting [Friedman, 1999; Mason, Baxter, Bartlett, Frean, 1999]

- Training data: $((x_i, y_i))_{i=1}^{n}$ from $\mathcal{X} \times \mathcal{Y}$
- Initial predictor: $F_0 :=$ (constant zero function)
- For $t = 1, \dots, T$:
  - Run "base learner" to get function $h_t: \mathcal{X} \to \mathbb{R}$ to try to minimize
$$\sum_{i=1}^{n}\left(h_t(x_i) - \left(-\frac{\partial \text{loss}(s, y_i)}{\partial s}\left(F_{t-1}(x_i)\right)\right)\right)^2$$
  - Update function (with step size $\eta_t > 0$):
$$F_t := F_{t-1} + \eta_t\, h_t$$
- Final predictor:
$$F_T = \eta_1\, h_1 + \cdots + \eta_T\, h_T$$

Example instantiation with squared loss function

- (Half) squared error:

$$\text{loss}(s, y) = \frac{1}{2}(s - y)^2$$
$$-\frac{\partial \text{loss}(s, y)}{\partial s}(s') = y - s'$$

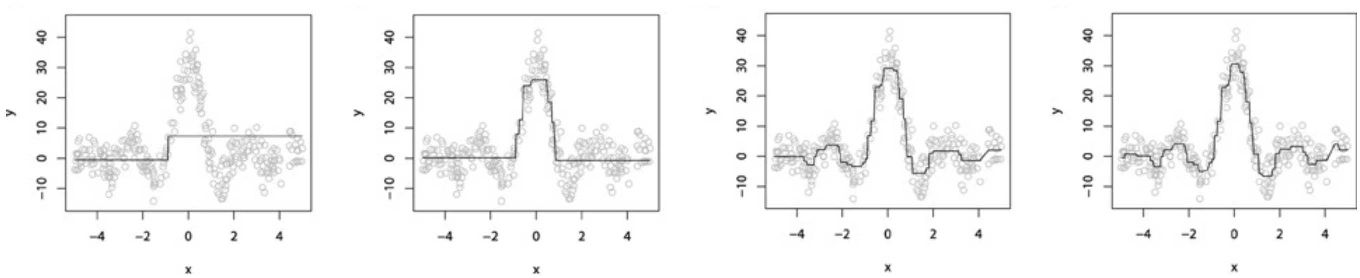
- "Base learner" goal is to find function $h_t: \mathcal{X} \to \mathbb{R}$ to (try to) minimize

$$\sum_{i=1}^{n} \left(h_t(x_i) - (y_i - F_{t-1}(x_i))\right)^2$$

- So want h_t to predict residuals $y_i - F_{t-1}(x_i)$

Synthetic example [Natekin & Knoll, 2013, tutorial on gradient boosting]

- Base learner: returns decision stumps h_t
- Fit to "sinc" data after $t \in \{1, 10, 50, 100\}$ iterations



Hopes and fears with gradient boosting

- Ideal case: "base learner" finds h_t such that

$$h_t(x) = -\frac{\partial \text{loss}(s, y)}{\partial s}\big(F_{t-1}(x)\big)$$

for all $x \in \mathcal{X}$, where y is correct label of x

  - Note: we don't have label y for x that's not in training data
  - We hope that because h_t fits well on training data, it also fits well elsewhere!

- Bad case: "base learner" finds h_t such that for all $i \in \{1, \dots, n\}$

$$h_t(x_i) = -\frac{\partial \text{loss}(s, y_i)}{\partial s}\big(F_{t-1}(x_i)\big)$$

and for all $x \notin \{x_1, \dots, x_n\}$,

$$h_t(x) = (\text{arbitrary random number})$$

Comparison to neural networks

- Two-layer neural network: for $\vec{x} \in \mathbb{R}^d$

$$F(\vec{x}) = \sum_{i=1}^{p} \alpha_i \, \sigma(\langle \vec{w}_i, \vec{x} \rangle)$$

- Ensemble predictor from boosting: for $x \in \mathcal{X}$

$$F(x) = \sum_{t=1}^{T} \eta_t \, h_t(x)$$

Other issues and variants

- Step size: common to set η_t as small number, though use of smaller η_t usually requires more iterations (and hence larger ensemble)
- AdaBoost: special case with $\text{loss}(s, y) = e^{-ys}$ for $y \in \{-1,1\}$, $h_t: \mathcal{X} \to \{-1,1\}$, and adaptively chosen η_t
- Stochastic Gradient Boosting: use random sample of training data in each iteration (akin to minibatch SGD)
- XGBoost: functional version of Newton's method with regression tree learner as base learner (+ many tricks for base learner)
- …