

Machine Learning & Computer Vision

MAM4 2018 - 2019

*Learning by classifier
combination: boosting*

Cours n°1

Frederic Precioso

Binary classification

- In this course we will focus on Binary Classification: separate the data between relevant and irrelevant.
- More specifically, the objective is to detect regions which correspond to a certain category among data.
- This defines a binary classification: regions corresponding **vs** regions not corresponding to the target category.

Classification

How to find the boundary between classes (or between categories) :

- Rose (flower)
- Not rose (other)



Observations:

- Easy to produce decision rules often right
 - If the histogram contains blue, then predict other
 - If the histogram contains pink, then predict rose
- Hard to find decision rules right all the time
- We focus here on a solution among many others: ***boosting***

Ideas of boosting: Football Bets



If Mbappé and Dembélé are happy together,
French Football team wins.

If Lemar is not injured, French
Football team wins.



If French
France



If Pogba is happy,
French Football team wins.

How to win?

- Ask to professional gamblers
- Lets assume:
 - That professional gamblers can provide one single decision rule simple and relevant
 - But that face to several games, they can always provide decision rules a little bit better than random
- Can we become rich?

Idea

- Ask **heuristics** to the *expert*
 - Gather a set of cases for which these **heuristics** fail (**difficult cases**)
 - Ask again the *expert* to provide **heuristics for the difficult cases**
 - And so one...
-
- **Combine** these heuristics
 - *expert* stands for **weak learner**

Questions

- How to choose games (i.e. learning examples) at each step?
 - Focus on games (examples) the most “difficult” (the ones on which previous heuristics are the less relevant)
- How to merge heuristics (decision rules) into one single decision rule?
 - Take a weighted vote of all decision rules

Boosting

- **boosting** = general method to convert several poor decision rules into one very powerful decision rule
- More precisely:
 - Let have a weak learner which can always provide a decision rule with an error $\leq \frac{1}{2} - \gamma$
 - A boosting algorithm can build (theoretically) a global decision rule with an error rate $\leq \varepsilon$

Probabilistic boosting

3 main ideas to generalize towards *probabilistic boosting*:

1. A set of specialized experts and ask them to vote to take a decision.
2. Adaptive weighting of votes by multiplicative update.
3. Modifying example distribution to train each expert, increasing the weights iteratively of examples misclassified at previous iteration.

Boosting: Basics

Parameter: weak learners h

Input: A set S

Initialization: All examples have same weights

For $t = 1, T$ **do**

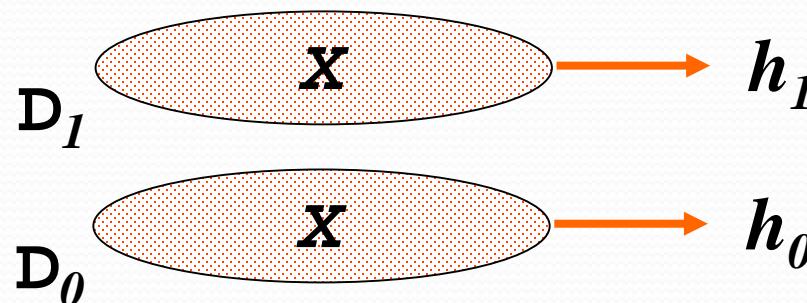
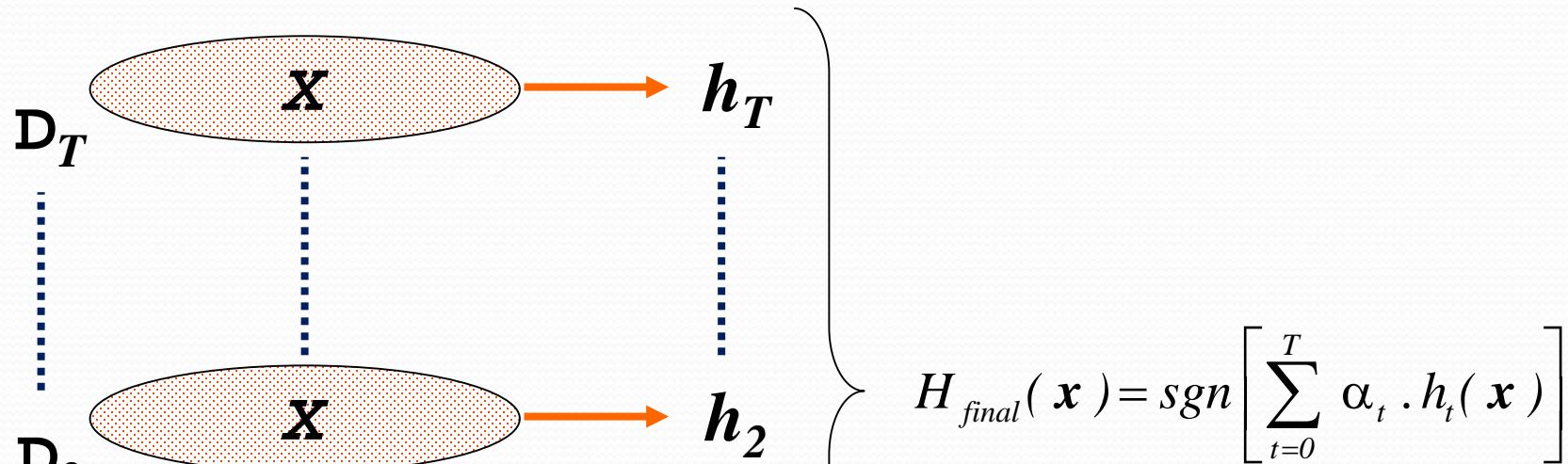
use h with S and current distribution

increase weight of misclassified examples at
current iteration

done

Output H : a majority weighted vote of all the h

Boosting: Basics



- How to update D_t to D_{t+1} ?
- How to compute the weight α_t ?
- What error is minimized ?

Probabilistic boosting: AdaBoost

The standard algorithm is **AdaBoost** (*Adaptive Boosting*).

The main idea is to define at each iteration $1 < t < T$, a new distribution of *a priori probabilities* on training samples with respect to classification result at previous iteration.

The weight at iteration t of an example (x_i, y_i) of index i is denoted $D_t(i)$.

Initially, all the examples get a similar weight, at each iteration, weights of misclassified examples are increased

Thus the learner **focuses** on **difficult examples** of training set

Boosting: the algorithm

- A training set: $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- $y_i \in \{-1, +1\}$ label (annotation) of example $x_i \in S$
- For $t = 0, \dots, T$:
 - Build the distribution D_t on $\{1, \dots, m\}$
 - Find the weak decision (“heuristic”)
 - $h_t : S \rightarrow \{-1, +1\}$
 - with a small error ε_t on D_t :

$$\varepsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$$

- Find the final decision h_{final}

The AdaBoost Algorithm

What **goal** the AdaBoost wants to reach?

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

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

The AdaBoost Algorithm

What **goal** the AdaBo

They are goal dependent.

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier:

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$

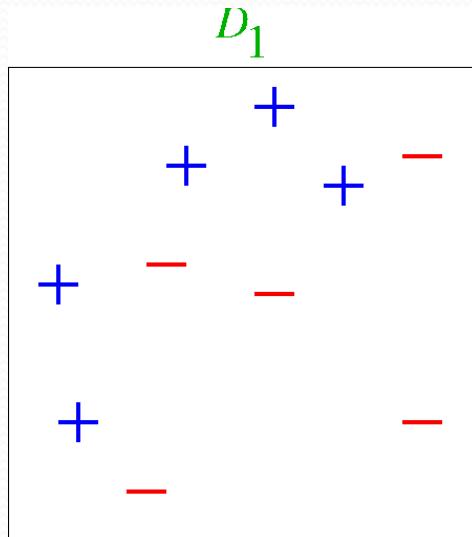
- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, Z_t \text{ is for normalization}$$

Output final classifier:

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

Example “Toy”

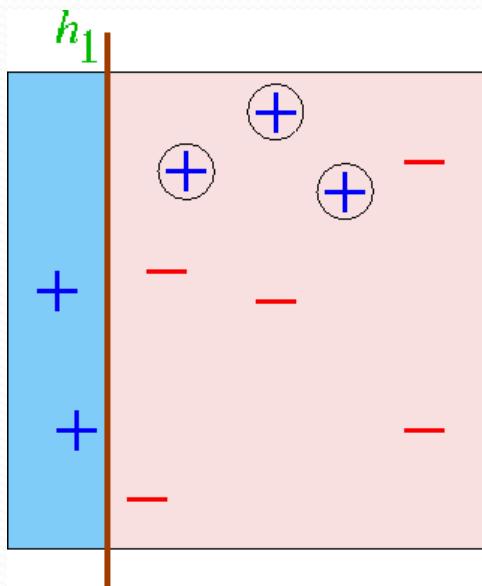


Very popular «Toy» example, 2D points in domain D_1 , to illustrate boosting schema.

(these points can be seen as vectors $\in \mathbb{R}^2$).

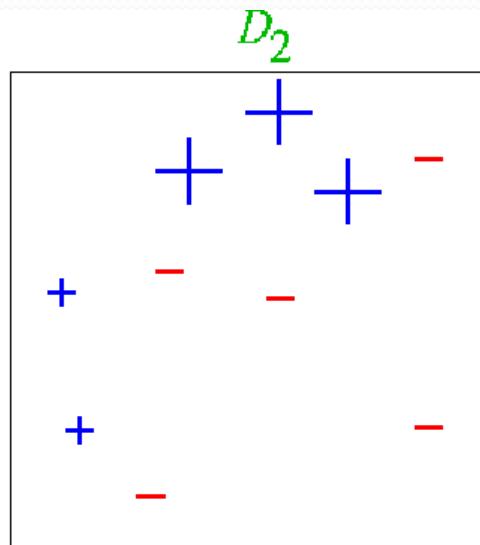
D1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Step 1



$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$



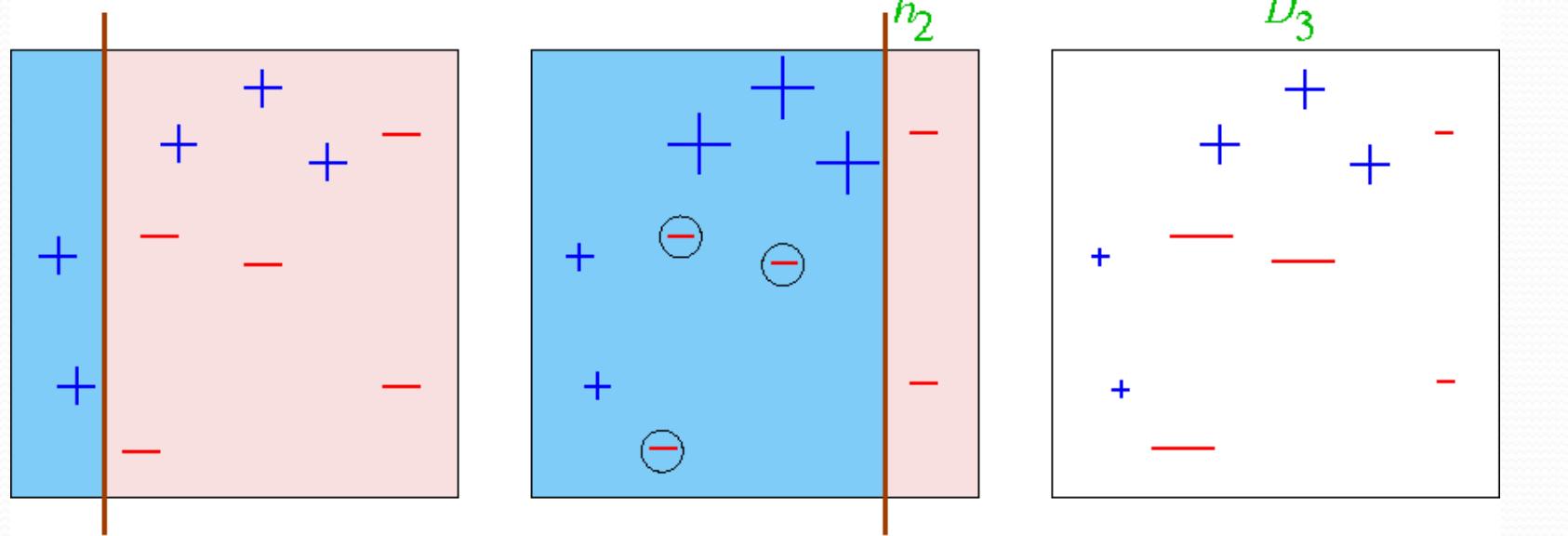
$$\begin{aligned} \alpha_1 &= \frac{1}{2} \ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \frac{1}{2} \ln\left(\frac{1-0.3}{0.3}\right) \\ &= \frac{1}{2} \ln\left(\frac{0.7}{0.3}\right) = \ln\left(\sqrt{\frac{7}{3}}\right) \end{aligned}$$

$$D'_1 = \begin{cases} 0.1 e^{-\alpha_1} = 0.1 \sqrt{\frac{3}{7}} = 0.065 \\ 0.1 e^{\alpha_1} = 0.1 \sqrt{\frac{7}{3}} = 0.152 \end{cases}$$

$$Z_1 = 3 \times 0.152 + 7 \times 0.065 = 0.911$$

$D_2 = D'_1 / Z_1$	0.167	0.167	0.167	0.071	0.071	0.071	0.071	0.071	0.071	0.071
--------------------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

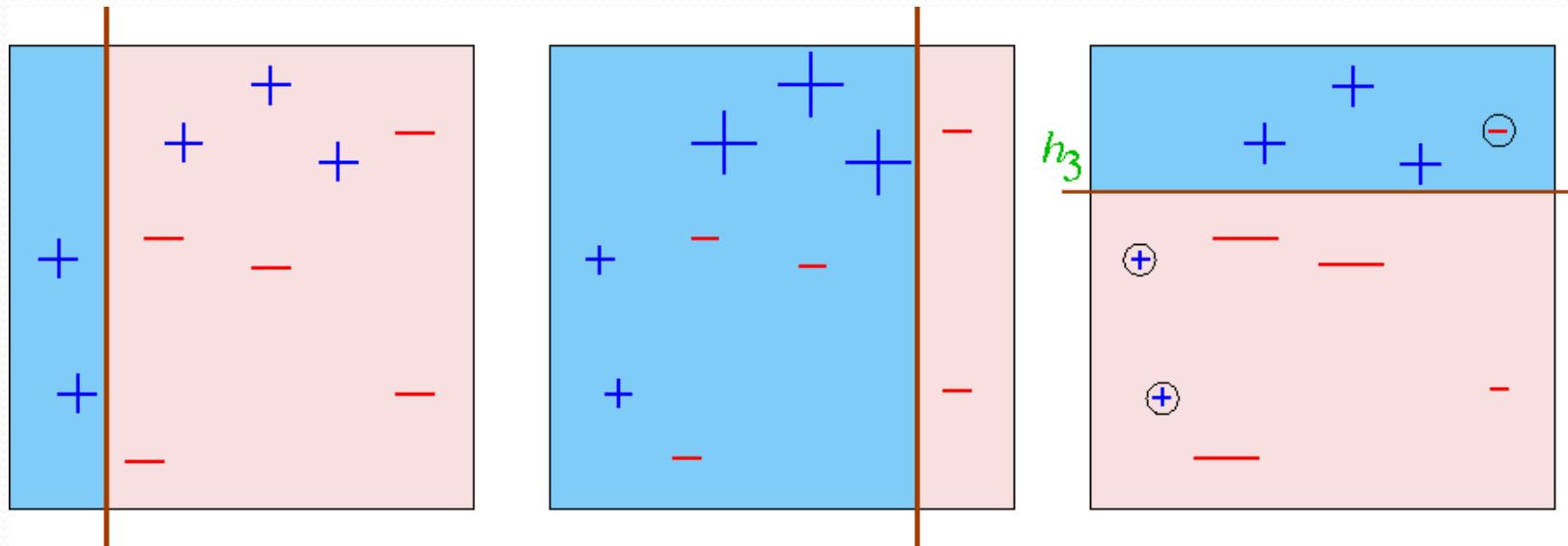
Step 2



$$D'_2 = \begin{cases} 0.17 e^{-\alpha_2} = 0.0876 \\ 0.07 e^{-\alpha_2} = 0.036 \\ 0.07 e^{\alpha_2} = 0.1357 \end{cases} \quad \begin{matrix} \epsilon_2 = 0.21 \\ \alpha_2 = 0.65 \end{matrix} \quad Z_2 = 3 \times 0.0876 + 4 \times 0.036 + 3 \times 0.1357 = 0.814$$

$D_3 = D'_2 / Z_2$	0.107	0.107	0.107	0.044	0.044	0.044	0.044	0.166	0.166	0.166
--------------------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

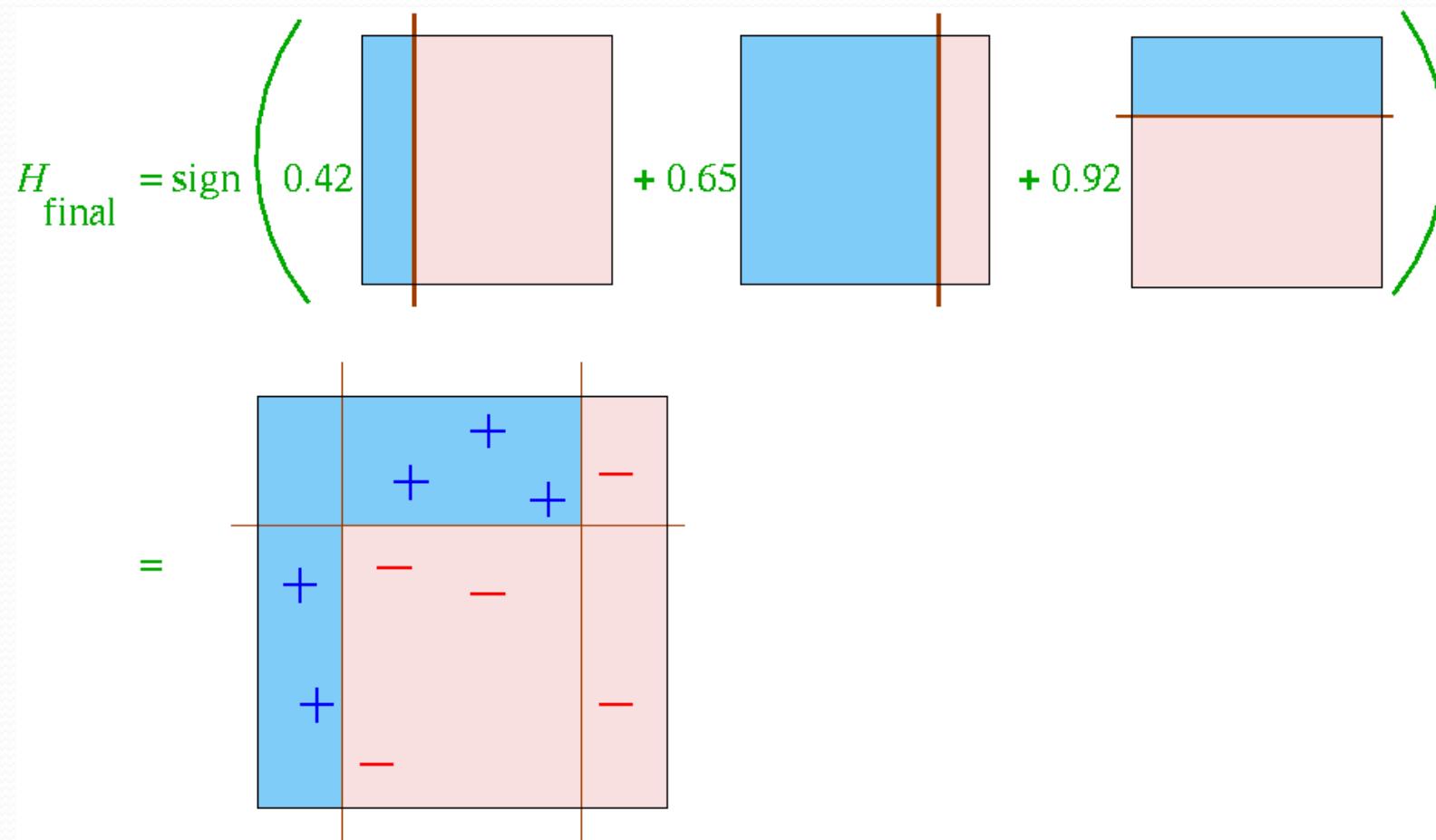
Step 3



$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Final decision



Boosting: global classifier

$$H_{\text{final}}(x) = \text{sgn} \left(\sum_t \alpha_t h_t(x) \right)$$

The boosting algorithm builds the finale decision as a summation on a function basis $\{h_t\}$.

This is a classic process of machine learning algorithms (as SVM, bayesian classifier, neural networks, etc.).

AdaBoost Hinge Loss

Lets write the error ε_t of h_t as: $1/2 - \gamma_t$, where γ_t measure the improvement brought by h_t regarding the basic error of $1/2$.

Freund and Schapire, have shown that the hinge loss (the amount of misclassified data over training set S) of finale decision H is bounded by:

$$\text{Error}_{\text{Hinge_Loss}}(H_{\text{final}}(x)) \leq \exp \left(-2 \sum_t \gamma_t^2 \right)$$

Hence, if each weak learner is slightly better than random ($\gamma_t > 0$) then the hinge loss decrease exponentially with t .

Proof

Let $H_{FINAL}(x) = f(x) = \sum_t \alpha_t h_t(x)$

$$\begin{aligned} \text{We have: } D_{Final}(x_i, y_i) &= \frac{D_t}{Z_t} \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i)) = \frac{D_{t-1}}{Z_{t-1}} \exp(-y_i \alpha_{t-1} \cdot h_{t-1}(x_i)) \frac{\exp(-y_i \alpha_t \cdot h_t(x_i))}{Z_t} \\ &= \frac{1}{m} \frac{\exp\left(-y_i \sum_t \alpha_t \cdot h_t(x_i)\right)}{\prod_t Z_t} = \frac{1}{m} \frac{\exp(-y_i f(x_i))}{\prod_t Z_t} \end{aligned}$$

We can prove that the hinge loss (i.e. error when counting 1 for every wrong result) is less than:

$$\prod_t Z_t$$

$$\begin{aligned} \text{Error}_{Hinge_Loss}(H(x)) &= \frac{1}{m} \sum_i \mathbf{1}_{H(x_i) \neq y_i} = \frac{1}{m} \sum_i \mathbf{1}_{y_i f(x_i) < 0} \\ &\leq \frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \sum_i D_{final}(i) \prod_t Z_t = \prod_t Z_t \end{aligned}$$

Proof

And

$$\begin{aligned} Z_t &= \sum_i D_t(i) \exp(-\alpha_t y_i f(x_i)) = \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \mathcal{E}_t e^{\alpha_t} + (1 - \mathcal{E}_t) e^{-\alpha_t} = 2 \sqrt{\mathcal{E}_t (1 - \mathcal{E}_t)} \end{aligned}$$

Then the **Hinge Loss** is bounded by:

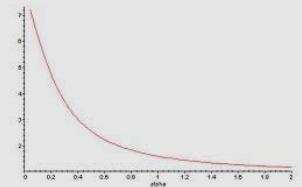
$$\prod_t Z_t = \prod_t [2 \sqrt{\varepsilon_t (1 - \varepsilon_t)}] = \prod_t \sqrt{1 - 4 \gamma_t^2} \leq \exp \left(-2 \sum_t \gamma_t^2 \right)$$

The error of learning decreases exponentially with t

Error of generalizaton

Error of generalization of H can be bounded by:

$$E_{\text{Real}}(H_T) = E_{\text{HingeLoss}}(H_T) + O\left(\sqrt{\frac{T \cdot d}{m}}\right)$$



where T is the number of boosting iterations, m the number of examples, and d a dimension (VC-dimension) of H_T space (« weaks learner complexity »)

When T becomes large, the 2nd term becomes large too, then one can think that the learner would **overfit**
⇒ the solution = over-weighting of ambiguous examples

Advantages of AdaBoost

- (very) fast
- simple + easy to code
- One unique parameter to set: the number of boosting iterations (T)
- Can be used with any machine learning algorithm
- No-overfitting (margin theory)
- Is adapted to multi-class problems where $y_i \in \{1, \dots, c\}$ and to multi-label problems
- Allow to find out outliers

Drawbacks of AdaBoost

- Performances depend on training data & weak learners
- AdaBoost can fail if
 - The weak learner is too complex
 - Low margins → overfitting
 - The weak learner is too weak
 - $\gamma_t \rightarrow 0$ too fast → underfitting
- AdaBoost seems specifically sensitive to noise

Applications

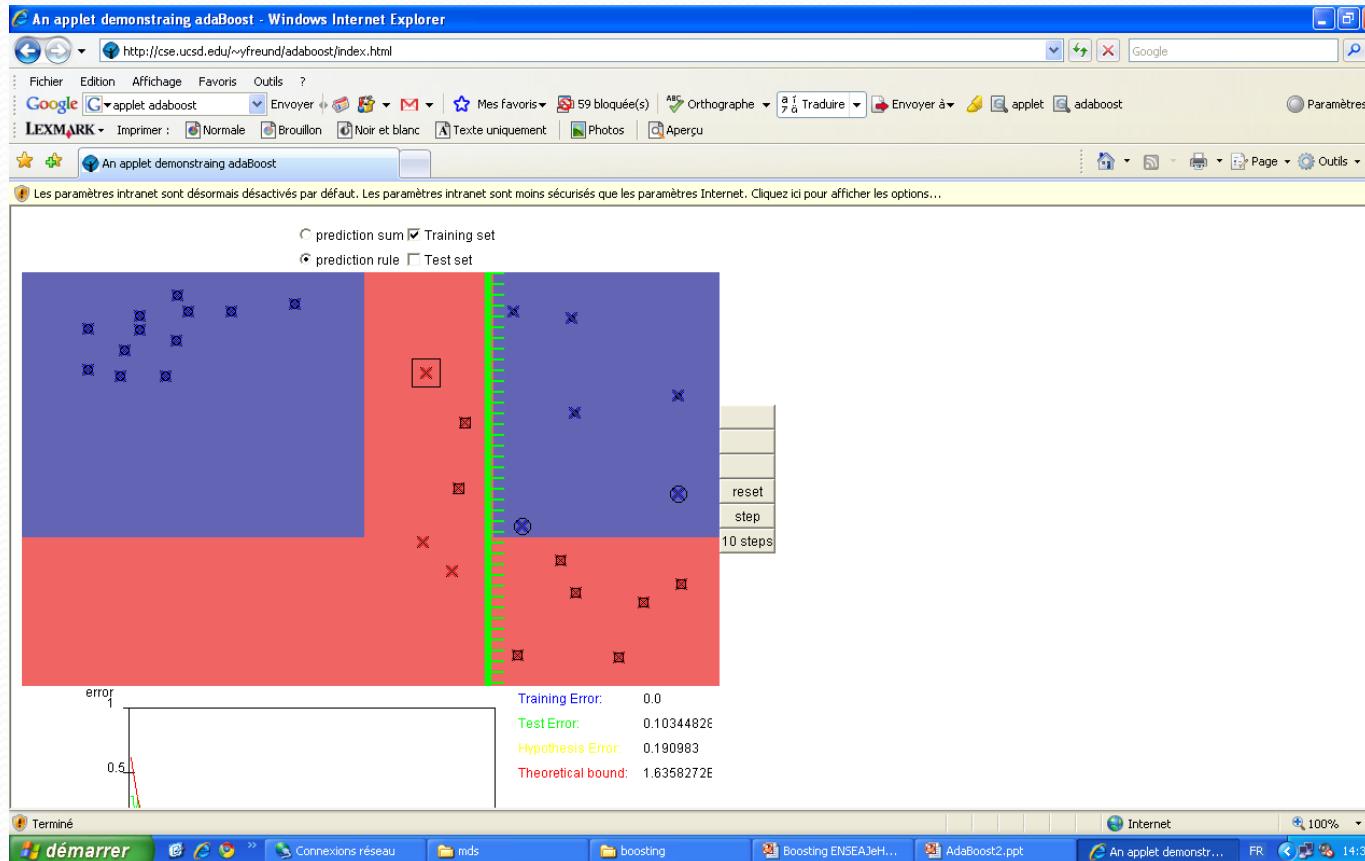
- Detection, tracking and face recognition
- Spam, Zip Code OCR
- Text classification: Schapire and Singer - Used stumps with normalized term frequency and multi-class encoding
- OCR: Schwenk and Bengio (neural networks)
- Natural language Processing: Collins; Haruno, Shirai and Ooyama
- Image retrieval: Thieu and Viola
- Medical diagnosis: Merle et al.
- Fraud Detection: Rätsch & Müller 2001
- Drug Discovery: Rätsch, Demiriz, Bennett 2002
- Elect. Power Monitoring: Onoda, Rätsch & Müller 2000

Boosting: Applet & demo

<http://morph.cs.st-andrews.ac.uk/fof/haarDemo/index.html>

<http://cse.ucsd.edu/~yfreund/adaboost/index.html>

<http://vision.ucla.edu/~vedaldi/code/snippets/assets/adademo/adademo.mov>



Background

- [Valiant'84]
 - introduced theoretical PAC model for studying machine learning
- [Kearns&Valiant'88]
 - open problem of finding a boosting algorithm
- [Schapire'89], [Freund'90]
 - first polynomial-time boosting algorithms
- [Drucker, Schapire&Simard '92]
 - first experiments using boosting

Background (cont.)

- [Freund&Schapire '95]
 - Provided AdaBoost algorithm
 - Bigger advantage in practice than previous boosting algorithms
- experiences using AdaBoost:

[Drucker&Cortes '95]	[Schapire&Singer '98]
[Jackson&Cravon '96]	[Maclin&Opitz '97]
[Freund&Schapire '96]	[Bauer&Kohavi '97]
[Quinlan '96]	[Schwenk&Bengio '98]
[Breiman '96]	[Dietterich'98]
- Further development on theory & algorithms:

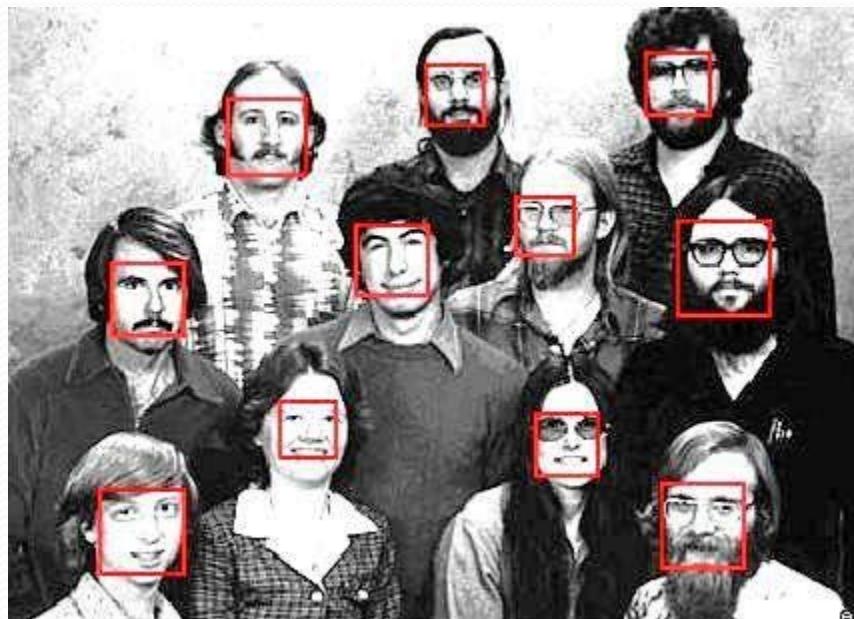
[Schapire,Freund,Bartlett&Lee '97]	[Schapire&Singer '98]
[Breiman '97]	[Mason, Bartlett&Baxter '98]
[Grive and Schuurmans'98]	[Friedman, Hastie&Tibshirani '98]

Variations

- Solve overfitting when noisy data:
 - **Weight Decay** (1998): introduces slack variables regularizing the error
 - **GentleBoost** (1998)
 - **LogitBoost** (2000): additive logistic regression model (statistical point of view)
 - **Regularized AdaBoost** (2000): « soft » margins
 - **BrownBoost** (2001): removes examples many times misclassified, considered as « noisy »
 - **WeightBoost** (2003): weights in final decision dependent on input
- Reduce the number of features (or Weak Learner) :
 - **FloatBoost** (2003): removes the worst WL after each iteration
 - **JointBoost** (2004): multi-class, jointly train N binary classifier sharing same features
- Adaboost Multi-class
 - **AdaBoost.M1**: same as standard but $y \in \{1, \dots, k\}$
 - **AdaBoost.M2**: introduces a confidence degree

Boosting example: Face detection

The Task of Face Detection



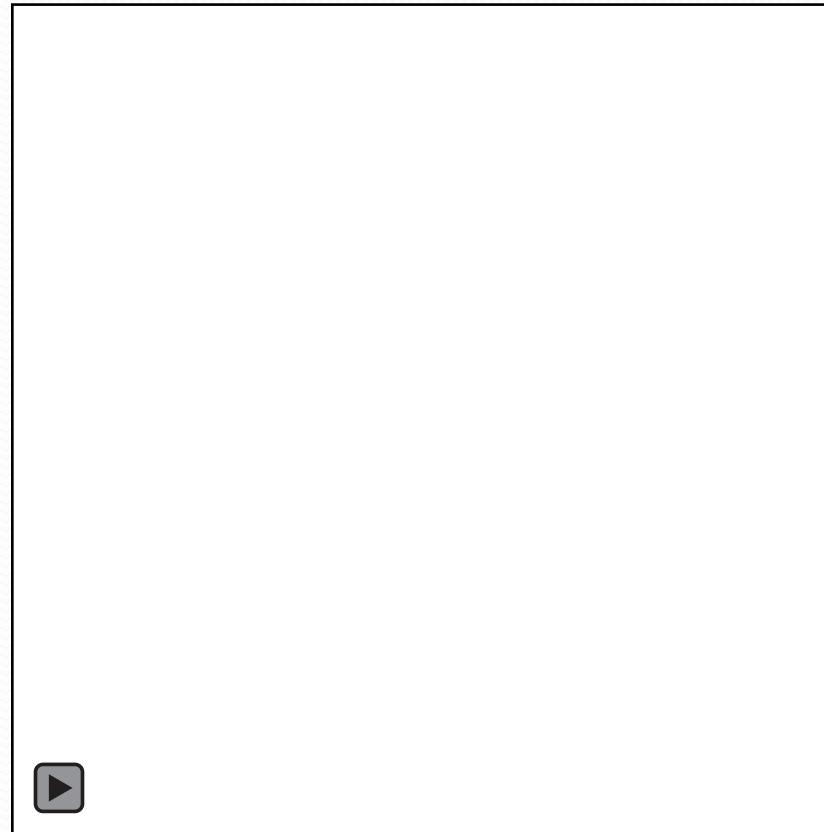
Many slides adapted from P. Viola

Basic Idea

- Slide a window across image and evaluate a face model at every location.



Face Detect Computational Complexity



Challenges

- Slide a window across image and evaluate a face model at every location.
- Sliding window detector must evaluate tens of thousands of location/scale combinations.
- Faces are rare: 0–10 per image
 - For computational efficiency, we should try to spend as little time as possible on the non-face windows
 - A megapixel image has $\sim 10^6$ pixels and a comparable number of candidate face locations
 - To avoid having a false positive in every image, our false positive rate has to be less than 10^{-6}

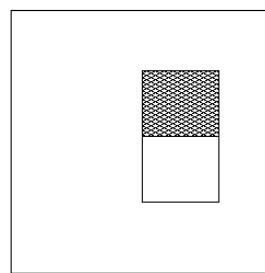
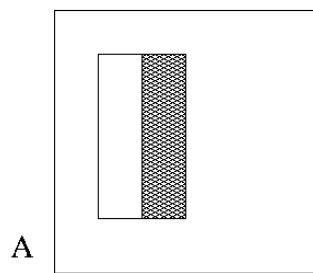
The Viola/Jones Face Detector

- A seminal approach to **real-time object detection**
- Training is **slow**, but **detection** is very **fast**
- Key ideas
 - *Integral images* for **fast** feature evaluation
 - *Boosting* for **feature selection**
 - *Attentional cascade* for **fast** rejection of non-face windows

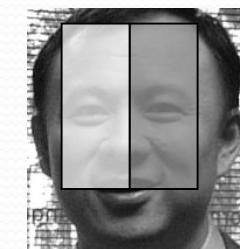
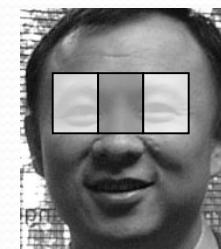
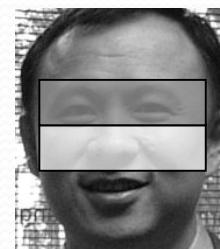
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. CVPR 2001.

P. Viola and M. Jones. *Robust real-time face detection*. IJCV 57(2), 2004.

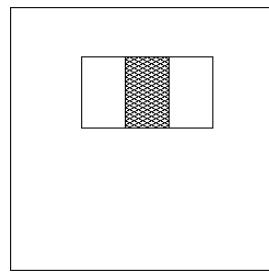
Image Features



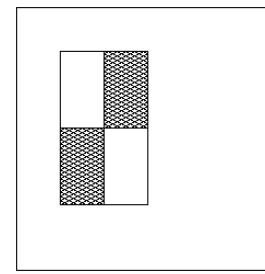
B



A



C



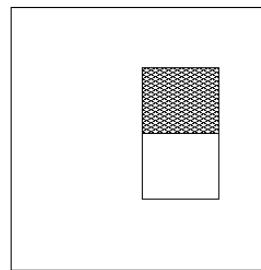
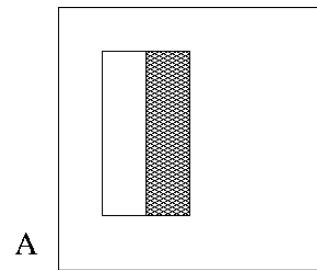
D

Rectangle filters

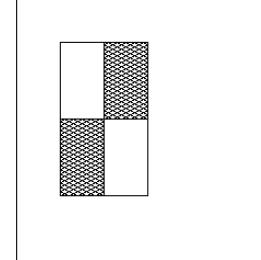
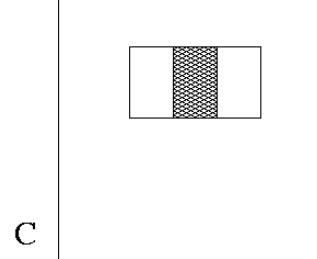
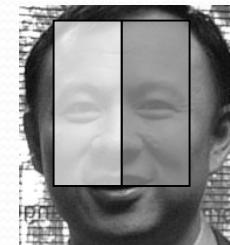
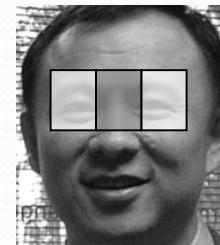
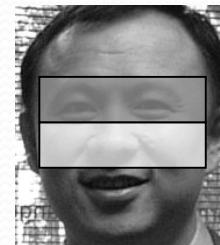
$$\text{Feature Value} = \sum (\text{Pixel in white area}) - \sum (\text{Pixel in black area})$$

Image Features

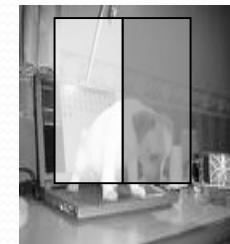
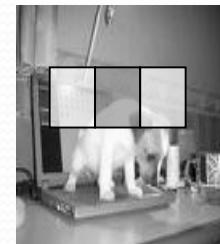
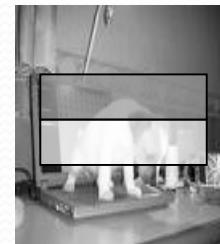
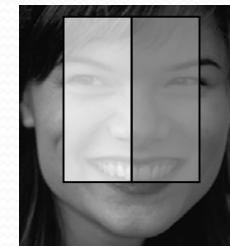
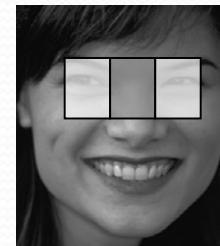
Feature Value = \sum (Pixel in white area) – \sum (Pixel in black area)



B

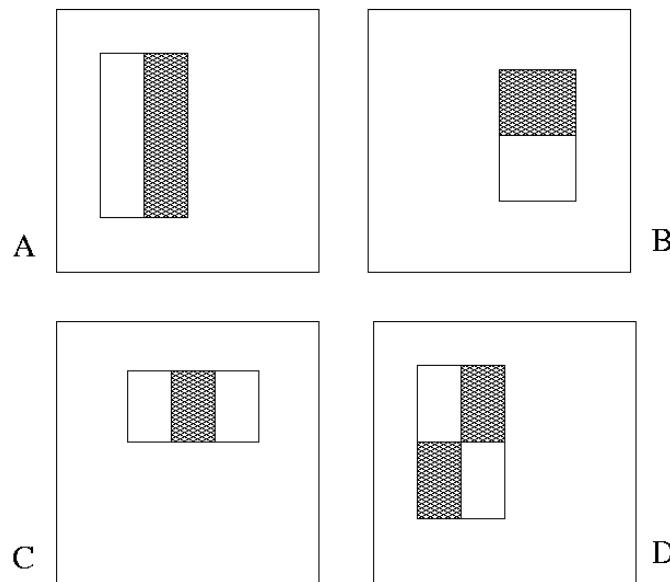


D



Rectangle filters

Size of Feature Space



- How many number of possible rectangle features for a 24×24 detection region?

A+B $2 \sum_{w=1}^{12} \sum_{h=1}^{24} (24 - 2w + 1)(24 - h + 1) +$

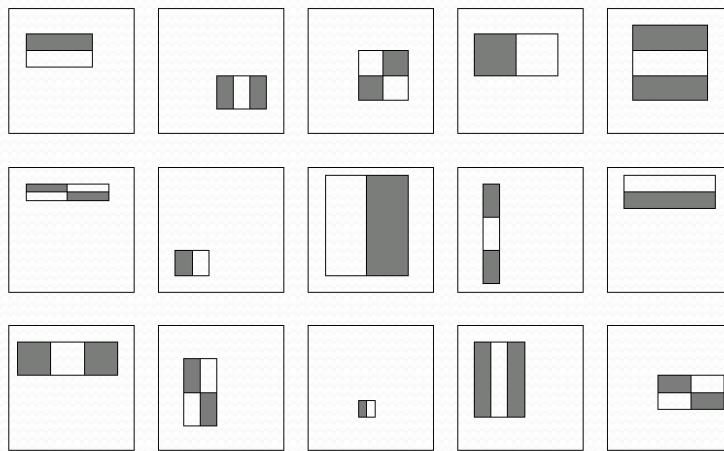
C $2 \sum_{w=1}^8 \sum_{h=1}^{24} (24 - 3w + 1)(24 - h + 1) +$

D $\sum_{w=1}^{12} \sum_{h=1}^{12} (24 - 2w + 1)(24 - 2h + 1)$

Rectangle filters

$\approx 160\ 000$

Feature Selection



What features are good for face detection?

- How many number of possible rectangle features for a 24×24 detection region?

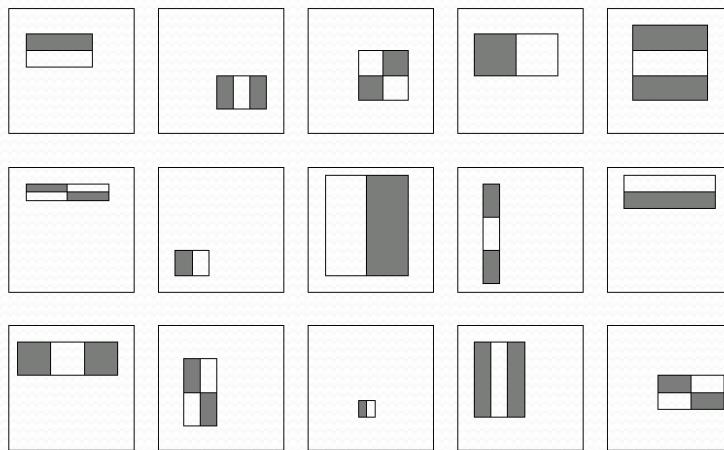
A+B $2 \sum_{w=1}^{12} \sum_{h=1}^{24} (24 - 2w + 1)(24 - h + 1) +$

C $2 \sum_{w=1}^8 \sum_{h=1}^{24} (24 - 3w + 1)(24 - h + 1) +$

D $\sum_{w=1}^{12} \sum_{h=1}^{12} (24 - 2w + 1)(24 - 2h + 1)$

$\approx 160\ 000$

Feature Selection



- Can we create a good classifier using just a small subset of all possible features?
- How to select such a subset?

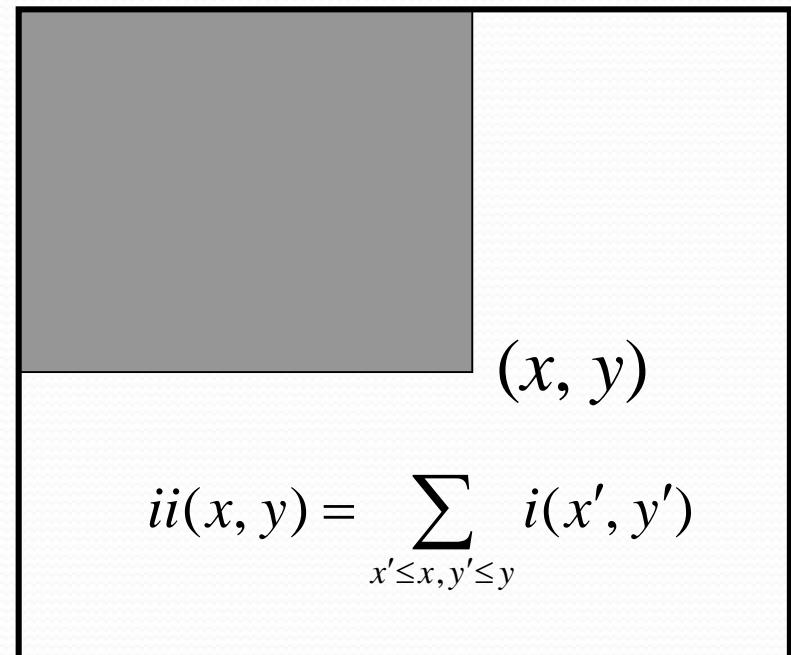
- How many number of possible rectangle features for a 24×24 detection region?

$$\begin{aligned} A+B & 2 \sum_{w=1}^{12} \sum_{h=1}^{24} (24 - 2w + 1)(24 - h + 1) + \\ C & 2 \sum_{w=1}^8 \sum_{h=1}^{24} (24 - 3w + 1)(24 - h + 1) + \\ D & \sum_{w=1}^{12} \sum_{h=1}^{12} (24 - 2w + 1)(24 - 2h + 1) \end{aligned}$$

$\approx 160\ 000$

Integral images

- The integral image computes a value at each pixel (x, y) that is the **sum** of the **pixel values** above and to the left of (x, y) , inclusive.

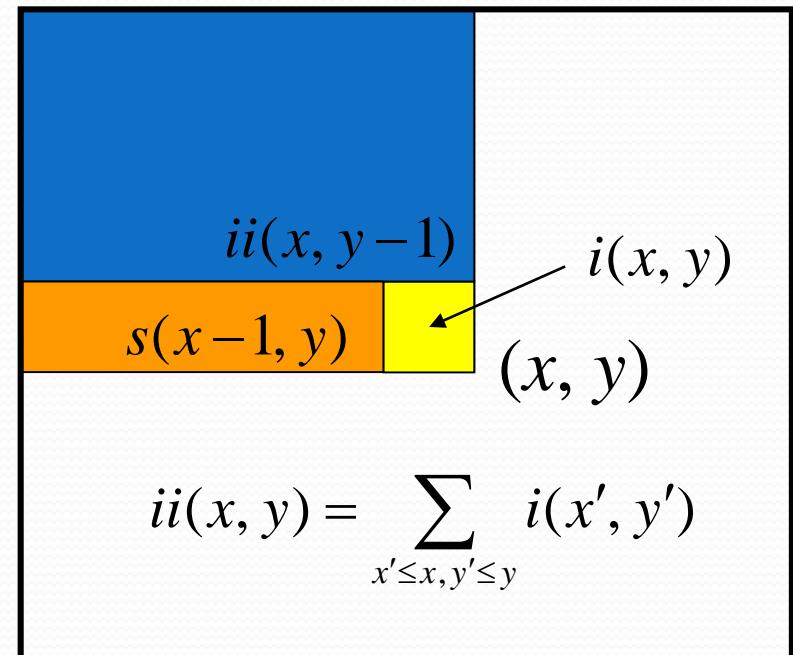


Computing the Integral Image

- The integral image computes a value at each pixel (x, y) that is the **sum** of the **pixel values** above and to the left of (x, y) , inclusive.
- This can quickly be computed in one pass through the image.

$$s(x, y) = s(x - 1, y) + i(x, y)$$

$$ii(x, y) = ii(x, y - 1) + s(x, y)$$



Computing the Integral Image

1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

input image

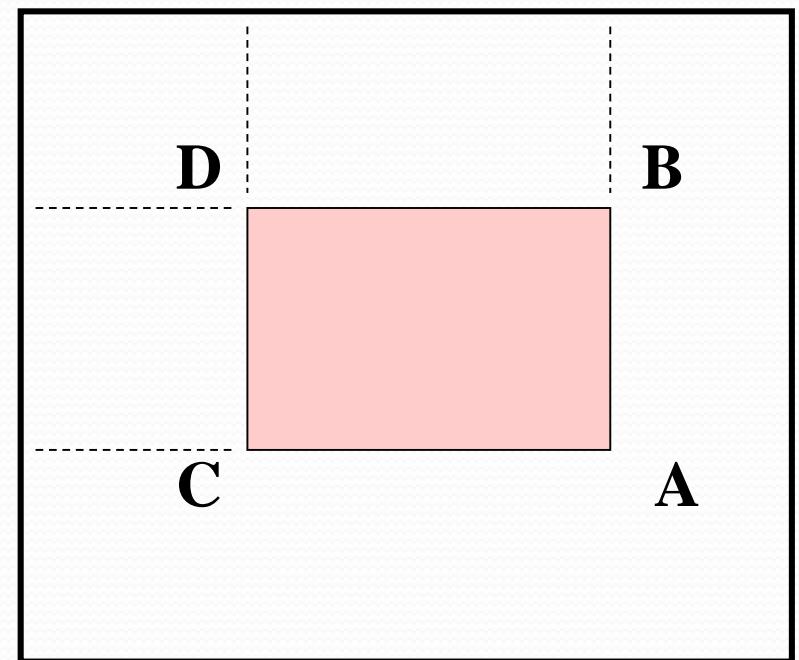
0	0	0	0	0	0
0	1	3	5	9	10
0	4	10	13	22	25
0	6	15	21	32	39
0	10	20	31	46	59
0	16	29	42	58	74

integral image

Computing Sum within a Rectangle

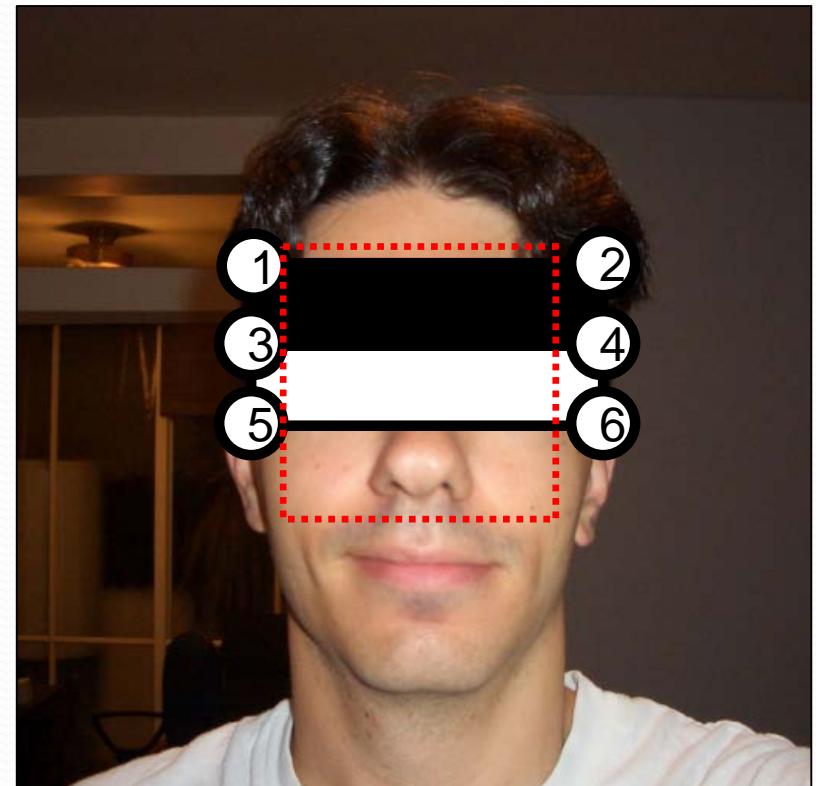
$$\text{sum} = ii_A - ii_B - ii_C + ii_D$$

Only 3 additions are required for
any size of rectangle!



Scaling

- Integral image enables us to evaluate *all* rectangle sizes in constant time.
- Therefore, *no* image scaling is necessary.
- Scale the rectangular features instead!



Boosting

- Boosting is a classification scheme that works by combining *weak learners* into a more accurate ensemble classifier
 - A weak learner need only do better than chance
- Training consists of *multiple boosting rounds*
 - During each boosting round, we select a weak learner that does well on examples that were hard for the previous weak learners
 - “Hardness” is captured by weights attached to training examples

Y. Freund and R. Schapire, *A short introduction to boosting*, *Journal of Japanese Society for Artificial Intelligence*, 14(5):771-780, September, 1999.

The AdaBoost Algorithm

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$ $D_t(i)$: probability distribution of x_i 's at time t

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)] \quad \text{minimize weighted error}$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$ for minimize exponential loss

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

Give error classified patterns more chance for learning.

The AdaBoost Algorithm

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

Output final classifier: $sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Weak Learners for Face Detection

Given: $(x_1, y_1), \dots, (x_m, y_m)$

What base learner is proper for face detection?

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

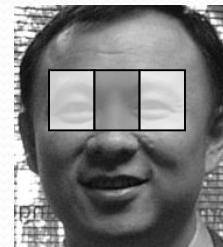
Output final classifier: $\text{sign}\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x) \right)$

Image Features

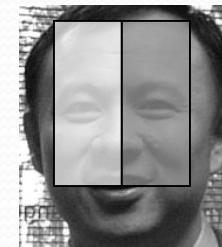
$$\text{Feature Value} = \sum_{\text{Pixel in white area}} - \sum_{\text{Pixel in black area}}$$



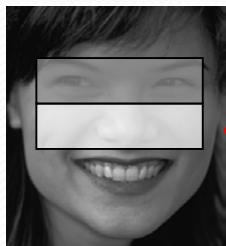
-33



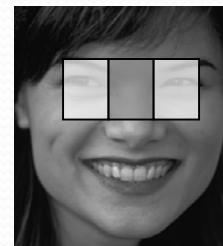
3



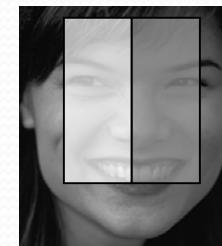
27



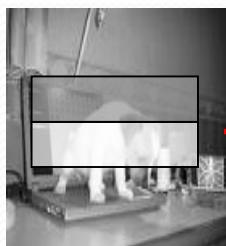
-29



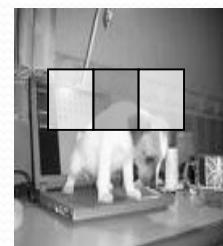
1



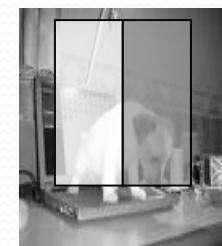
29



-30



28



6

Weak Learners for Face Detection

$$h_1\left(\begin{array}{c} \text{gray} \\ \text{white} \end{array}\right) = \begin{cases} 1 & \text{if } \begin{array}{c} \text{gray} \\ \text{white} \end{array} < -29 \\ 0 & \text{otherwise} \end{cases}$$

$$h_2\left(\begin{array}{c|c|c} \text{white} & \text{gray} & \text{white} \end{array}\right) = \begin{cases} 1 & \text{if } \begin{array}{c|c|c} \text{white} & \text{gray} & \text{white} \end{array} < 26 \\ 0 & \text{otherwise} \end{cases}$$

$$h_3\left(\begin{array}{c|c} \text{white} & \text{gray} \end{array}\right) = \begin{cases} 1 & \text{if } \begin{array}{c|c} \text{white} & \text{gray} \end{array} > 11 \\ 0 & \text{otherwise} \end{cases}$$

Weak Learners for Face Detection

$$h_t(W) = \begin{cases} 1 & \text{if } p_t f_t(W) > p_t \theta_t \\ 0 & \text{otherwise} \end{cases}$$

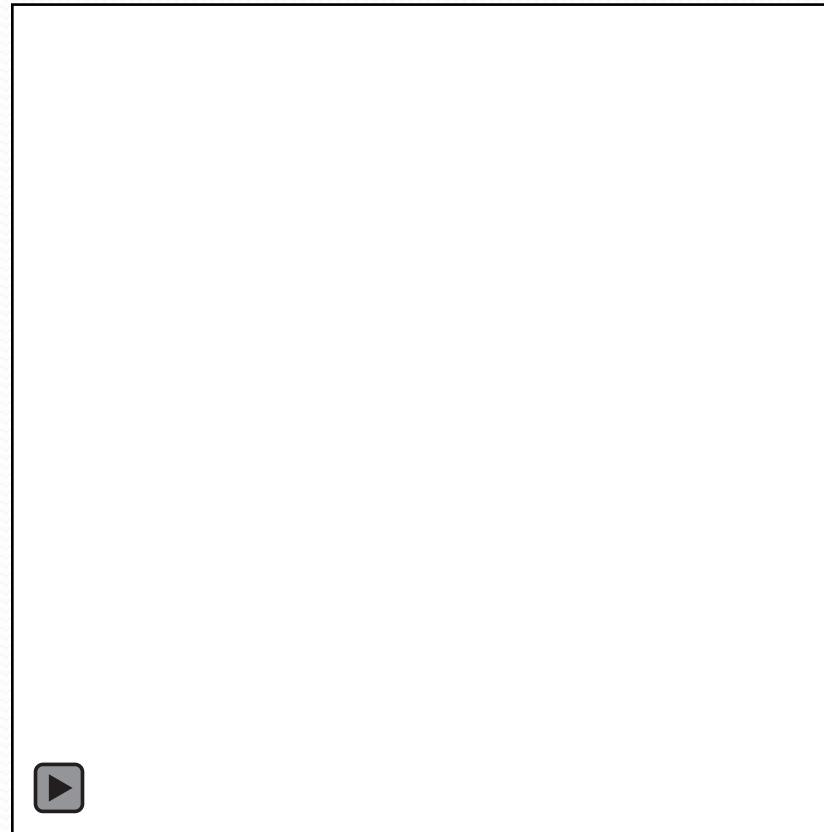
value of rectangle feature →
window
parity →
threshold →

$$h_1(\square) = \begin{cases} 1 & \text{if } (-1)\square > (-1)(-29) \\ 0 & \text{otherwise} \end{cases}$$

$$h_2(\square|\square) = \begin{cases} 1 & \text{if } (-1)\square|\square > (-1)26 \\ 0 & \text{otherwise} \end{cases}$$

$$h_3(\square|\square) = \begin{cases} 1 & \text{if } (+1)\square|\square > (+1)11 \\ 0 & \text{otherwise} \end{cases}$$

Face Detect Computational Complexity

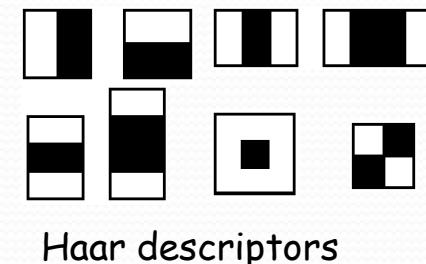


Boosting

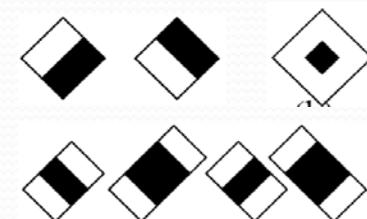
- Training set contains **face** and **nonface** examples
 - Initially, with equal weight
- For each round of boosting:
 - Evaluate each **rectangle filter** on each example
 - Select best **threshold** for each filter
 - Select best **filter/threshold combination**
 - Reweight examples
- Computational complexity of learning: $O(MNK)$
 - M rounds, N examples, K features

AdaBoost

- 1990 Schapire, Boosting
- 1996 Freund & Schapire, AdaBoost
 - Do not require any a priori knowledge
- 2001 Viola & Jones
 - Face detection
 - Haar descriptors, integral image, cascades
 - Robust, fast
- 2002 Lienhart et al.
 - Extended descriptors, 45 degrees rotation



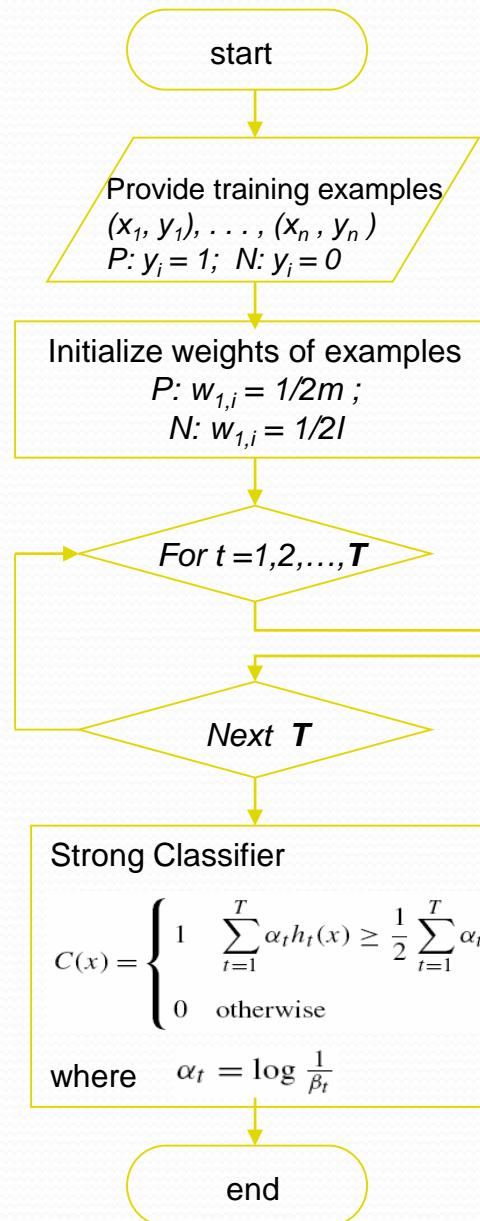
Haar descriptors



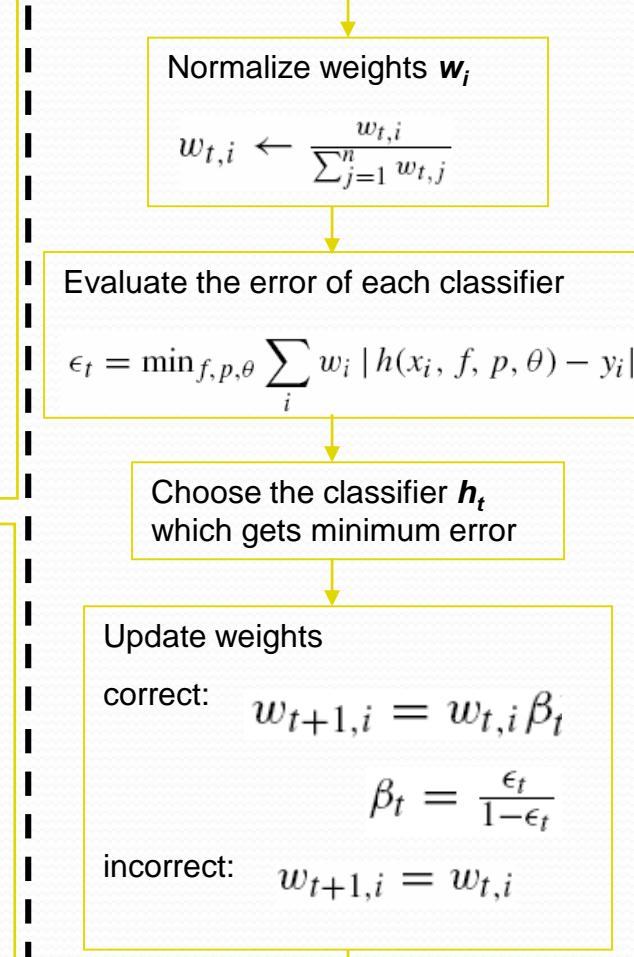
Extended descriptors

AdaBoost

Obtenir un classifieur fort

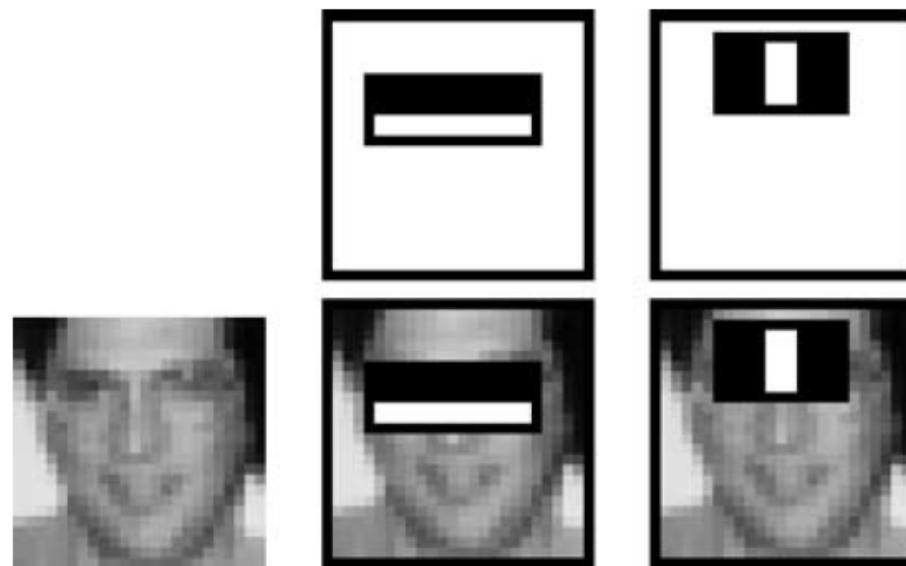


Choisir un classifieur faible



Features Selected by Boosting

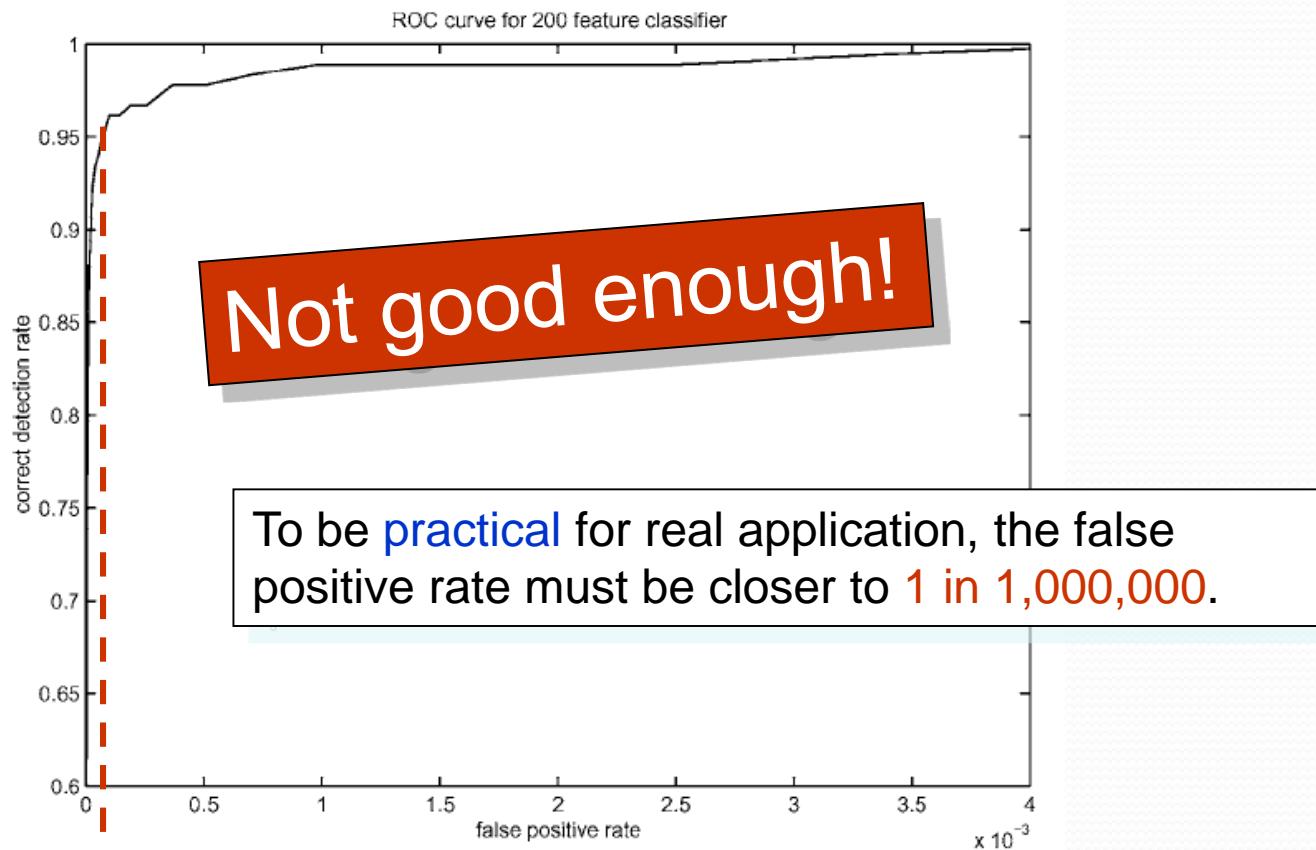
First two features selected by boosting:



This feature combination can yield 100% detection rate and 50% false positive rate

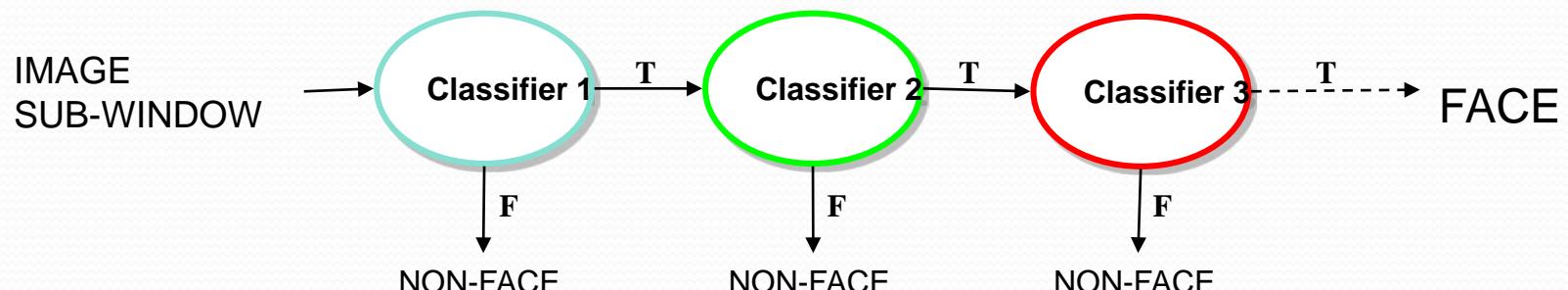
ROC Curve for 200-Feature Classifier

A 200-feature classifier can yield **95%** detection rate and a **false positive rate** of **1 in 14084**.



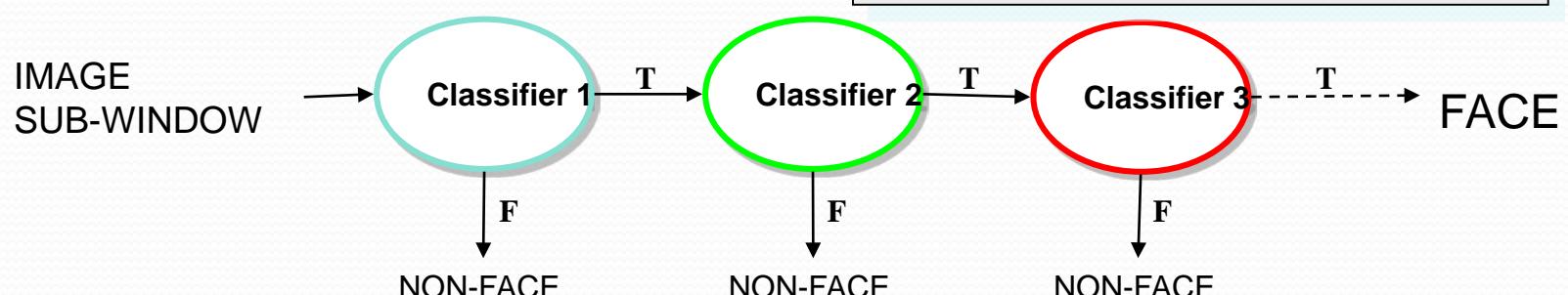
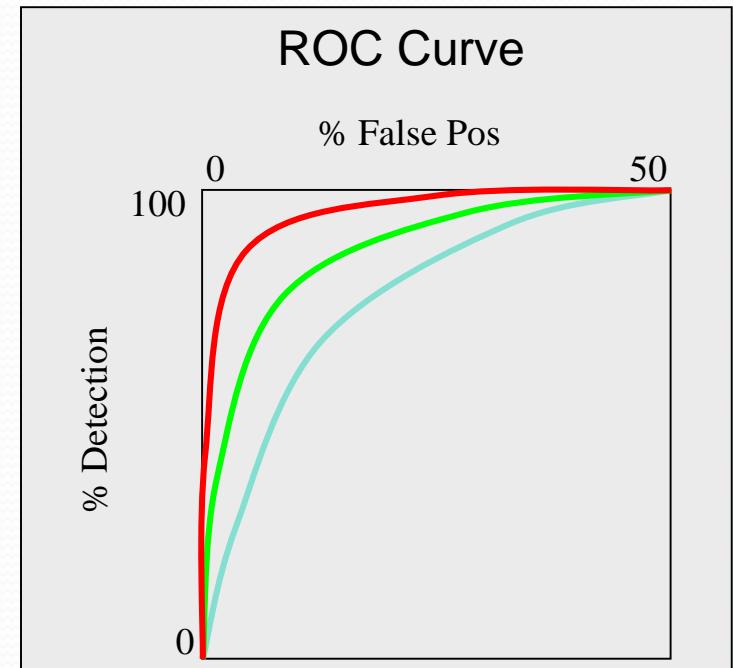
Attentional Cascade

- We start with simple classifiers which reject many of the negative sub-windows while detecting almost all positive sub-windows
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on
- A negative outcome at any point leads to the immediate rejection of the sub-window



Attentional Cascade

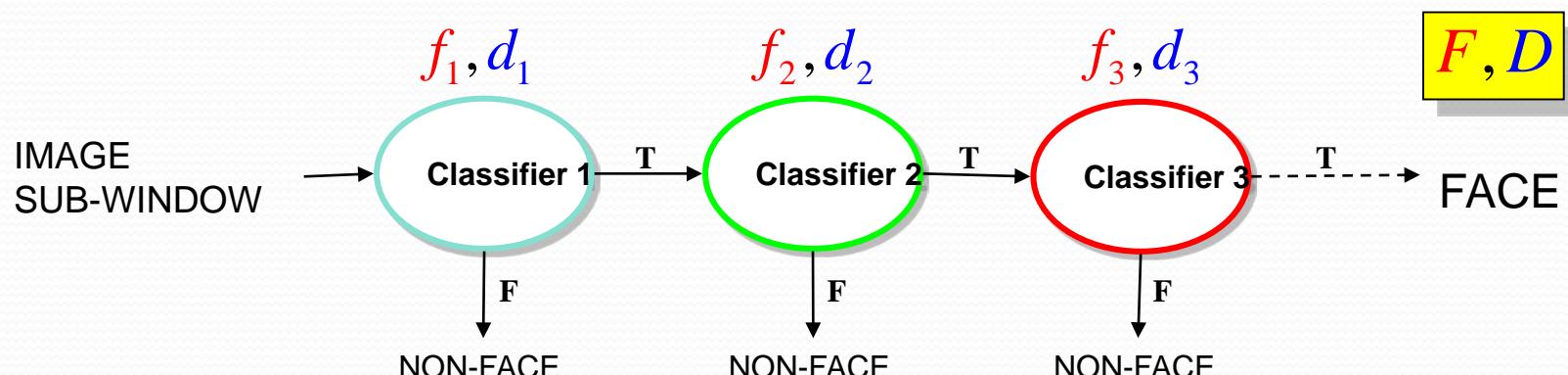
- Chain classifiers that are progressively more complex and have lower false positive rates



Detection Rate and False Positive Rate for Chained Classifiers

$$F = \prod_{i=1}^K f_i, \quad D = \prod_{i=1}^K d_i$$

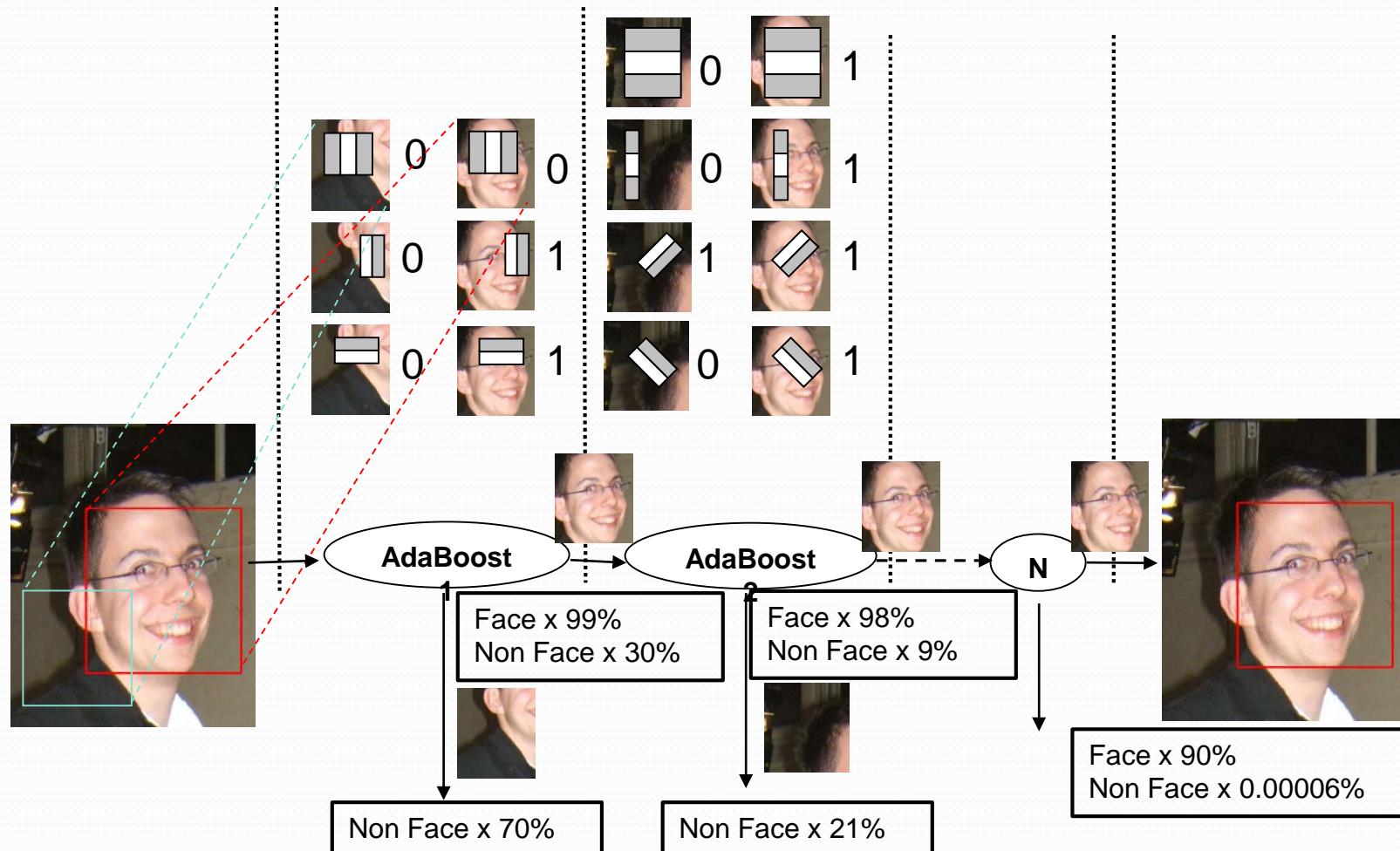
- The detection rate and the false positive rate of the cascade are found by multiplying the respective rates of the individual stages
- A detection rate of **0.9** and a false positive rate on the order of **10^{-6}** can be achieved by a **10-stage** cascade if each stage has a detection rate of **0.99** ($0.99^{10} \approx 0.9$) and a false positive rate of about **0.30** ($0.3^{10} \approx 6 \times 10^{-6}$)



Training the Cascade

- Set target detection and false positive rates for each stage
- Keep adding features to the current stage until its target rates have been met
 - Need to lower AdaBoost threshold to maximize detection (as opposed to minimizing total classification error)
 - Test on a *validation set*
- If the overall false positive rate is *not* low enough, then add another stage
- Use false positives from current stage as the negative training examples for the next stage

AdaBoost (6) – Cascade Principle



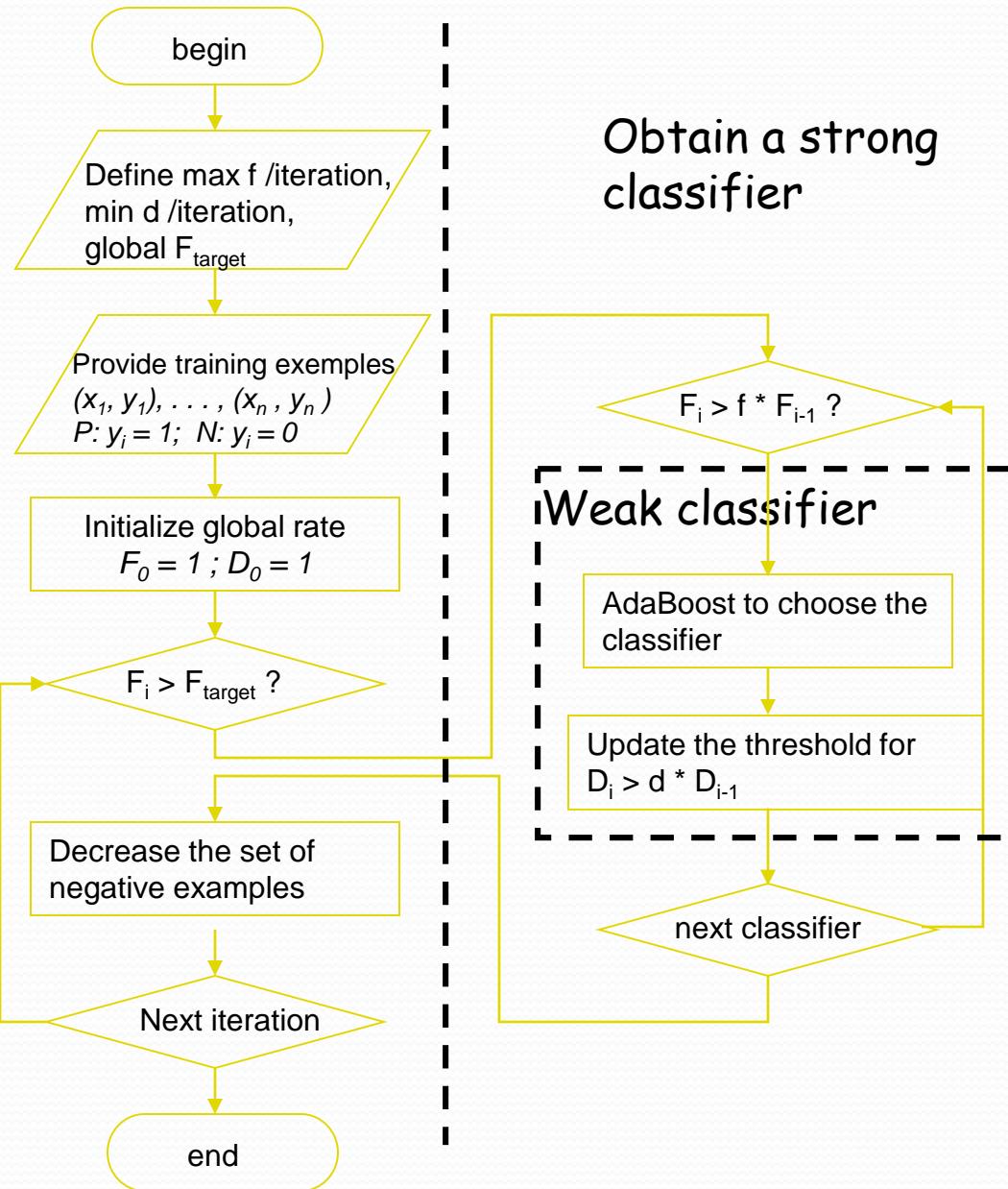
Algo

Table 2. The training algorithm for building a cascaded detector.

- User selects values for f , the maximum acceptable false positive rate per layer and d , the minimum acceptable detection rate per layer.
- User selects target overall false positive rate, F_{target} .
- P = set of positive examples
- N = set of negative examples
- $F_0 = 1.0; D_0 = 1.0$
- $i = 0$
- while $F_i > F_{target}$
 - $i \leftarrow i + 1$
 - $n_i = 0; F_i = F_{i-1}$
 - while $F_i > f \times F_{i-1}$
 - * $n_i \leftarrow n_i + 1$
 - * Use P and N to train a classifier with n_i features using AdaBoost
 - * Evaluate current cascaded classifier on validation set to determine F_i and D_i .
 - * Decrease threshold for the i th classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects F_i)
 - $N \leftarrow \emptyset$
 - If $F_i > F_{target}$ then evaluate the current cascaded detector on the set of non-face images and put any false detections into the set N

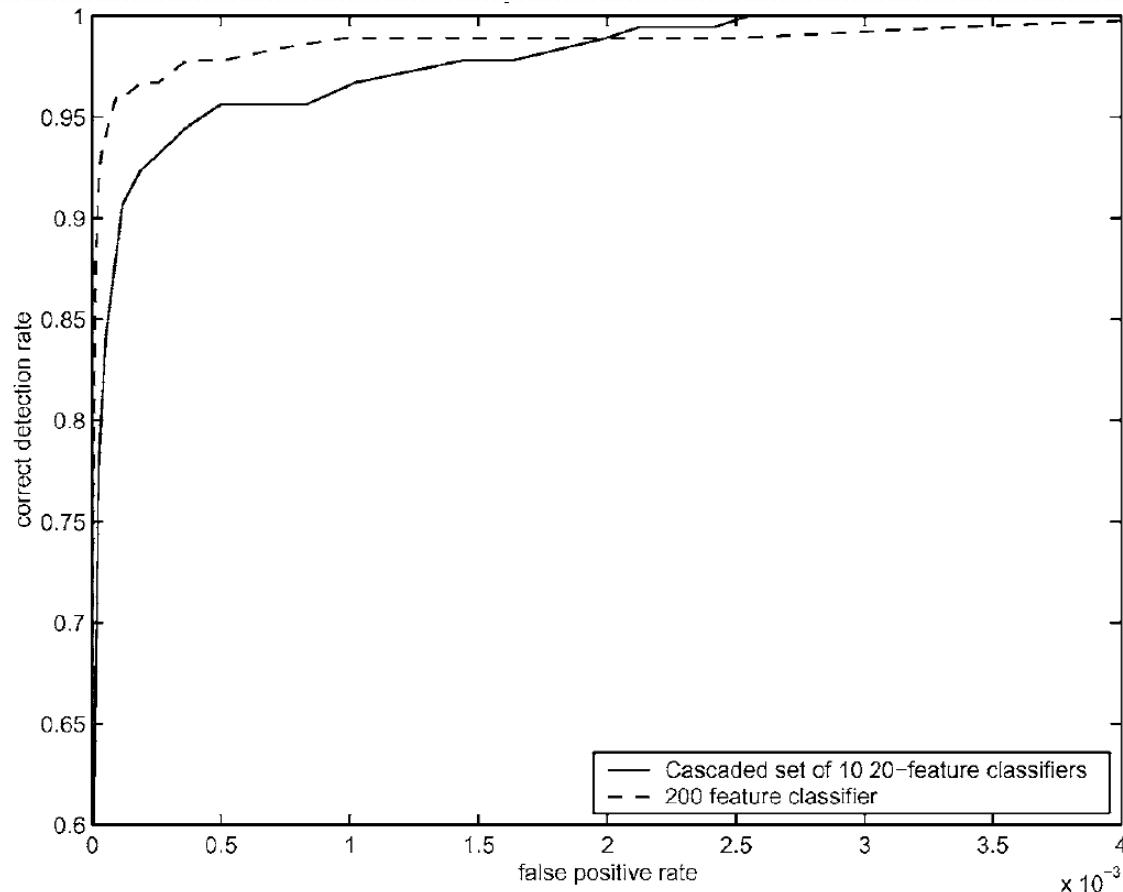
Attentional Cascade

Obtain a cascade of strong classifiers

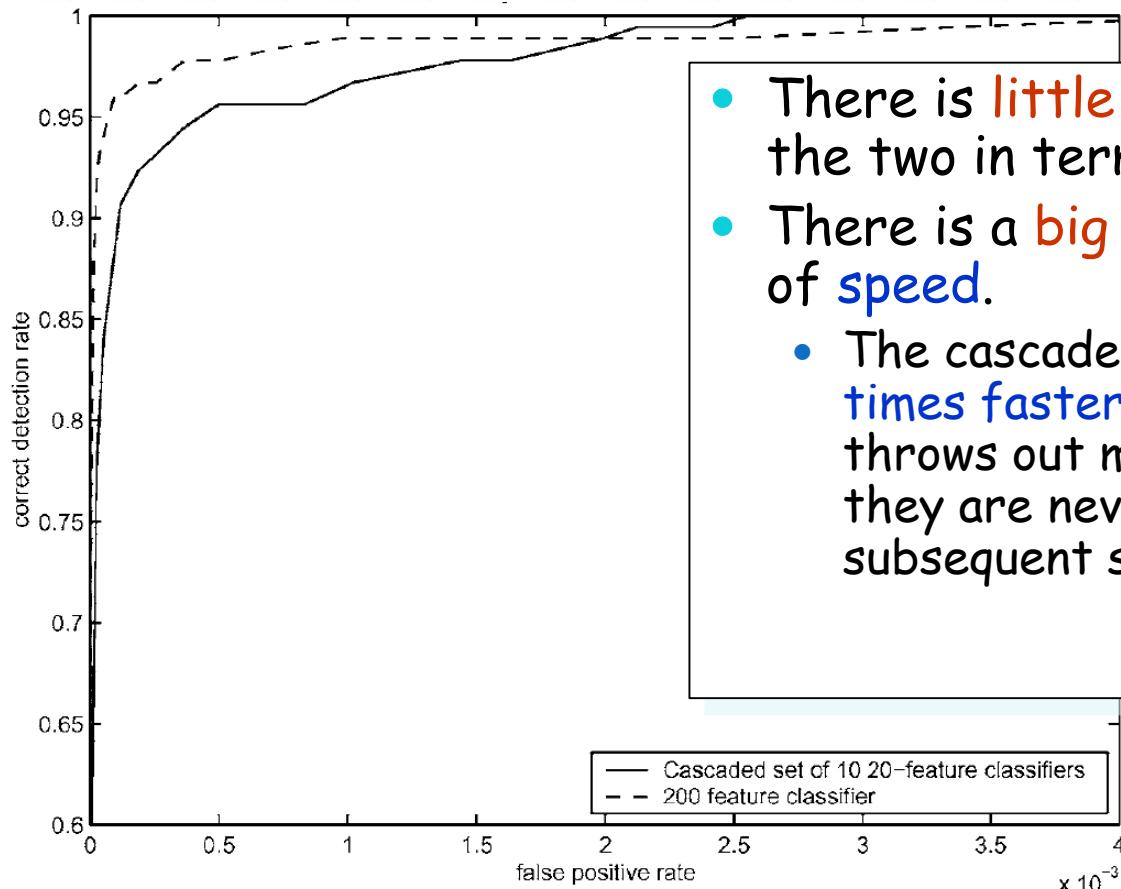


Obtain a strong classifier

ROC Curves Cascaded Classifier to Monolithic Classifier



ROC Curves Cascaded Classifier to Monolithic Classifier



- There is **little difference** between the two in terms of accuracy.
- There is a **big difference** in terms of **speed**.
 - The cascaded classifier is nearly **10 times faster** since its first stage throws out most non-faces so that they are never evaluated by subsequent stages.

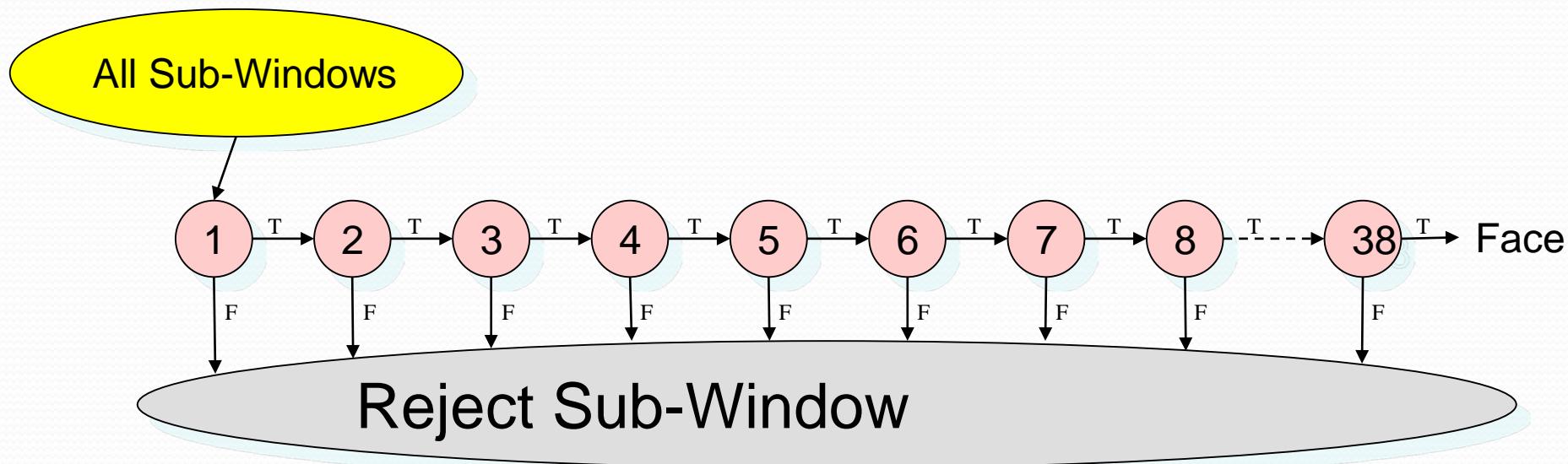
The Implemented System

- Training Data
 - 5000 faces
 - All frontal, rescaled to 24x24 pixels
 - 300 million non-faces
 - 9500 non-face images
 - Faces are normalized
 - Scale, translation
- Many variations
 - Across individuals
 - Illumination
 - Pose

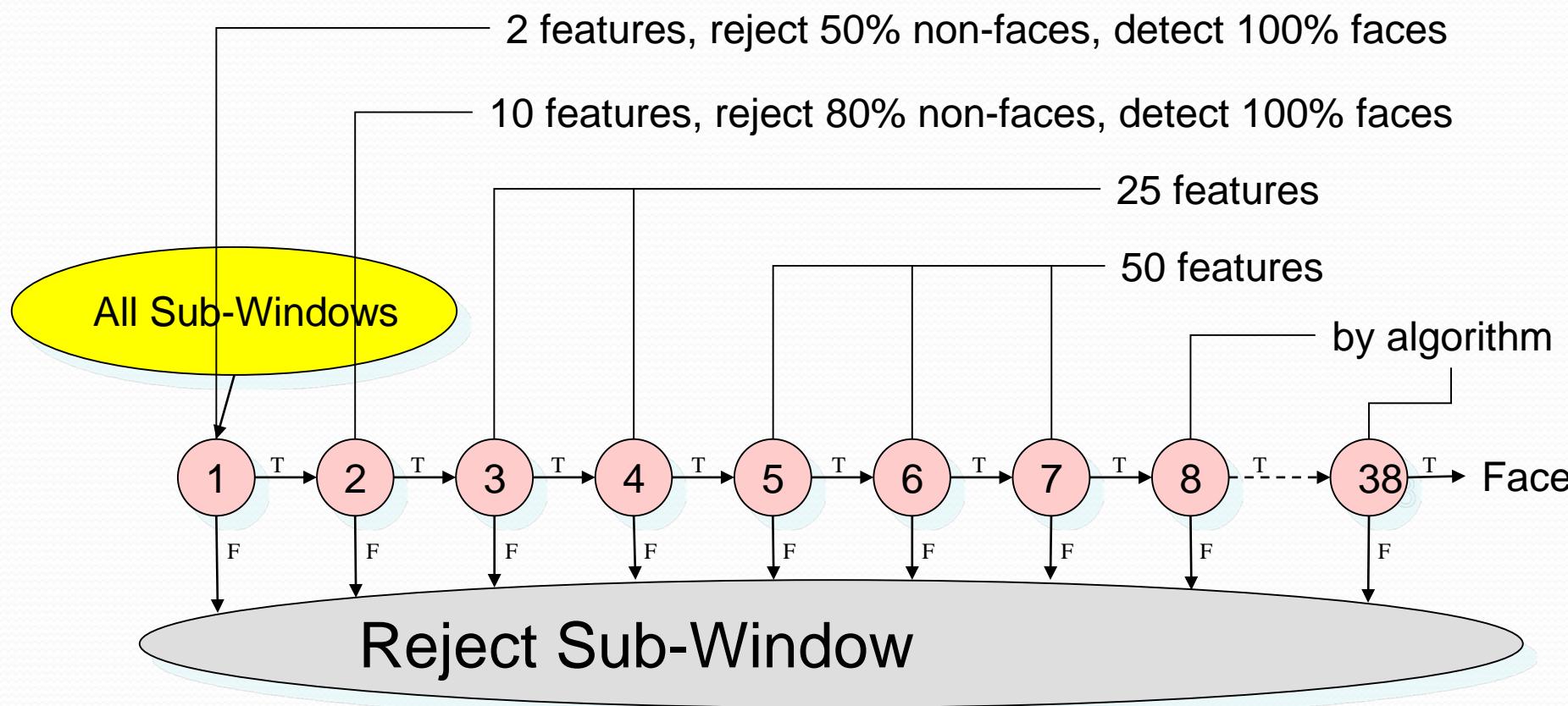


Structure of the Detector Cascade

- Combining successively more complex classifiers in cascade
 - 38 stages
 - included a total of 6060 features



Structure of the Detector Cascade



Speed of the Final Detector

- On a 700 Mhz Pentium III processor, the face detector can process a 384×288 pixel image in about **.067 seconds**
 - 15 Hz
 - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)
- Average of **8 features** evaluated per window on test set

Image Processing

- Training — all example sub-windows were **variance normalized** to minimize the effect of different lighting conditions
- Detection — **variance normalized** as well

$$\sigma^2 = m^2 - \frac{1}{N} \sum x^2$$

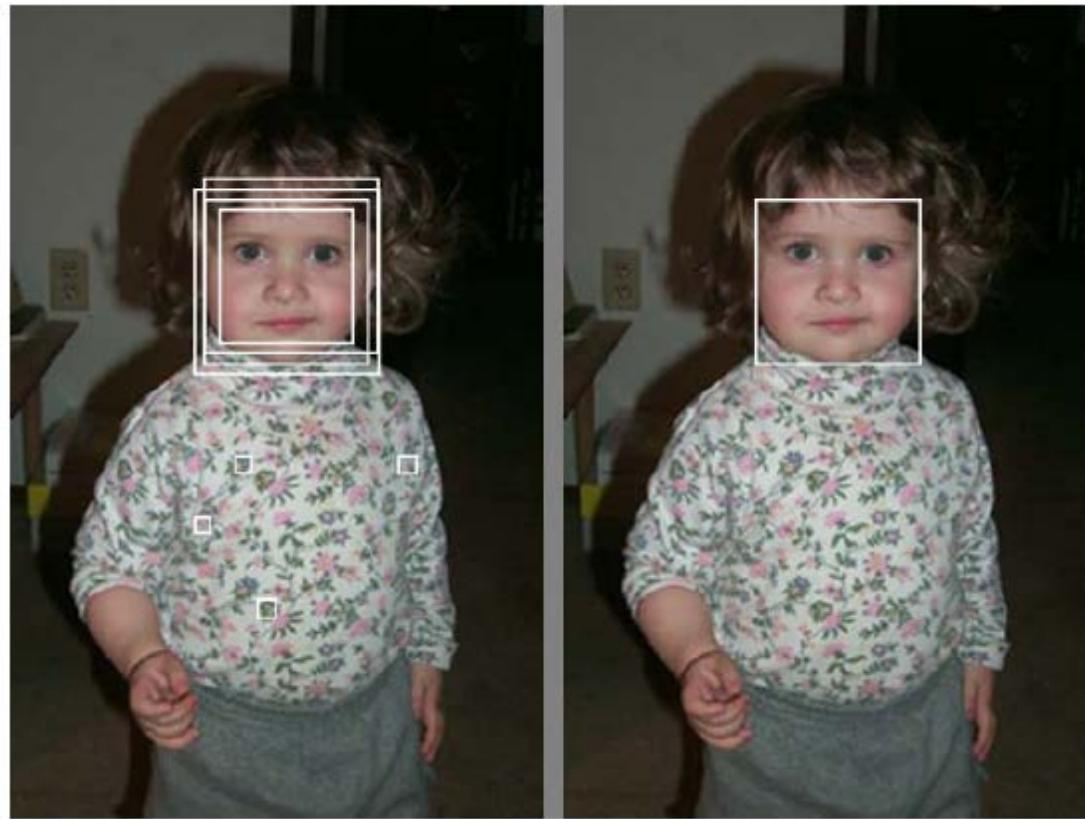
Can be computed using
integral image

Can be computed using
integral image of
squared image

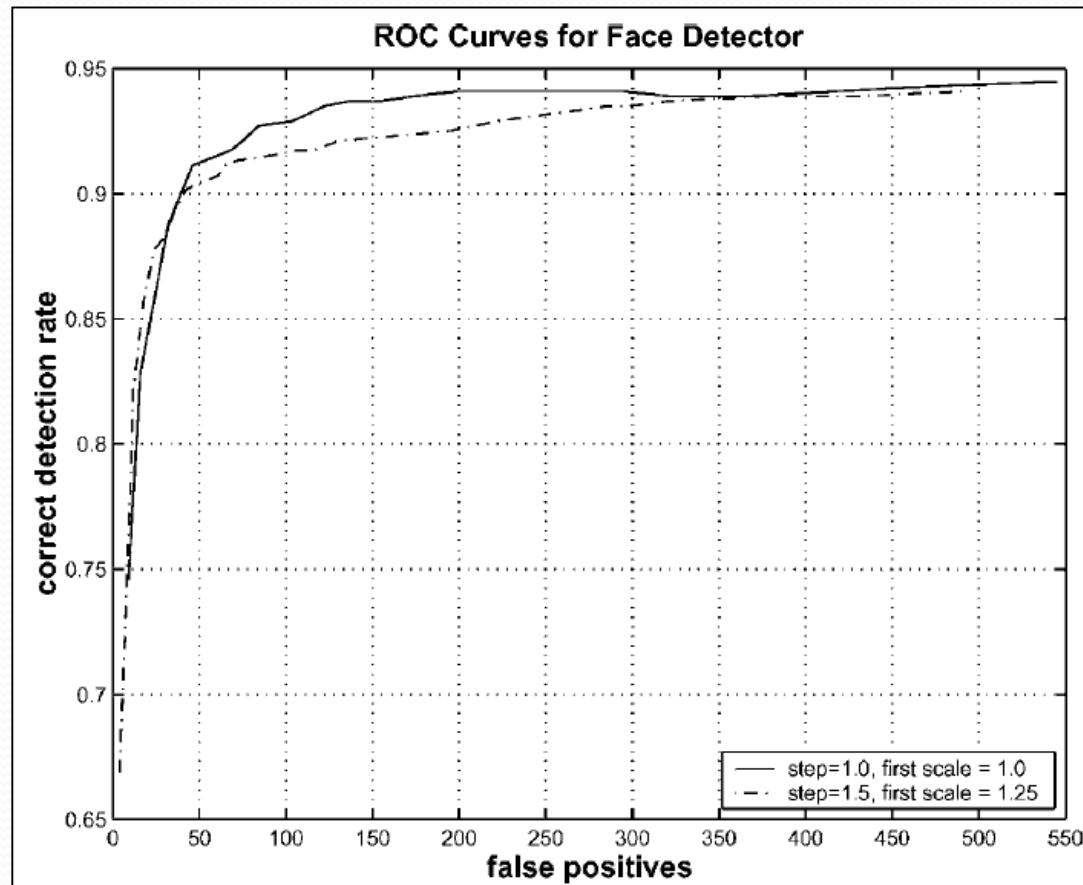
Scanning the Detector

- Scaling is achieved by scaling the detector itself, rather than scaling the image
- Good detection results for scaling factor of 1.25
- The detector is scanned across location
 - Subsequent locations are obtained by shifting the window [$s\Delta$] pixels, where s is the current scale
 - The result for $\Delta = 1.0$ and $\Delta = 1.5$ were reported

Merging Multiple Detections



ROC Curves for Face Detection



Results

- Fixed images
- Video sequence

○ Frontal face

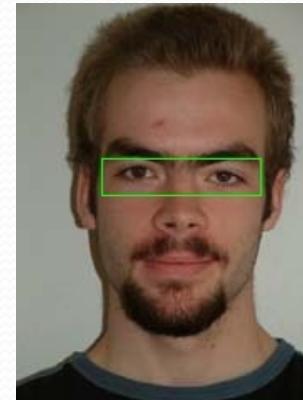
○ Left profile face

○ Right profile face



Extension

- Fast and robust
- Other descriptors instead of Haar
- Other cascades (rotation...)
- Eye detection



The AdaBoost Algorithm

What **goal** the AdaBoost want

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier:

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, Z_t \text{ is for normalization}$$

Output final classifier:

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

Goal

Final classifier: $\text{sign}\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x) \right)$

Minimize exponential loss

$$\text{loss}_{\text{exp}}[H(x)] = E_{x,y} [e^{-yH(x)}]$$

Maximize the margin $yH(x)$

Minimize $loss_{\exp}[H(x)] = E_{x,y} \left[e^{-yH(x)} \right]$

Goal

Final classifier: $sign \left(H(x) = \sum_{t=1}^T \alpha_t h_t(x) \right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$\begin{aligned}
 E_{x,y} \left[e^{-yH_t(x)} \right] &= E_x \left[E_y \left[e^{-yH_t(x)} \mid x \right] \right] \\
 &= E_x \left[E_y \left[e^{-y[H_{t-1}(x) + \alpha_t h_t(x)]} \mid x \right] \right] \\
 &= E_x \left[E_y \left[e^{-yH_{t-1}(x)} e^{-y\alpha_t h_t(x)} \mid x \right] \right] \\
 &= E_x \left[E_y \left[e^{-yH_{t-1}(x)} \left[e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x)) \right] \mid x \right] \right]
 \end{aligned}$$

Goal

$\alpha_t = ?$

Minimize $loss_{\text{exp}}[H(x)] = E_{x,y}[e^{-yH(x)}]$

Final classifier: $sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$E_{x,y}\left[e^{-yH_t(x)}\right] = E_x\left[E_y\left[e^{-yH_{t-1}(x)} \left[e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x))\right] | x\right]\right]$$

Set $\frac{\partial}{\partial \alpha_t} E_{x,y}\left[e^{-yH_t(x)}\right] = 0$

$$\Rightarrow E_x\left[E_y\left[e^{-yH_{t-1}(x)} \underbrace{\left[-e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x))\right]}_0 | x\right]\right] = 0$$

Goal

$\alpha_t = ?$

$$\text{Minimize} \quad loss_{\exp}[H(x)] = E_{x,y} [e^{-yH(x)}]$$

$$\text{Final classifier: } sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$\Rightarrow E_x \left[E_y \left[e^{-yH_{t-1}(x)} \underbrace{\left[-e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x)) \right]}_0 | x \right] \right] = 0$$

$$\Rightarrow \alpha_t = \frac{1}{2} \ln \frac{P(y = h_t(x))}{P(y \neq h_t(x))} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$

$$P(x_i, y_i) = D_t(i)$$

$$\varepsilon_t = P(\text{error}) \approx \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

Goal

$\alpha_t = ?$

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$
 Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$
 For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

Then, $H(x) = H_t(x)$
 Output final classifier: $\text{sign}\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

$$\Rightarrow E_x \left[E_y \left[e^{-yH_{t-1}(x)} \underbrace{\left[-e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x)) \right]}_{0} | x \right] \right] = 0$$

$$\Rightarrow \alpha_t = \frac{1}{2} \ln \frac{P(y = h_t(x))}{P(y \neq h_t(x))} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \quad P(x_i, y_i) = D_t(i)$$

$$\varepsilon_t = P(\text{error}) \approx \sum_{i=1}^m D_t(i)[y_i \neq h_t(x_i)]$$

Goal

$$D_{t+1} = ?$$

Define $H_t(x) = \dots$

Then, $H(x) = H_t(x) + \dots$

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

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

$$\Rightarrow E_x \left[E_y \left[e^{-yH_{t-1}(x)} \underbrace{\left[-e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x)) \right]}_{0} | x \right] \right] = 0$$

$$\Rightarrow \alpha_t = \frac{1}{2} \ln \frac{P(y = h_t(x))}{P(y \neq h_t(x))} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \quad P(x_i, y_i) = D_t(i)$$

$$\varepsilon_t = P(\text{error}) \approx \sum_{i=1}^m D_t(i)[y_i \neq h_t(x_i)]$$

Goal

$$D_{t+1} = ?$$

Minimize $loss_{\exp}[H(x)] = E_{x \sim D, y} [e^{-yH(x)}]$

Final classifier: $sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$E_{x,y} [e^{-yH_t}] = E_{x,y} [e^{-yH_{t-1}} e^{-y\alpha_t h_t}] \approx E_{x,y} \left[e^{-yH_{t-1}} \left(1 - y\alpha_t h_t + \frac{1}{2} \alpha_t^2 y^2 h_t^2 \right) \right]$$

$$\Rightarrow h_t = \arg \min_h E_{x,y} \left[e^{-yH_{t-1}} \left(1 - y\alpha_t h + \frac{1}{2} \alpha_t^2 y^2 h^2 \right) \right] \quad y^2 h^2 = 1$$

$$\Rightarrow h_t = \arg \min_h E_{x,y} \left[e^{-yH_{t-1}} \left(1 - y\alpha_t h + \frac{1}{2} \alpha_t^2 \right) \right]$$

$$\Rightarrow h_t = \arg \min_h E_x \left[E_y \left[e^{-yH_{t-1}} \left(1 - y\alpha_t h + \frac{1}{2} \alpha_t^2 \right) | x \right] \right]$$

Goal

$$D_{t+1} = ?$$

Minimize $loss_{\exp}[H(x)] = E_{x \sim D, y} [e^{-yH(x)}]$

Final classifier: $sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$\Rightarrow h_t = \arg \min_h E_x \left[E_y \left[e^{-yH_{t-1}} \left(1 - y\alpha_t h + \frac{1}{2} \alpha_t^2 \right) \right] | x \right]$$

$$\Rightarrow h_t = \arg \min_h E_x \left[E_y \left[e^{-yH_{t-1}} (-y\alpha_t h) \right] | x \right]$$

$$\Rightarrow h_t = \arg \max_h E_x \left[E_y \left[e^{-yH_{t-1}} (yh) \right] | x \right]$$

$$\Rightarrow h_t = \arg \max_h E_x \left[1 \cdot h(x) e^{-H_{t-1}(x)} \cdot P(y=1|x) + (-1) \cdot h(x) e^{H_{t-1}(x)} \cdot P(y=-1|x) \right]$$

Goal

$$D_{t+1} = ?$$

Minimize $loss_{\exp}[H(x)] = E_{x \sim D, y} [e^{-yH(x)}]$

Final classifier: $sign\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$\Rightarrow h_t = \arg \max_h E_x \left[1 \cdot h(x) e^{-H_{t-1}(x)} \cdot P(y=1|x) + (-1) \cdot h(x) e^{H_{t-1}(x)} \cdot P(y=-1|x) \right]$$

$$\Rightarrow h_t = \arg \max_h E_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)} [yh(x)] \text{ maximized when } y = h(x) \quad \forall x$$

$$\Rightarrow h_t = sign\left(E_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)} [y | x]\right)$$

$$\Rightarrow h_t = sign\left(P_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)} (y=1|x) - P_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)} (y=-1|x)\right)$$

Goal

$$D_{t+1} = ?$$

Minimize $\text{loss}_{\exp}[H(x)] = E_{x \sim D, y} [e^{-yH(x)}]$

Final classifier: $\text{sign}\left(H(x) = \sum_{t=1}^T \alpha_t h_t(x)\right)$

Define $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$ with $H_0(x) = 0$

Then, $H(x) = H_T(x)$

$$\Rightarrow h_t = \text{sign}\left(P_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)}(y=1 | x) - P_{x, y \sim e^{-yH_{t-1}(x)} P(y|x)}(y=-1 | x)\right)$$

At time t $x, y \sim e^{-yH_{t-1}(x)} P(y | x)$

Goal

$$D_{t+1} = ?$$

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \dots, m$

For $t = 1, \dots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t , i.e.,

$$h_t = \arg \min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution: $D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}$, Z_t is for normalization

Define $H_t(x) = \sum_{i=1}^T \alpha_i h_i(x)$

Then, $H(x) = H_T(x)$

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

At time t $x, y \sim e^{-yH_{t-1}(x)} P(y | x)$

At time 1 $x, y \sim P(y | x)$ $P(y_i | x_i) = 1 \Rightarrow D_1(i) = \frac{1}{Z_1} = \frac{1}{m}$

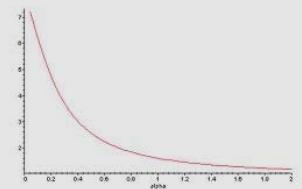
At time $t+1$ $x, y \sim e^{-yH_t(x)} P(y | x) \equiv D_t e^{-\alpha_t y h_t(x)}$

$$\Rightarrow D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, Z_t \text{ is for normalization}$$

Error of generalizaton

Error of generalization of H can be bounded by:

$$E_{\text{Real}}(H_T) = E_{\text{HingeLoss}}(H_T) + O\left(\sqrt{\frac{T \cdot d}{m}}\right)$$



where T is the number of boosting iterations, m the number of examples, and d a dimension (VC-dimension) of H_T space (« weaks learner complexity »)

When T becomes large, the 2nd term becomes large too, then one can think that the learner would **overfit**
⇒ the solution = over-weighting of ambiguous examples

Conclusions

We have seen in this lecture:

- How adaboost works?
- Why adaboost works?
- Adaboost for face detection
 - Rectangle features
 - Integral images for fast computation
 - Boosting for feature selection
 - Attentional cascade for fast rejection of negative windows