Boosting

COMS 4721 Spring 2022
Daniel Hsu

AdaBoost

Adaptive ensemble method

- Constituent predictors in "bagging" are essentially constructed in parallel
  - Computationally nice, but maybe a lot of unnecessary redundancy

Adaptive ensemble method

- Constituent predictors in "bagging" are essentially constructed in parallel
  - Computationally nice, but maybe a lot of unnecessary redundancy
- Can we do better by adaptively constructing the predictors in sequence?

Adaptive ensemble method

- Constituent predictors in "bagging" are essentially constructed in parallel
  - Computationally nice, but maybe a lot of unnecessary redundancy
- Can we do better by adaptively constructing the predictors in sequence?

AdaBoost algorithm
(Freund & Schapire, 1995)

Yoav Freund

Robert Schapire

Adaptive ensemble method

- Constituent predictors in "bagging" are essentially constructed in parallel
  - Computationally nice, but maybe a lot of unnecessary redundancy
- Can we do better by adaptively constructing the predictors in sequence?

AdaBoost algorithm
(Freund & Schapire, 1995)

Yoav Freund

Robert Schapire

Boosting: Convert a "weak learning algorithm" into a "strong learning algorithm"

AdaBoost (Adaptive Boosting)

Ensemble learning algorithm for binary classification data $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \{-1, 1\}$

AdaBoost

- For $t = 1, 2, \ldots, T$:
  - Construct probability distribution $(D_t(1), \ldots, D_t(n))$
    (importance weights on training examples)
  - Run "**weak learning algorithm**" on D_t-weighted training examples
    $\longrightarrow$ "**weak classifier**" $h_t \colon \mathbb{R}^d \to \{-1, 1\}$, with

$$D_t\text{-weighted training error rate} \quad \epsilon_t \;:=\; \sum_{i=1}^{n} D_t(i) \cdot \mathbb{1}\{h_t(\vec{x}_i) \neq y_i\}$$

- Return final (ensemble) classifier $f \colon \mathbb{R}^d \to \{-1, 1\}$ based on $h_1, \ldots, h_T$

AdaBoost (Adaptive Boosting)

Ensemble learning algorithm for binary classification data $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \{-1, 1\}$

AdaBoost

- For $t = 1, 2, \ldots, T$:
  - Construct probability distribution $(D_t(1), \ldots, D_t(n))$
    (importance weights on training examples)
  - Run "**weak learning algorithm**" on D_t-weighted training examples
    $\longrightarrow$ "**weak classifier**" $h_t \colon \mathbb{R}^d \to \{-1, 1\}$, with

$$D_t\text{-weighted training error rate} \quad \epsilon_t \;:=\; \sum_{i=1}^{n} D_t(i) \cdot \mathbb{1}\{h_t(\vec{x}_i) \neq y_i\}$$

- Return final (ensemble) classifier $f \colon \mathbb{R}^d \to \{-1, 1\}$ based on $h_1, \ldots, h_T$

Questions:

- How to construct importance weights D_t?
- How to construct final (ensemble) classifier f?

Constructing the importance weights and final classifier

- $D_1(i) := 1/n$ for all $i = 1, \ldots, n$

Constructing the importance weights and final classifier

- $D_1(i) := 1/n$ for all $i = 1, \ldots, n$
- Given D_t and h_t,

$$D_{t+1}(i) := \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } y_i = h_t(\vec{x}_i) \\ \exp(+\alpha_t) & \text{if } y_i \neq h_t(\vec{x}_i) \end{cases}$$
$$= \frac{D_t(i)}{Z_t} \times \exp(-\alpha_t \, y_i \, h_t(\vec{x}_i))$$

where Z_t is normalization factor that ensures D_{t+1} is a valid probability distribution,

Constructing the importance weights and final classifier

- $D_1(i) := 1/n$ for all $i = 1, \ldots, n$
- Given D_t and h_t,

$$D_{t+1}(i) := \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } y_i = h_t(\vec{x}_i) \\ \exp(+\alpha_t) & \text{if } y_i \neq h_t(\vec{x}_i) \end{cases}$$
$$= \frac{D_t(i)}{Z_t} \times \exp(-\alpha_t \, y_i \, h_t(\vec{x}_i))$$

where Z_t is normalization factor that ensures D_{t+1} is a valid probability distribution, and

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

($\alpha_t > 0$ whenever $\epsilon_t < 1/2$)

Constructing the importance weights and final classifier

- $D_1(i) := 1/n$ for all $i = 1, \ldots, n$
- Given D_t and h_t,

$$D_{t+1}(i) := \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } y_i = h_t(\vec{x}_i) \\ \exp(+\alpha_t) & \text{if } y_i \neq h_t(\vec{x}_i) \end{cases}$$
$$= \frac{D_t(i)}{Z_t} \times \exp(-\alpha_t \, y_i \, h_t(\vec{x}_i))$$

where Z_t is normalization factor that ensures D_{t+1} is a valid probability distribution, and

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

($\alpha_t > 0$ whenever $\epsilon_t < 1/2$)

- Final classifier:

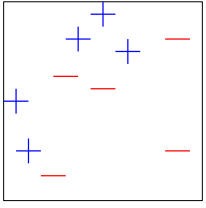
$$f(\vec{x}) \;:=\; \text{sign} \left(\sum_{t=1}^{T} \alpha_t \, h_t(\vec{x}) \right)$$

Example: AdaBoost with decision stumps

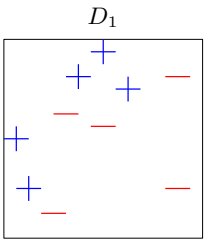
(Example from Figures 1.1 and 1.2 of Schapire & Freund text)

- Use decision stump learning algorithm as "weak learning algorithm"
- Each h_t has the form

$$h_t(\vec{x}) = \begin{cases} +1 & \text{if } x_{i_t} > \theta_t \\ -1 & \text{if } x_i \leq \theta \end{cases} \quad \text{or} \quad h_t(\vec{x}) = \begin{cases} -1 & \text{if } x_{i_t} > \theta_t \\ +1 & \text{if } x_i \leq \theta \end{cases}$$

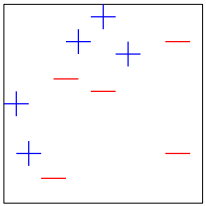


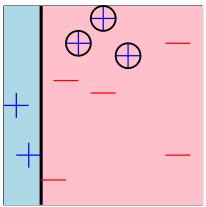
Execution of AdaBoost



Execution of AdaBoost

D_1

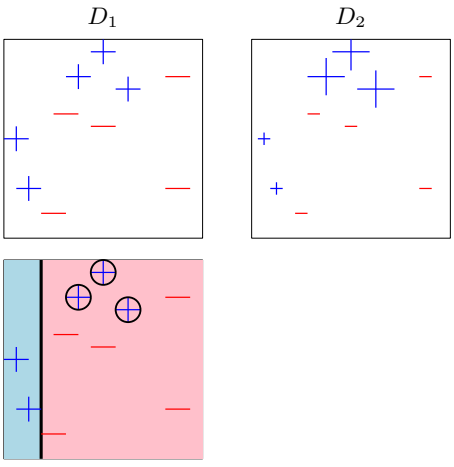




h_1

$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$

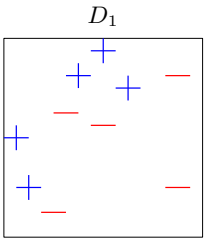
Execution of AdaBoost

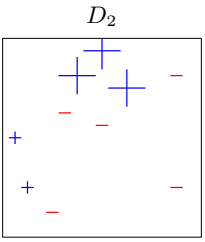


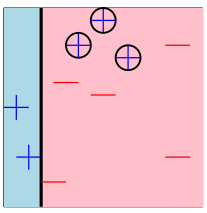
h_1

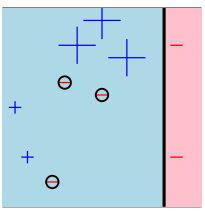
$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$

Execution of AdaBoost









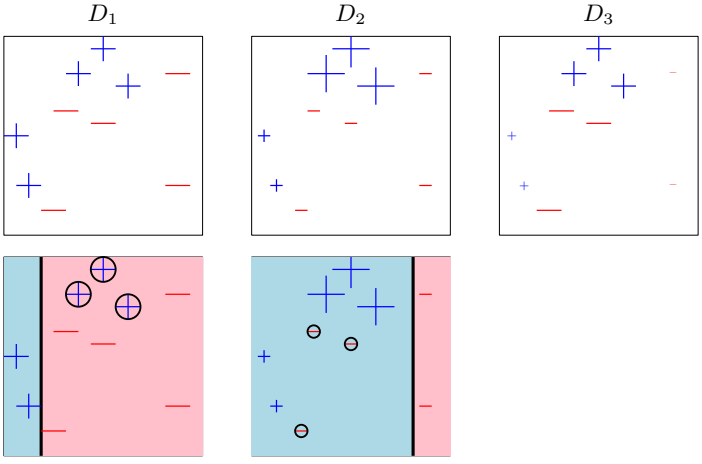
h_1

$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$

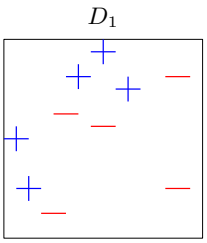
h_2

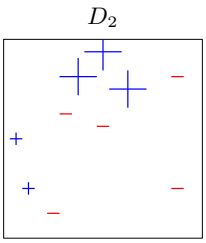
$\epsilon_2 = 0.21,\ \alpha_2 = 0.65$

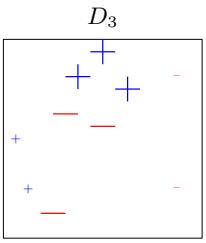
Execution of AdaBoost

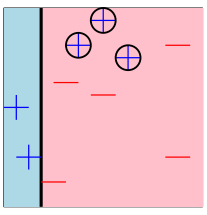


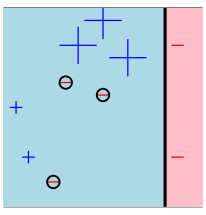
Execution of AdaBoost

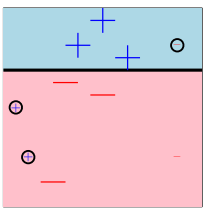










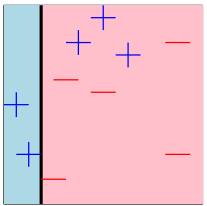


h_1
$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$

h_2
$\epsilon_2 = 0.21,\ \alpha_2 = 0.65$

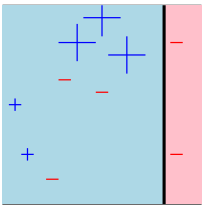
h_3
$\epsilon_3 = 0.14,\ \alpha_3 = 0.92$

Final classifier from AdaBoost



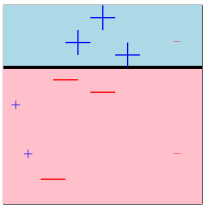
h_1

$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$



h_2

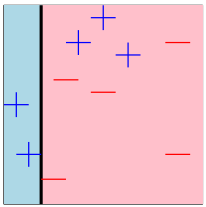
$\epsilon_2 = 0.21,\ \alpha_2 = 0.65$



h_3

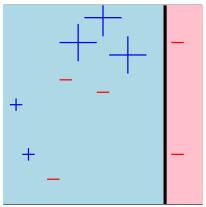
$\epsilon_3 = 0.14,\ \alpha_3 = 0.92$

Final classifier from AdaBoost



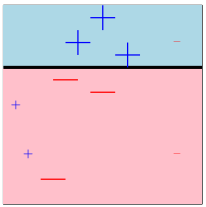
h_1

$\epsilon_1 = 0.30,\ \alpha_1 = 0.42$



h_2

$\epsilon_2 = 0.21,\ \alpha_2 = 0.65$



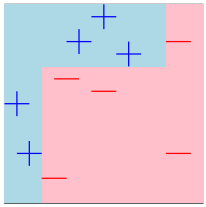
h_3

$\epsilon_3 = 0.14,\ \alpha_3 = 0.92$

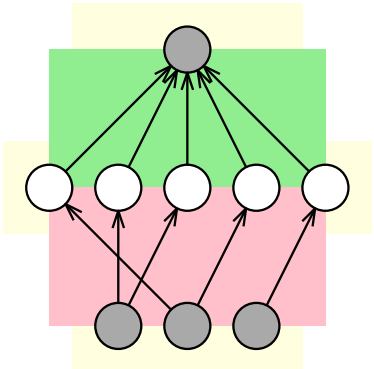
Final classifier

$f(x) = \text{sign}(0.42\, h_1(x) + 0.65\, h_2(x) + 0.92\, h_3(x))$

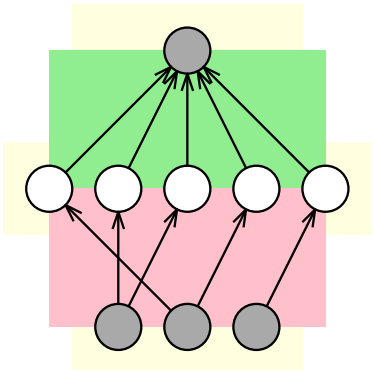
(Zero training error rate!)



Structure of final classifier

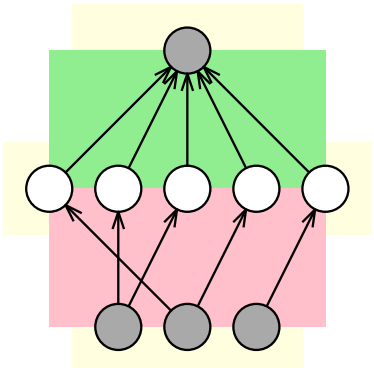


Structure of final classifier



- **Bottom layer:** input $\vec{x} = (x_1, \ldots, x_d)$ to classifier

Structure of final classifier



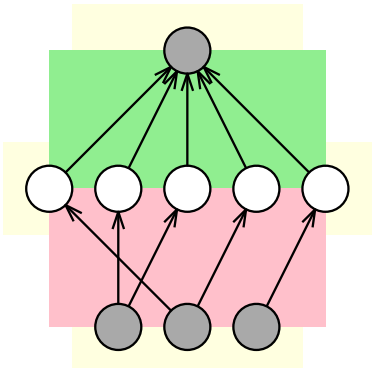
- **Middle layer:** predictions of "weak classifiers" h_t's

  e.g., $h_t(\vec{x}) = \text{sign}(\vec{x} \cdot \vec{w}_t - \theta_t)$

  (For decision stumps, $\vec{w}_t$ has exactly one non-zero entry)

- **Bottom layer:** input $\vec{x} = (x_1, \ldots, x_d)$ to classifier

Structure of final classifier



- **Top layer:** output of classifier
  Output is weighted majority vote of "weak classifiers"

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

- **Middle layer:** predictions of "weak classifiers" h_t's

$$\text{e.g.,} \quad h_t(\vec{x}) = \text{sign}(\vec{x} \cdot \vec{w}_t - \theta_t)$$

  (For decision stumps, $\vec{w}_t$ has exactly one non-zero entry)

- **Bottom layer:** input $\vec{x} = (x_1, \ldots, x_d)$ to classifier

Training error rate of final classifier

Write D_t-weighted training error rate as $\epsilon_t = \frac{1}{2} - \gamma_t$

Theorem (Freund & Schapire, 1995). Training error rate AdaBoost's final classifier is

$$\leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

Training error rate of final classifier

Write D_t-weighted training error rate as $\epsilon_t = \frac{1}{2} - \gamma_t$

Theorem (Freund & Schapire, 1995). Training error rate AdaBoost's final classifier is

$$\leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

- Adaptivity: Don't care about how large any individual γ_t is; only care about aggregate $\sum_t \gamma_t^2$

Training error rate of final classifier

Write D_t-weighted training error rate as $\epsilon_t = \frac{1}{2} - \gamma_t$

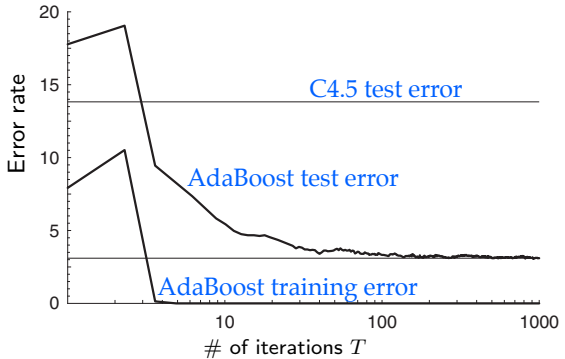
Theorem (Freund & Schapire, 1995). Training error rate AdaBoost's final classifier is

$$\leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

- Adaptivity: Don't care about how large any individual γ_t is; only care about aggregate $\sum_t \gamma_t^2$
- # iterations T = # constituent classifiers in final (ensemble) classifier
  (More iterations $\longrightarrow$ more complex classifier?)

Typical run of AdaBoost

(Figure 1.7 from Schapire & Freund text)
AdaBoost+C4.5 on "letters" dataset: (C4.5 = greedy training heuristic for decision trees)



(# nodes across all decision trees in final f is $>2 \times 10^6$)

Training error rate is zero after just five rounds,
but **test error rate continues to decrease, even up to** 1000 **rounds**!

Boosting the margin

Final (ensemble) classifier from AdaBoost:

$$f(\vec{x}) \;=\; \operatorname{sign}\Bigg(\underbrace{\frac{\sum_{t=1}^{T} \alpha_t \, h_t(\vec{x})}{\sum_{t=1}^{T} |\alpha_t|}}_{g(\vec{x})} \Bigg)$$

Call $y \cdot g(\vec{x}) \in [-1, +1]$ the L_1-**margin** achieved on example $(\vec{x}, y) \in \mathbb{R}^d \times \{-1, 1\}$

Boosting the margin

Final (ensemble) classifier from AdaBoost:

$$f(\vec{x}) \;=\; \operatorname{sign}\Bigg(\underbrace{\frac{\sum_{t=1}^{T}\alpha_t\, h_t(\vec{x})}{\sum_{t=1}^{T}|\alpha_t|}}_{g(\vec{x})}\Bigg)$$

Call $y \cdot g(\vec{x}) \in [-1,+1]$ the L_1-**margin** achieved on example $(\vec{x}, y) \in \mathbb{R}^d \times \{-1, 1\}$

New theory [Schapire, Freund, Bartlett, and Lee, 1998]:

- **Larger L_1-margins $\Rightarrow$ "simpler" classifier vis-à-vis "Occam's razor", independent of T**
- AdaBoost tends to increase L_1-margins on training examples

Boosting the margin

Final (ensemble) classifier from AdaBoost:

$$f(\vec{x}) \;=\; \operatorname{sign}\Bigg(\underbrace{\frac{\sum_{t=1}^{T}\alpha_t\, h_t(\vec{x})}{\sum_{t=1}^{T}|\alpha_t|}}_{g(\vec{x})}\Bigg)$$

Call $y \cdot g(\vec{x}) \in [-1,+1]$ the L_1-**margin** achieved on example $(\vec{x}, y) \in \mathbb{R}^d \times \{-1, 1\}$

New theory [Schapire, Freund, Bartlett, and Lee, 1998]:

- **Larger L_1-margins ⇒ "simpler" classifier vis-à-vis "Occam's razor"**, **independent of T**
- AdaBoost tends to increase L_1-margins 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%
% L_1-margins ≤ 0.5	7.7%	0.0%	0.0%
min. L_1-margin	0.14	0.52	0.55

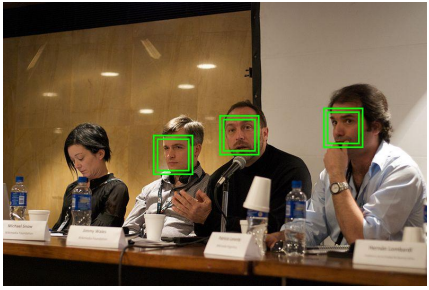
Recap

- AdaBoost sequentially + adaptively constructs classifiers for an ensemble
- Subjectivity of Occam's razor:
  - Measure complexity by size of ensemble (akin to number of features)?
  - Or measure simplicity by size of L_1-margins?

Application: Face Detection

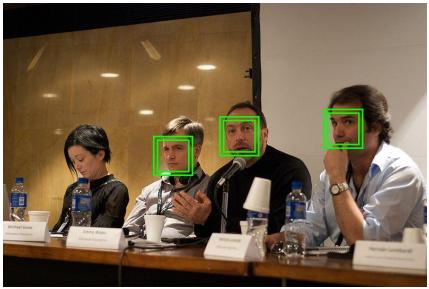
Application: Face detection

Problem: Given an image, locate all of the faces



Application: Face detection

Problem: Given an image, locate all of the faces

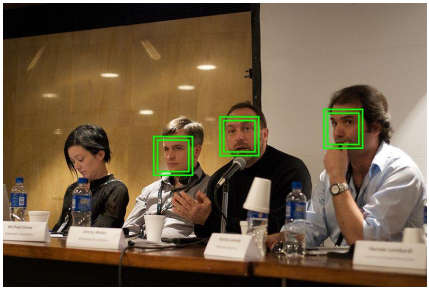


As a classification problem:

- Divide up images into patches (at varying scales, e.g., 24×24, 48×48)
- Make "face" vs "not face" prediction on each patch

Application: Face detection

Problem: Given an image, locate all of the faces



As a classification problem:

- Divide up images into patches (at varying scales, e.g., 24×24, 48×48)
- Make "face" vs "not face" prediction on each patch

Question: How to make something that works in real-time?

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.

Face detectors via AdaBoost

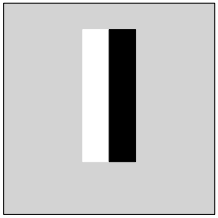
Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \mathrm{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

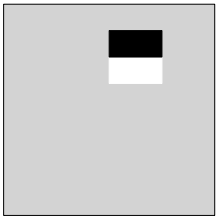


$\vec{w} \cdot \vec{x}$ = average pixel value in black box
− average pixel value in white box

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

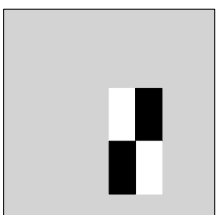


$\vec{w} \cdot \vec{x}$ = average pixel value in black box
− average pixel value in white box

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

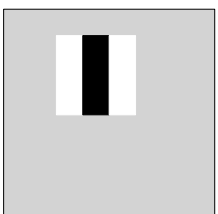


$\vec{w} \cdot \vec{x}$ = average pixel value in black box
− average pixel value in white box

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

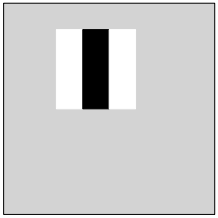


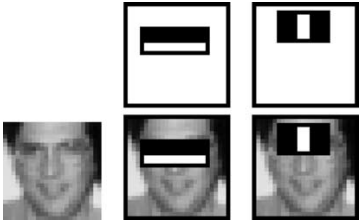
$\vec{w} \cdot \vec{x}$ = average pixel value in black box
− average pixel value in white box

Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

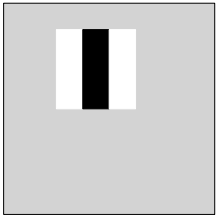


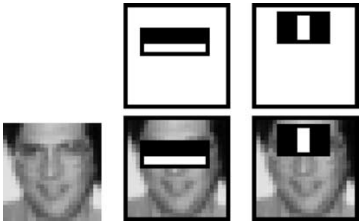


Face detectors via AdaBoost

Real-time face detector (Viola & Jones, 2001), major achievement in computer vision

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\vec{x} \in [0,1]^{d^2}$.
- Use "weak learning algorithm" that returns linear classifiers $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} - \theta)$, where each $\vec{w}$ has a very particular form:

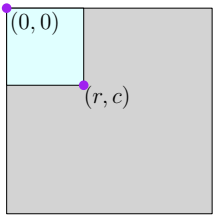




- AdaBoost combines many simple weak classifiers of this form
  - Extremely fast to evaluate via pre-computation ("integral image" trick)

Viola & Jones “integral image” trick

“Integral image” trick:



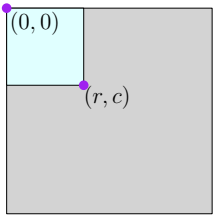
For given image, pre-compute

$$s(r, c) = \text{sum of pixel values in rectangle from } (0, 0) \text{ to } (r, c)$$

Can be done in a single pass through the image (i.e., linear time)

Viola & Jones "integral image" trick

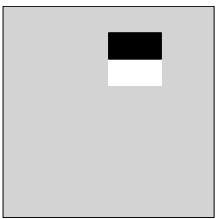
"Integral image" trick:



For given image, pre-compute

$$s(r,c) = \text{sum of pixel values in rectangle from } (0,0) \text{ to } (r,c)$$

Can be done in a single pass through the image (i.e., linear time)



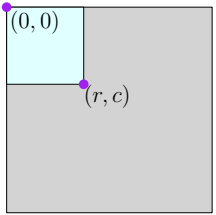
To compute inner product

$$\vec{w} \cdot \vec{x} = \text{average pixel value in black box} - \text{average pixel value in white box}$$

just need to add and subtract a few $s(r,c)$ values

Viola & Jones "integral image" trick

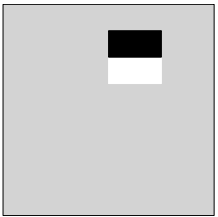
"Integral image" trick:



For given image, pre-compute

$$s(r,c) = \text{sum of pixel values in rectangle from } (0,0) \text{ to } (r,c)$$

Can be done in a single pass through the image (i.e., linear time)



To compute inner product

$$\vec{w} \cdot \vec{x} = \text{average pixel value in black box} - \text{average pixel value in white box}$$

just need to add and subtract a few $s(r,c)$ values

$\Rightarrow$ Evaluating weak classifiers is extremely fast

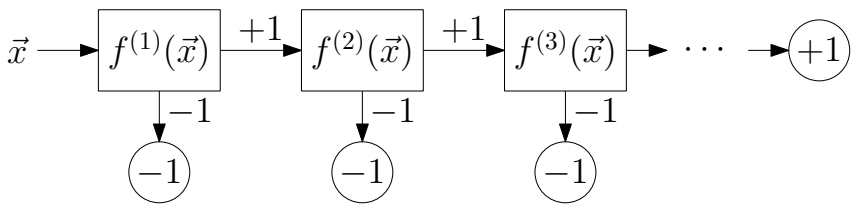
Viola & Jones cascade architecture

Observation: Most patches don't contain a face

Viola & Jones cascade architecture

Observation: Most patches don't contain a face

Idea: Train several classifiers (each using AdaBoost); arrange in a **decision list** (a.k.a. cascade)

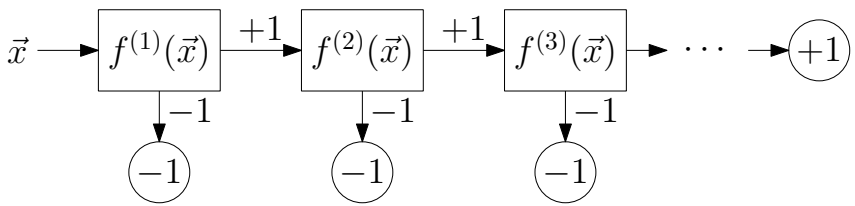


- Each $f^{(\ell)}$ is trained using AdaBoost; but modify (e.g., by adjusting threshold before sign) to *minimize false negative rate*
- Can make $f^{(\ell)}$ in later stages more "complex" (i.e., larger T) than in earlier stages, since most examples don't make it to the end

Viola & Jones cascade architecture

Observation: Most patches don't contain a face

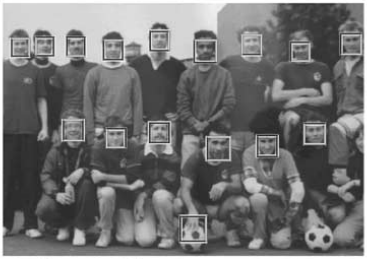
Idea: Train several classifiers (each using AdaBoost); arrange in a **decision list** (a.k.a. cascade)



- Each $f^{(\ell)}$ is trained using AdaBoost; but modify (e.g., by adjusting threshold before sign) to *minimize false negative rate*
- Can make $f^{(\ell)}$ in later stages more "complex" (i.e., larger T) than in earlier stages, since most examples don't make it to the end

$\Rightarrow$ (Cascade) classifier evaluation is extremely fast

Viola & Jones detector: example results



Recap

- Viola-Jones face detector: Takes problem-specific requirements into account for
  - structuring the final classifier (cascade),
  - setting up the learning objective (focus on FNR),
  - using the learning algorithm (AdaBoost with particular weak learner),
  - etc.

  Really require domain-knowledge to make these choices!