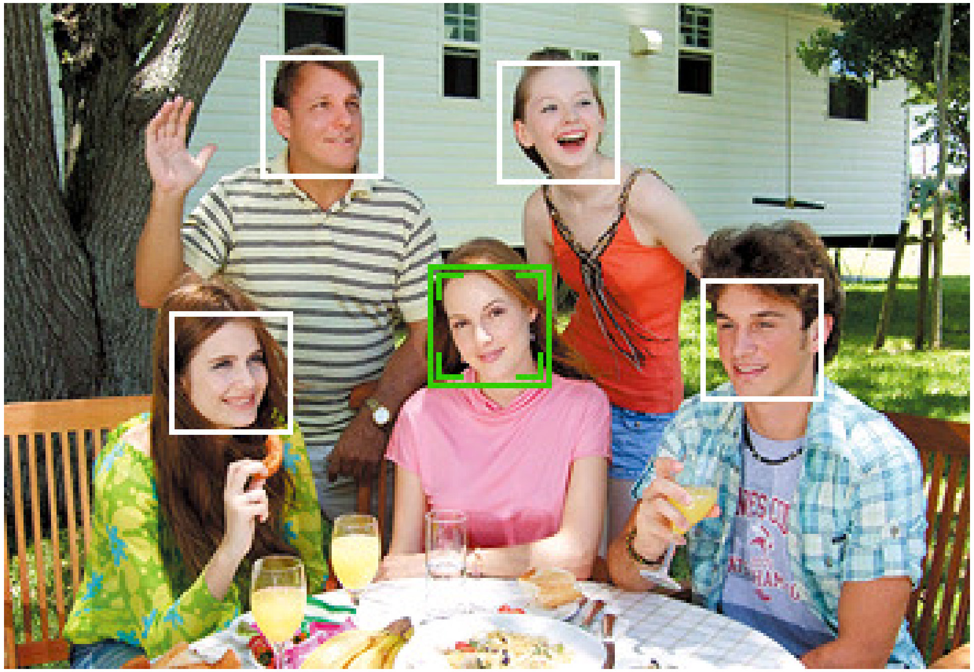
Viola-Jones Type Face Detection



lecturer: Jiří Matas, matas@cmp.felk.cvut.cz

authors: Jiří Matas, Ondřej Drbohlav

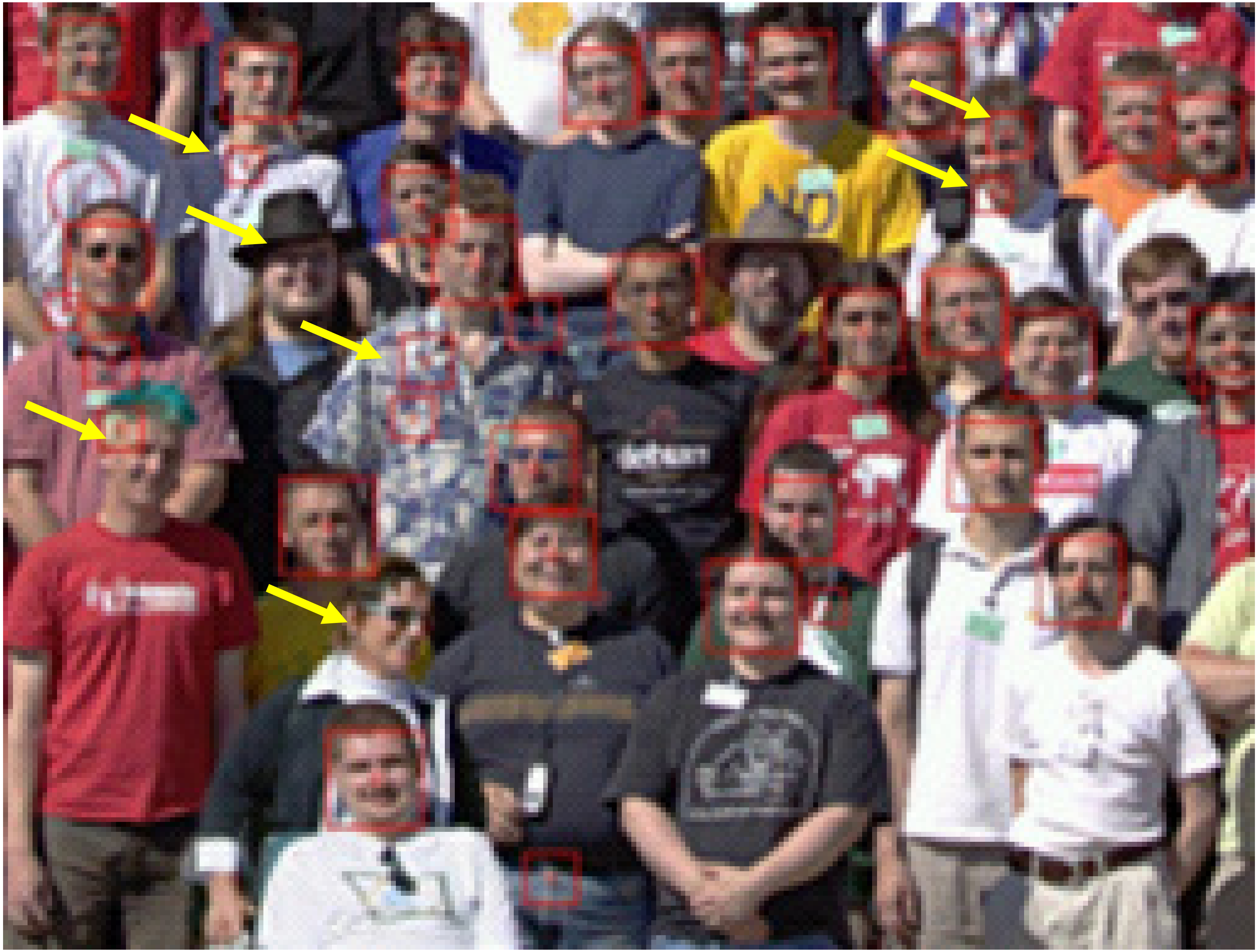
Czech Technical University, Faculty of Electrical Engineering

Department of Cybernetics, Center for Machine Perception

16/May/2016

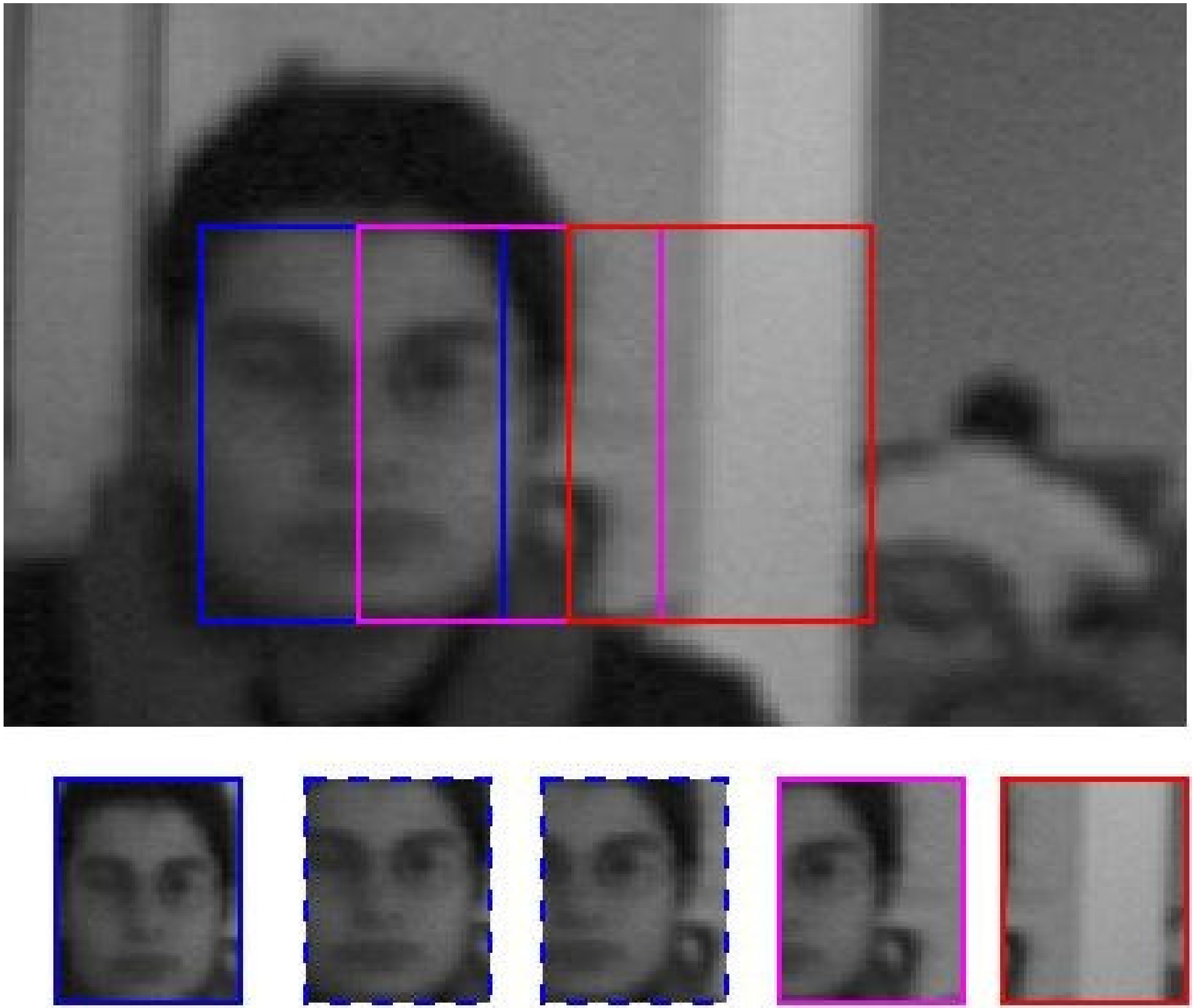
Last update: 15/May/2016

The task in not simple



CENTER FOR MACHINE PERCEPTION

What is/is not a face?



face = 1 | face = ? | face = ? | face = ? | face = 0

Face Detection Task Formulation

Typically,

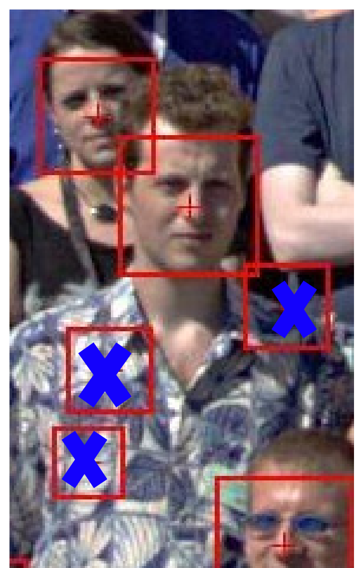
- The user supplies L_o rectangular sub-windows with faces and L_n rectangular sub-windows with non-faces
- The task of the detection system is to output a tight rectangular bounding box (BB) around the detected face
- The BB is considered a good detection when:
  - Eyes, mouth are inside the BB ($=$ face inside BB)
  - Between-eye distance is bigger than 0.5 x larger side of BB ($=$ BB is tight)

Errors

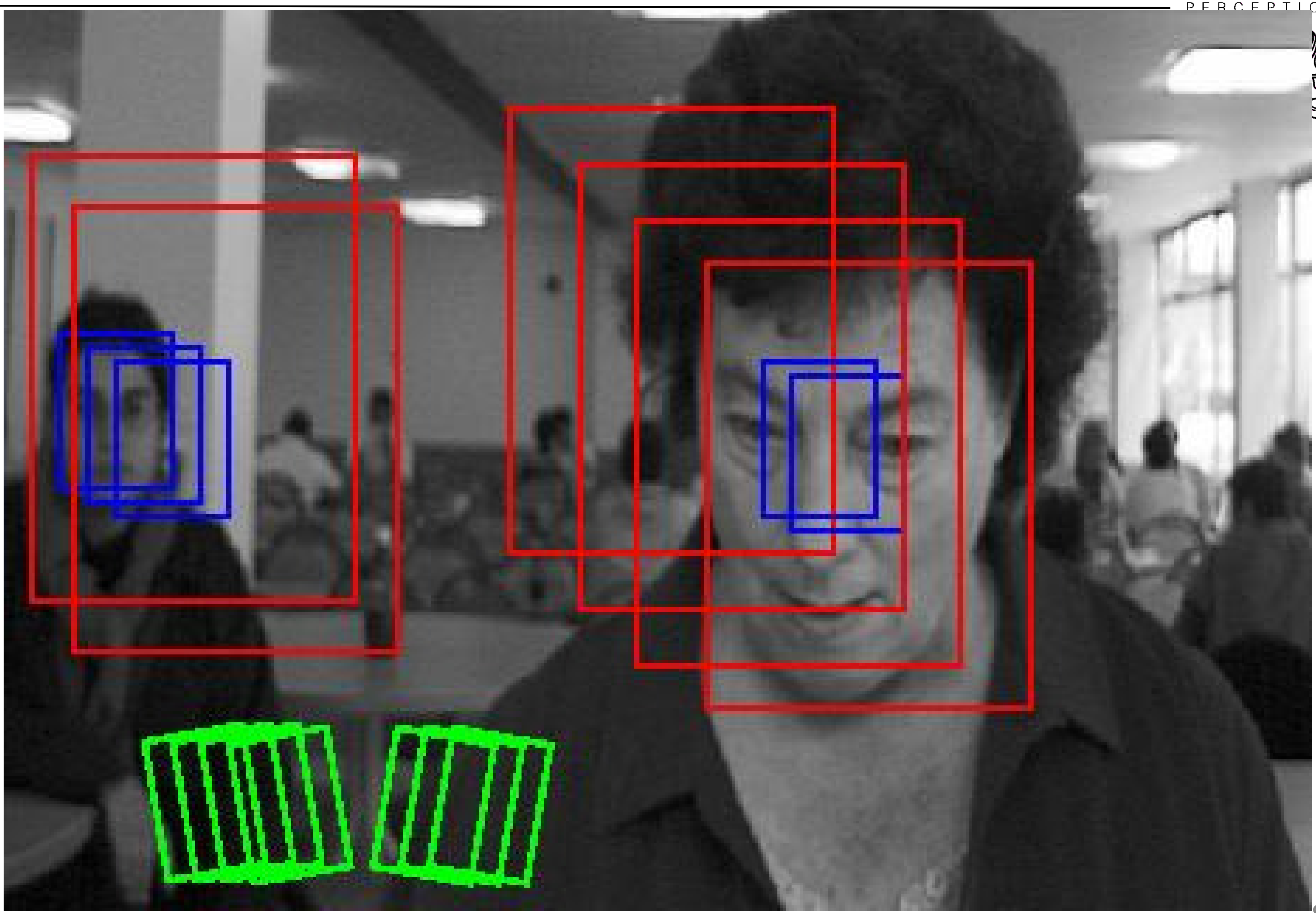
1. false negative
2. false positives
3. localization

Problems:

- where is the border between 1. + 2. and 3. ?



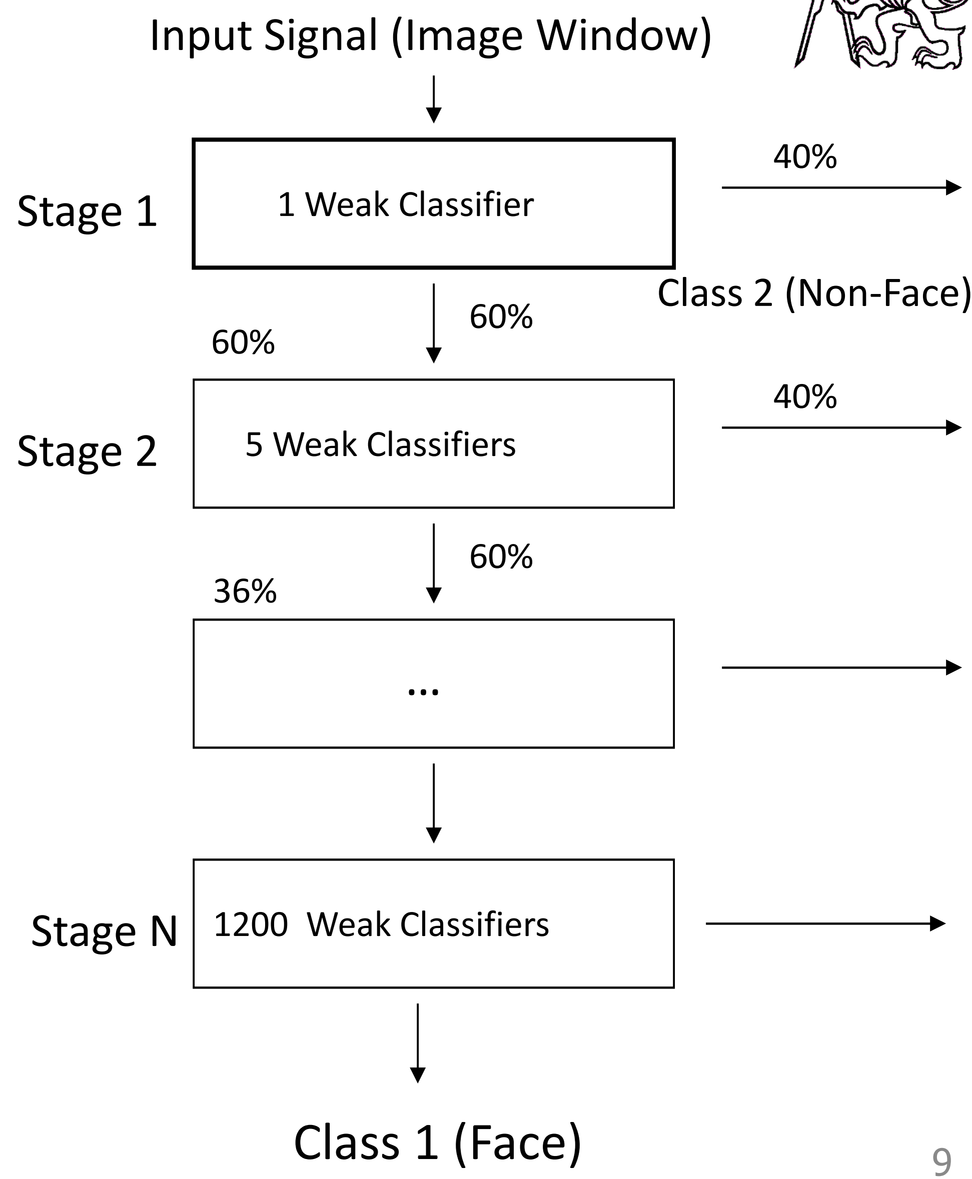
Viola and Jones Suggested a Brute-Force Search



Viola – Jones (2001)

Breakthrough #1

- Speed depends on negative examples only
- This is addressed by sequential decision making

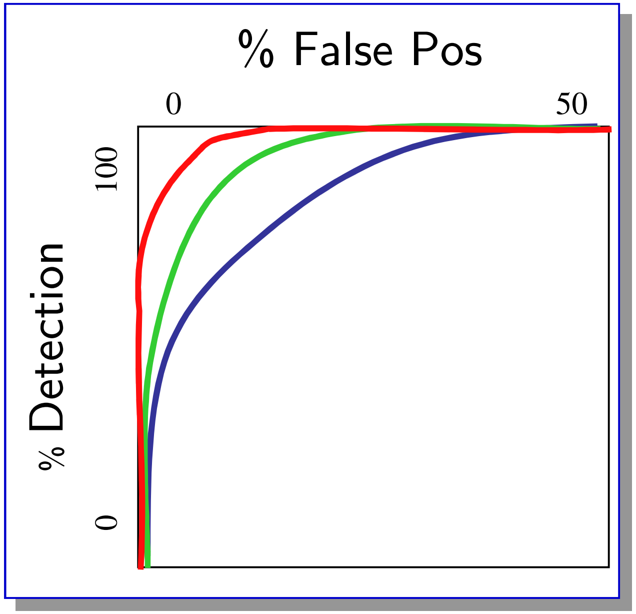


Viola – Jones (2001)

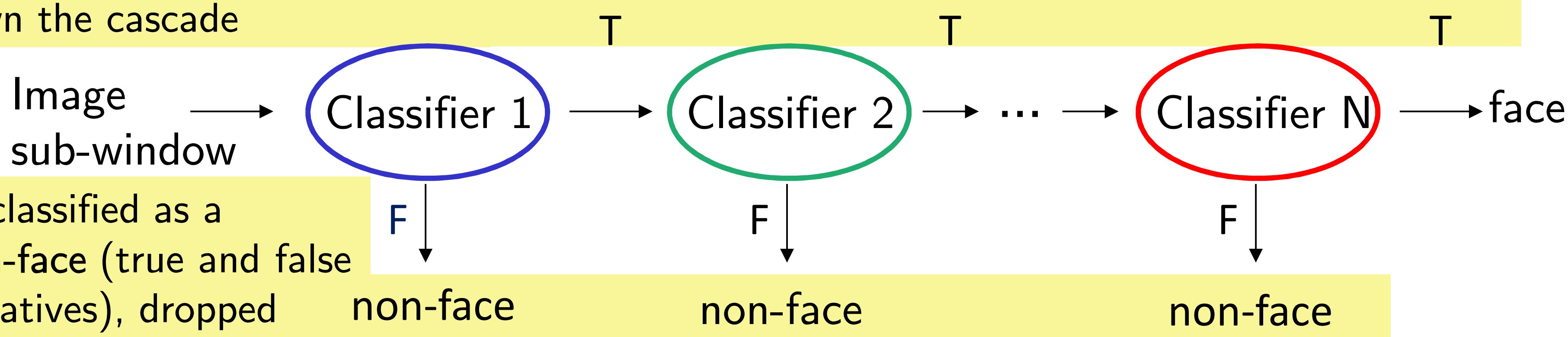
Breakthrough #2 – bootstrap

- Classifiers further down the cascade are trained on samples which had **not** been already discarded by previous classifiers (by being classified as a non-face). Their training set is thus **harder** but orders of magnitude **smaller** than training set faced by preceding classifiers. The classifiers thus can be designed to be increasingly **more complex**.

Receiver operating characteristic (ROC)



T: classified as a **face** (true positives and false positives), goes further down the cascade

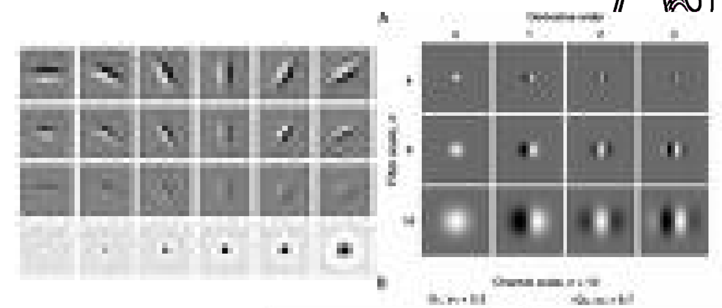


F: classified as a **non-face** (true and false negatives), dropped from the pipeline

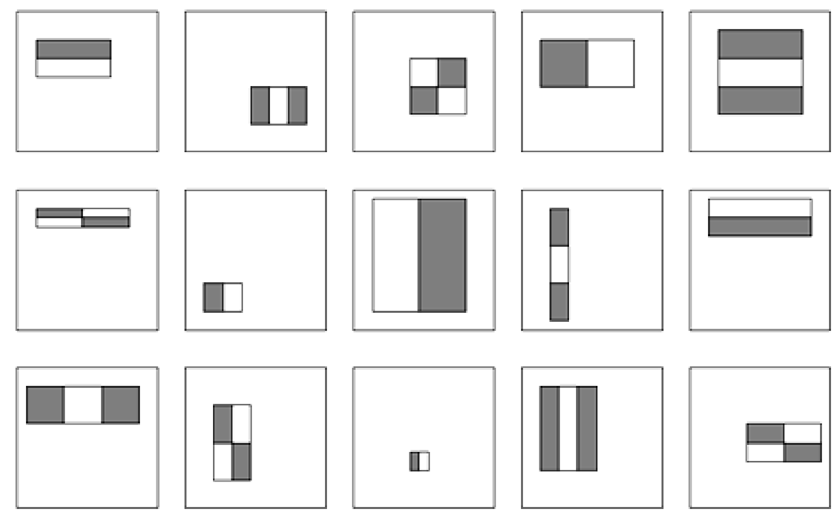
Viola – Jones (2001)

Breakthrough #3 : Fast features

- Gabor filters had been commonly used as features of choice; they are nice but expensive to compute.



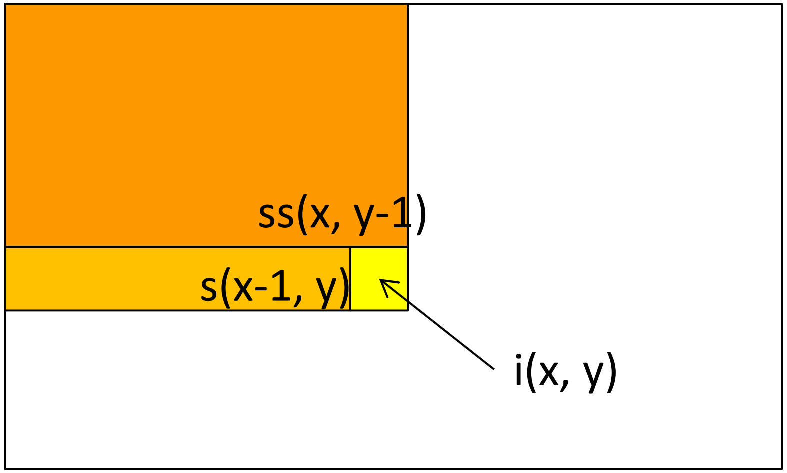
- Viola - Jones have approximated Gabors by piecewise constant functions - Haar wavelets.



Example:

$$\psi(t) = \begin{cases} 1 & \text{for} \quad 0 \le t < 0.5 \\ -1 & \text{for} \quad 0.5 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

Fast Calculation of Haar Wavelets



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

integral image : $ss(x, y) = ss(x, y-1) + s(x, y)$

MATLAB: ss = cumsum(cumsum(double(i)), 2);

Image

0	1	1	1
1	2	2	3
1	2	1	1
1	3	1	0

Integral image

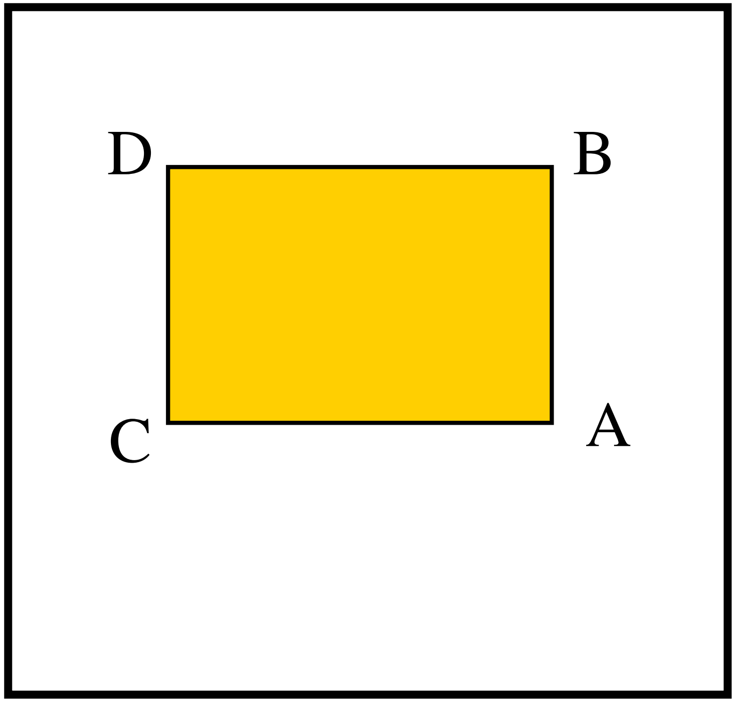
0	1	2	3
1	4	7	11
2	7	11	16
3	11	16	21

Fast Calculation of Haar Wavelets

- values at A,B,C,D are read out form the integral image
- Sum of the intensities within the rectangle is equal to:

$$\text{sum} = A - B - C + D$$

- Each rectangle requires 3 addtion/subtraction operations!



Viola – Jones (2001)

CENTER FOR MACHINE PERCEPTION

Breakthrough #4

- VJ have employed AdaBoost (Schapire a Freund, 1997) which both trains the classifier and selects the features
- Pros of Adaboost:
  - Well understood
  - Good detection rate (in many applications)
  - Easy to implement ("just 10 lines of code" [R. Schapire])

AdaBoost: Algorithm

Input: $(x_1, y_1), \ldots, (x_L, y_L)$, where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$

Initialize weights $D_1(i) = 1/L$.

For $t = 1, ..., T$:

- Find $h_t = \arg\min_{h \in \mathcal{B}} \epsilon_t; \quad \epsilon_t = \sum_{i=1}^{L} D_t(i) [\![y_i \neq h(x_i)]\!]$ (*WeakLearn*)

  $$[\![\text{true}]\!] \stackrel{\text{def}}{=} 1, \quad [\![\text{false}]\!] \stackrel{\text{def}}{=} 0$$

- If $\epsilon_t \geq 1/2$ then stop
- Set $\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- Update

Weights of incorrectly classified training examples are increased such that the next classifier puts more focus on these.

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}, \qquad Z_t = \sum_{i=1}^{L} D_t(i) e^{-\alpha_t y_i h_t(x_i)},$$

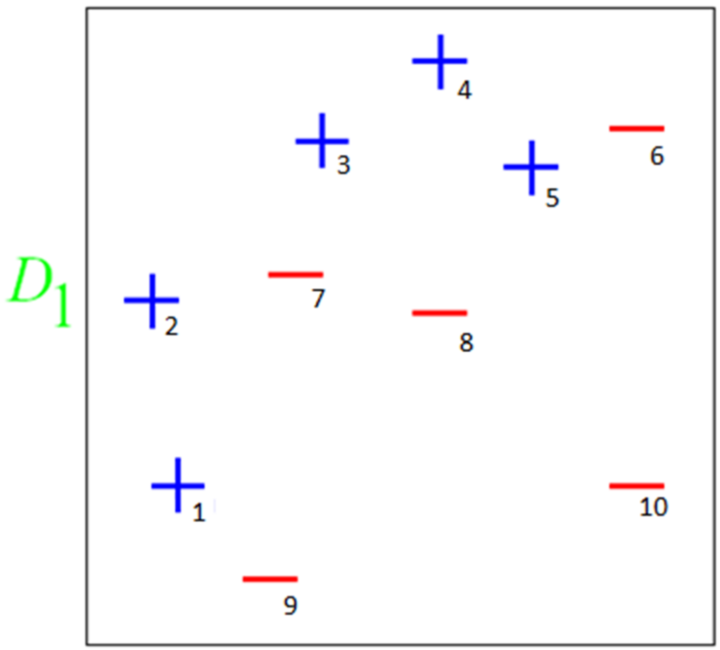
where Z_t is a normalization factor chosen so that D_{t+1} is a distribution.

Output the final classifier:

$$H(x) = \text{sign}(f(x)), \qquad f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

AdaBoost – Example 1

Data	1	2	3	4	5	6	7	8	9	10
Class	+	+	+	+	+	-	-	-	-	-
D_1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1



Taken from “**A Tutorial on Boosting” by** Yoav Freund and Rob Schapire

Example 1 – Iteration 1

Data	1	2	3	4	5	6	7	8	9	10
Class	+	+	+	+	+	-	-	-	-	-
$D_2 \cdot Z_2$	0.07	0.07	0.15	0.15	0.15	0.07	0.07	0.07	0.07	0.07

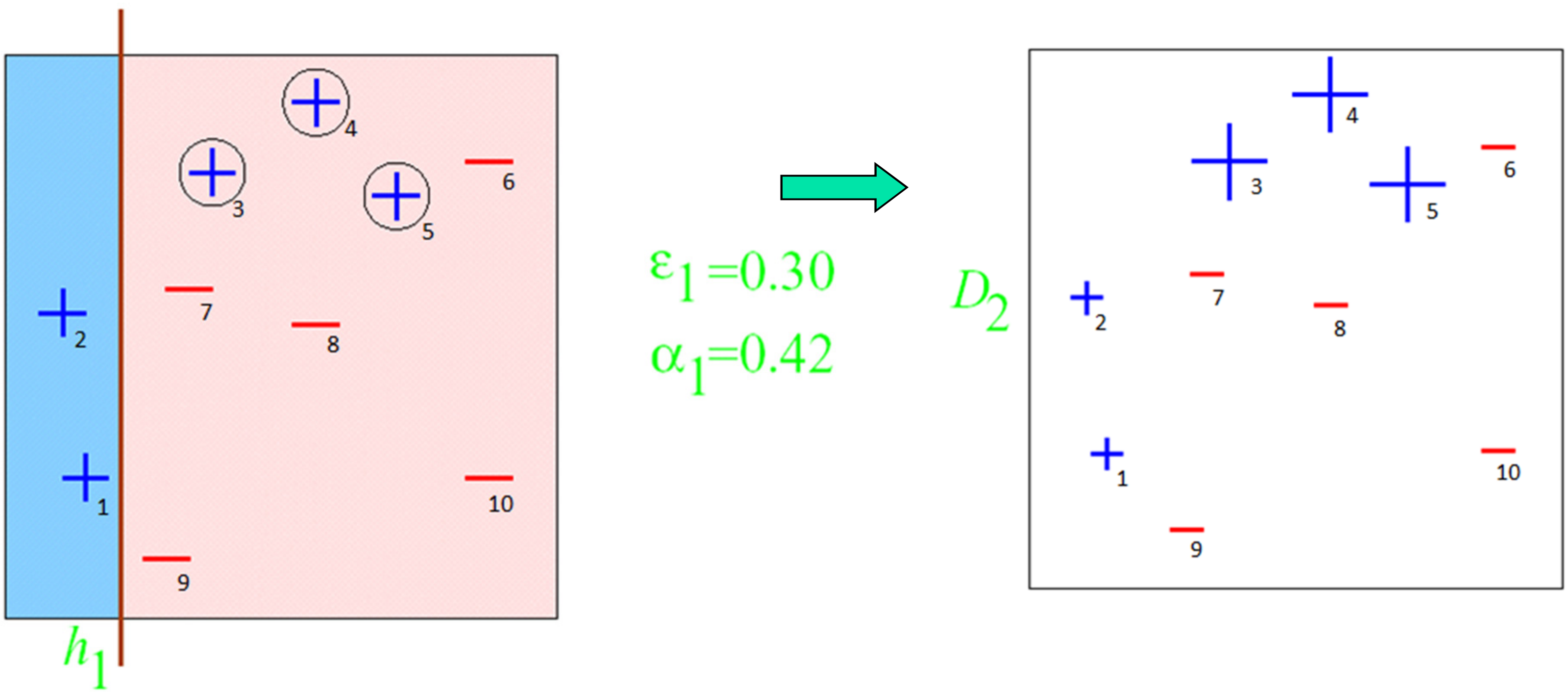
$Z_2 = 0.92$

ϵ_1 ... error

$\alpha_1 = 1/2 \log(1-\epsilon_1)/\epsilon_1$

$D_2 \approx D_1 \sqrt{\epsilon_1/(1-\epsilon_1)}$ for corr. class.

$D_2 \approx D_1 \sqrt{(1-\epsilon_1)/\epsilon_1}$ for wrongly class.



Example 1 – Iteration 2

Data	1	2	3	4	5	6	7	8	9	10
Class	+	+	+	+	+	-	-	-	-	-
$D_3 \cdot Z_3$	0.04	0.04	0.09	0.09	0.09	0.04	0.14	0.14	0.14	0.04

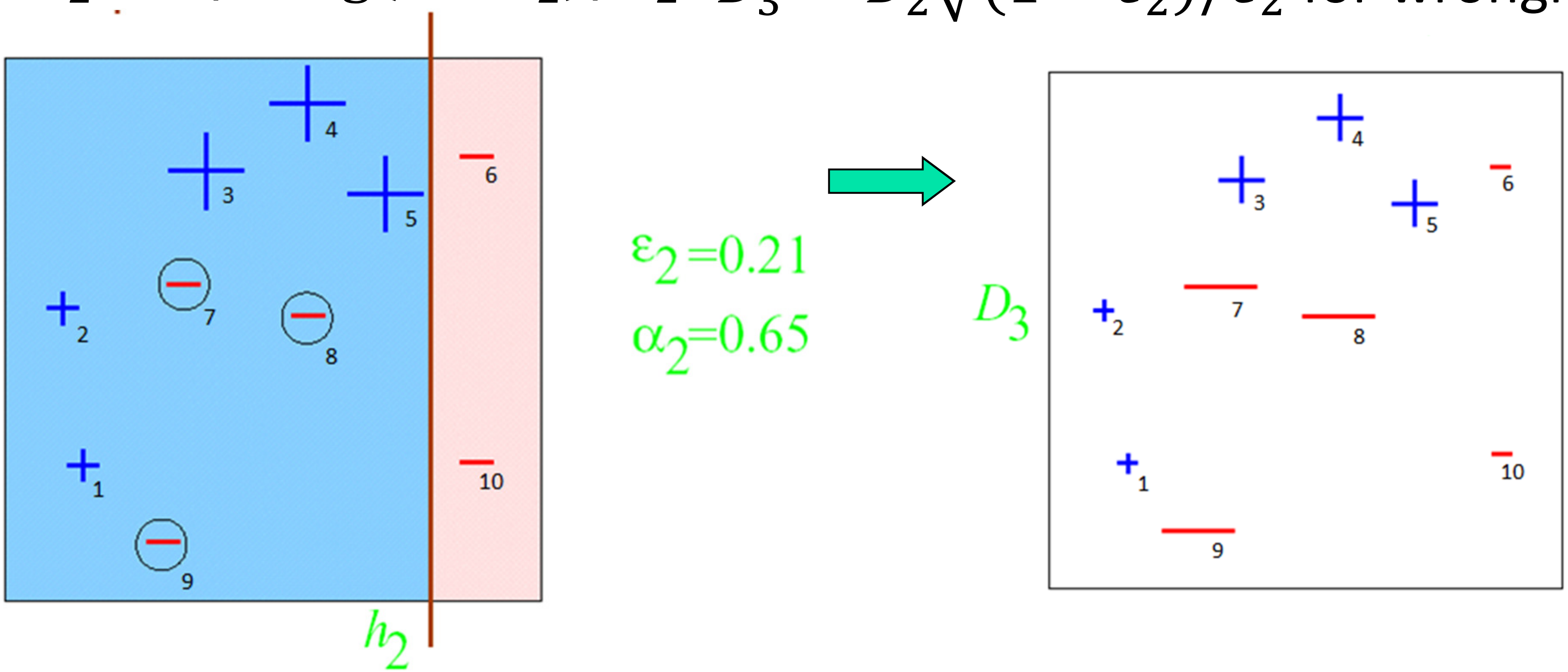
$Z_3 = 0.82$

ϵ_2 ... error

$\alpha_2 = 1/2 \log(1-\epsilon_2)/\epsilon_2$

$D_3 \approx D_2\sqrt{\epsilon_2/(1-\epsilon_2)}$ for corr. class.

$D_3 \approx D_2\sqrt{(1-\epsilon_2)/\epsilon_2}$ for wrongly class.



Example 1 – Iteration 3

Data	1	2	3	4	5	6	7	8	9	10
Class	+	+	+	+	+	-	-	-	-	-
$D_4 \cdot Z_4$	0.11	0.11	0.04	0.04	0.04	0.11	0.07	0.07	0.07	0.02

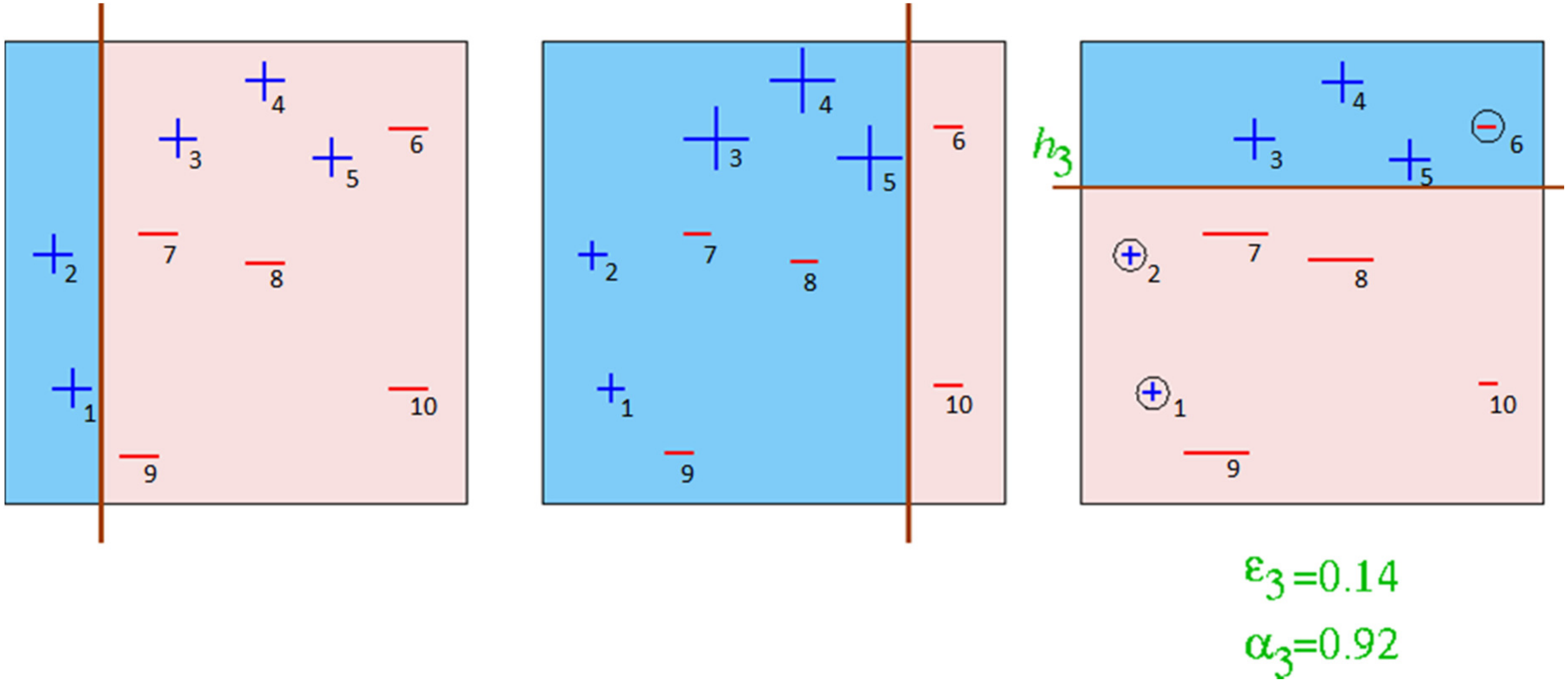
$Z_4 = 0.68$

ϵ_3 ... error

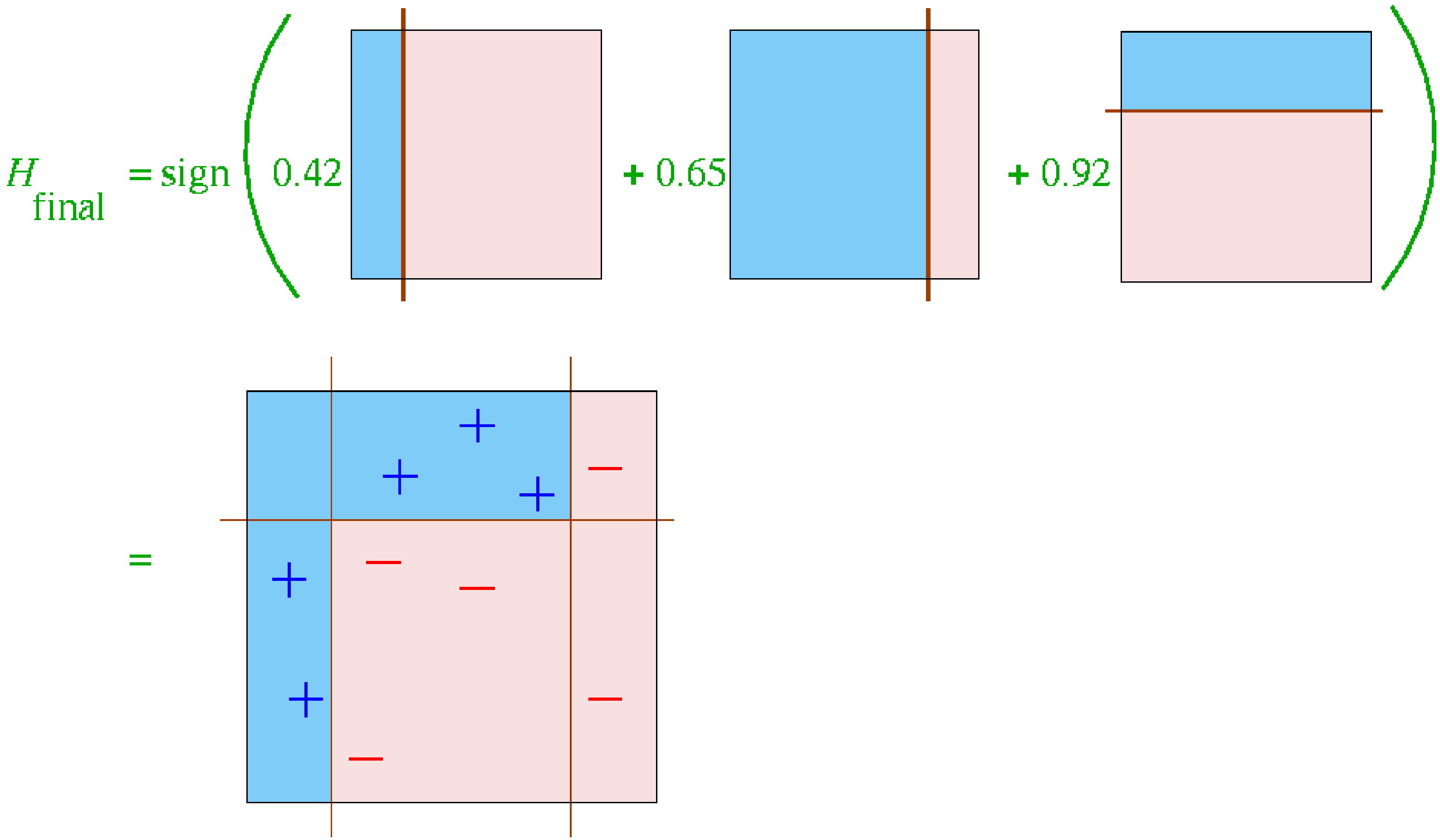
$\alpha_3 = 1/2 \log(1-\epsilon_3)/\epsilon_3$

$D_4 \approx D_3 \sqrt{\epsilon_3/(1-\epsilon_3)}$ for corr. class.

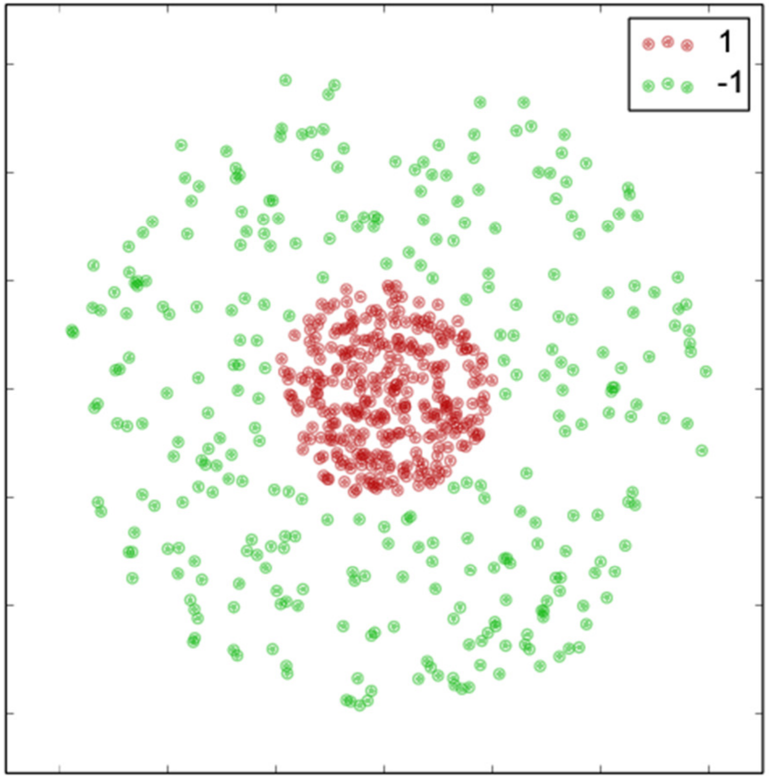
$D_4 \approx D_3 \sqrt{(1-\epsilon_3)/\epsilon_3}$ for wrongly class.



Example 1 – Final Classifier after Iter. 3



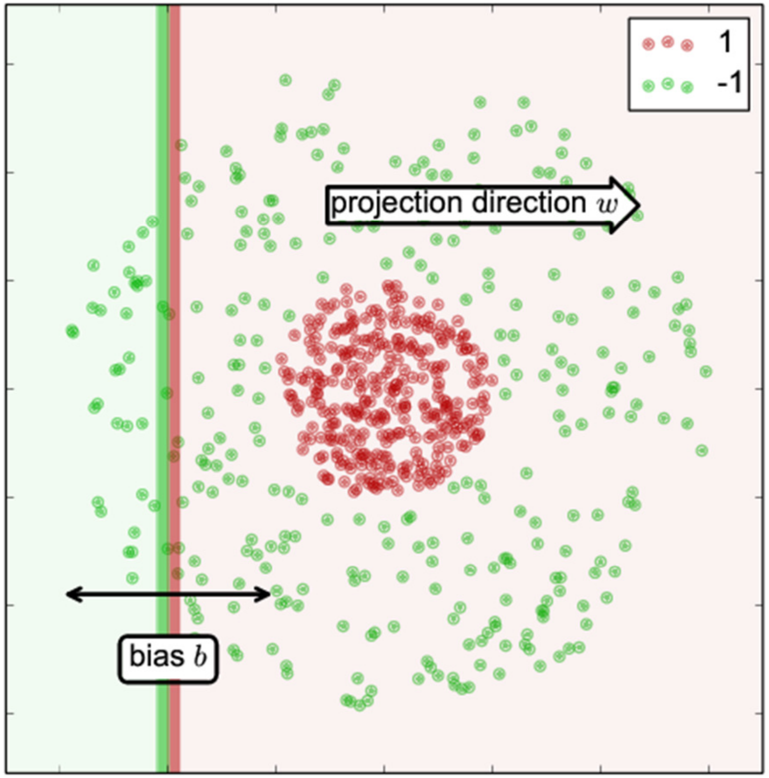
AdaBoost – Example 2



Dataset: $(x_1, y_1), \ldots, (x_L, y_L)$, where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$.

The two class distributions do not overlap (Bayes error is 0). The class distributions are not known to AdaBoost.

Example 2, Weak Classifier



Dataset: $(x_1, y_1), \ldots, (x_L, y_L)$, where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$.

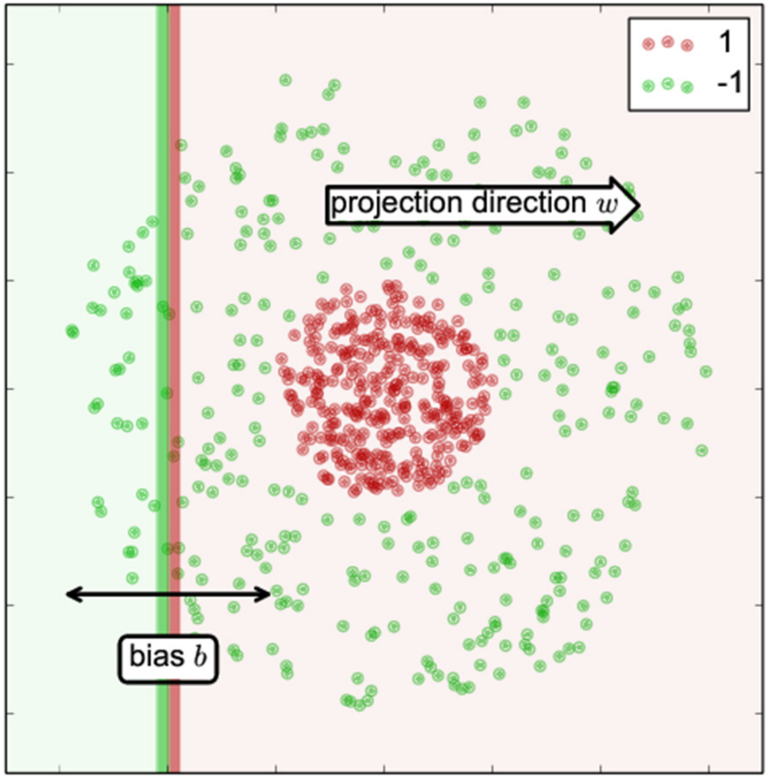
The two class distributions do not overlap (Bayes error is 0). The class distributions are not known to AdaBoost.

Weak classifier: a linear classifier

$$h_{w,b}(x) = \text{sign}(w \cdot x + b) \,,$$

where w is the projection direction vector and b is the bias.

Example 2, Weak Classifier Set



Dataset: $(x_1, y_1), \dots, (x_L, y_L)$, where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$.

The two class distributions do not overlap (Bayes error is 0). The class distributions are not known to AdaBoost.

Weak classifier: a linear classifier

$$h_{w,b}(x) = \mathrm{sign}(w \cdot x + b)\,,$$

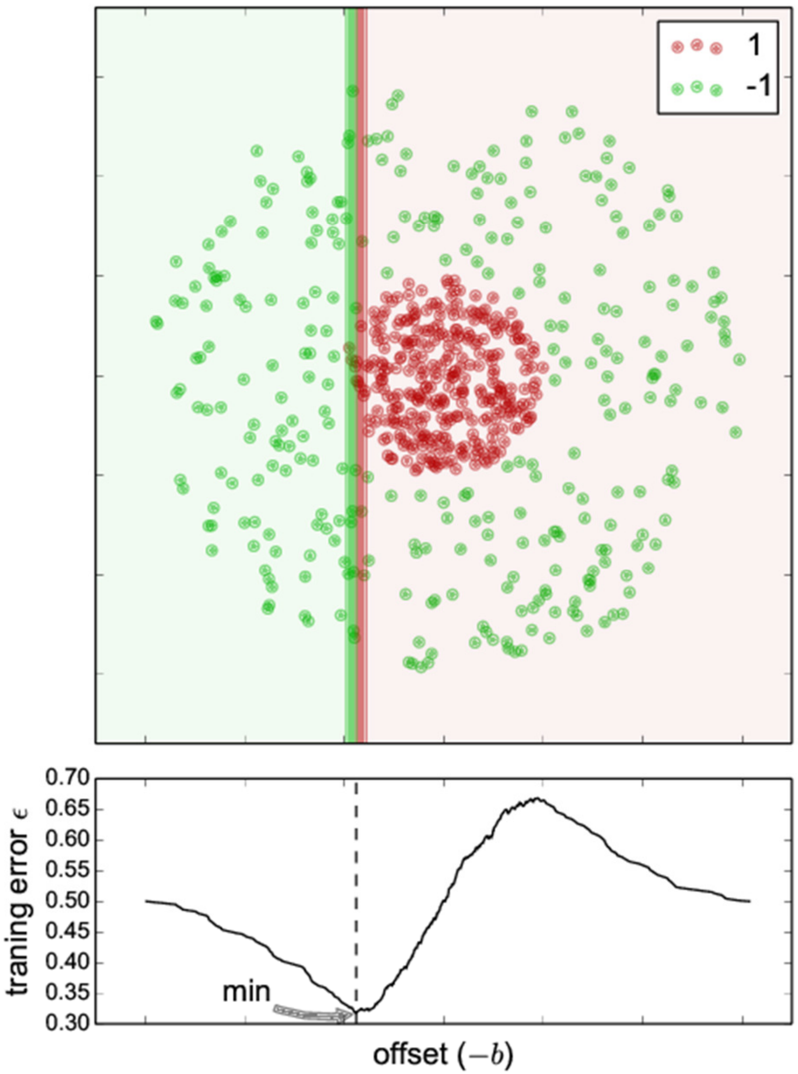
where w is the projection direction vector and b is the bias.

Weak classifier set $\mathcal{B}$:

$$\{h_{w,b} \mid w \in \{w_1, w_2, ..., w_N\}, b \in \mathbb{R}\}$$

- N is the number of projection directions used

Example 2, Weak Classifier Set



Dataset: $(x_1, y_1), \ldots, (x_L, y_L)$, where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$.

The two class distributions do not overlap (Bayes error is 0). The class distributions are not known to AdaBoost.

Weak classifier: a linear classifier

$$h_{w,b}(x) = \operatorname{sign}(w \cdot x + b)\,,$$

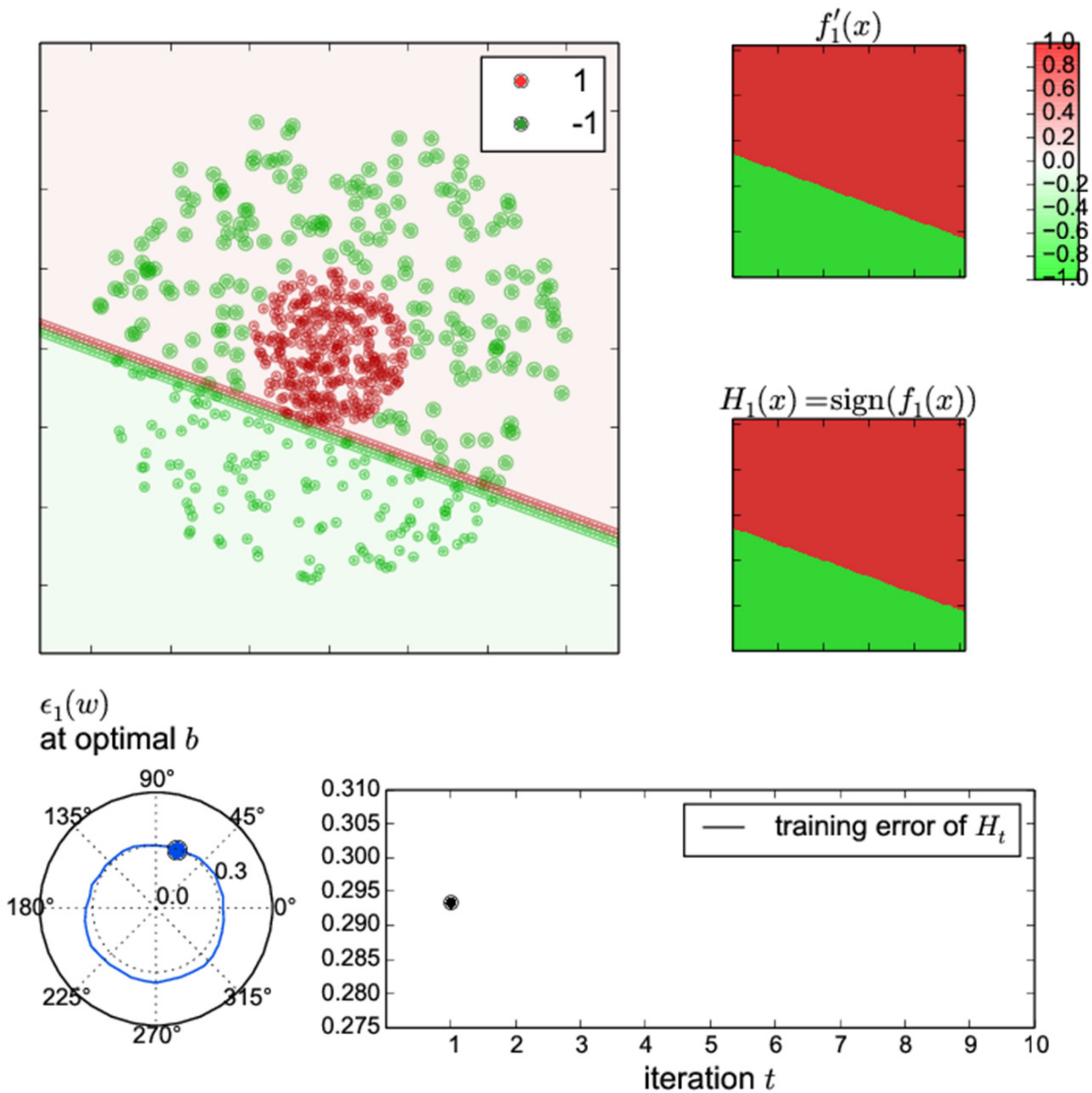
where w is the projection direction vector and b is the bias.

Weak classifier set $\mathcal{B}$:

$$\{h_{w,b} \mid w \in \{w_1, w_2, ..., w_N\}, b \in \mathbb{R}\}$$

- N is the number of projection directions used
- for each projection direction w, varying bias b results in different training errors ϵ.

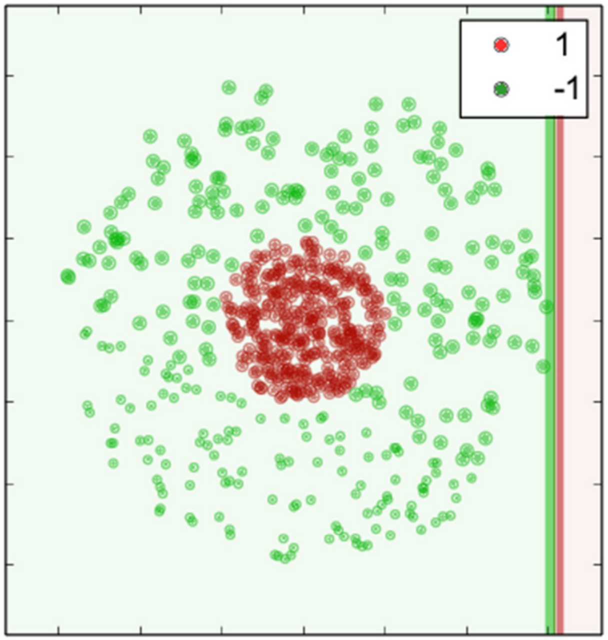
Example 2, Iteration 1

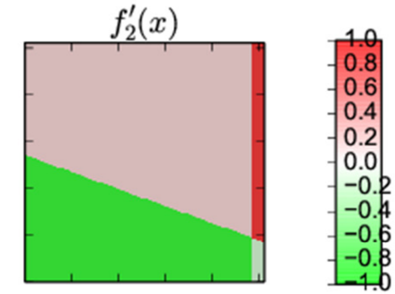


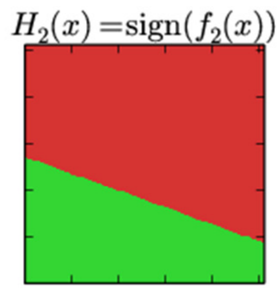
$t = 1$

- h_1 selected (note ● in the polar plot). $N = 36$ directions w are used.
- $\epsilon_1 < 0.5$, continue
- $\alpha_1 = \frac{1}{2}\log(\frac{1-\epsilon_1}{\epsilon_1})$
- re-weighting D puts more weight to mis-classified samples in the ● class
- $f_1(x) = \alpha_1 h_1(x)$
- $f_1'(x) = f_1(x)/\alpha_1$
- $H_1(x) = \mathrm{sign}(f_1(x))$

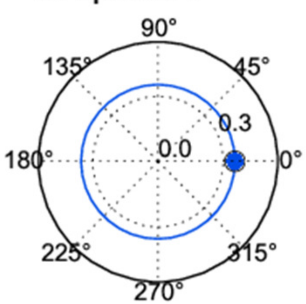
Example 2, Iteration 2

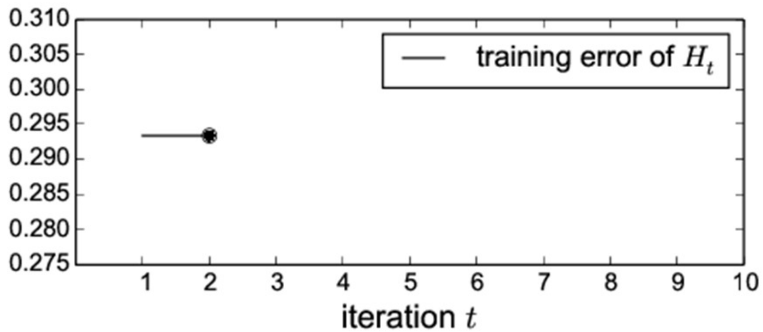






$\epsilon_2(w)$
at optimal b





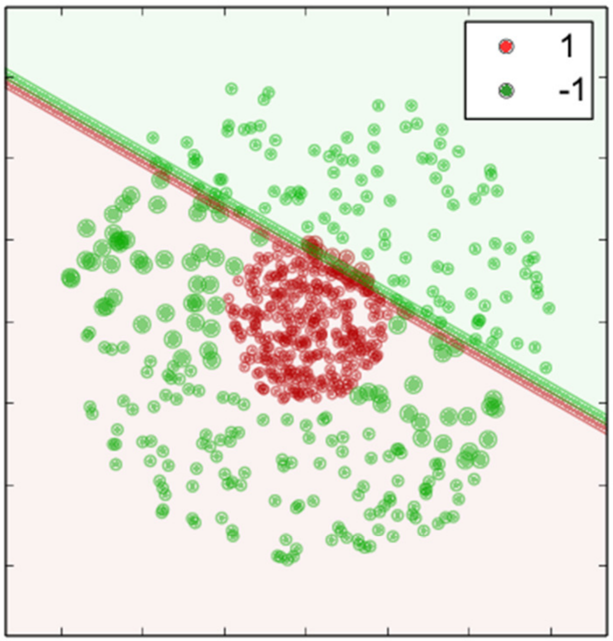
$t = 2$

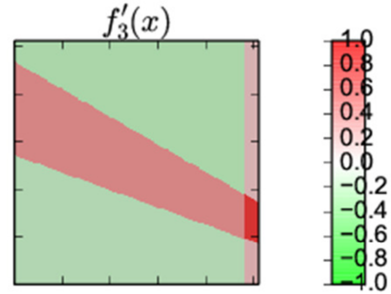
- minimum errors $\epsilon_2(w)$ for all weak classifier directions w are equal. Everything is classified as -1 (●)
- this essentially re-weights the classes; gives more weight to class 1 (●)
- $f_2(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x)$
- $f_2'(x) = \frac{f_2(x)}{\alpha_1 + \alpha_2}$
- $H_2(x) = \mathrm{sign}(f_2(x))$

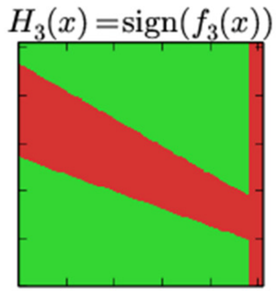
Quiz question:

- What is the difference between $f_2(x)$ and the previous $f_1(x)$? (Note that all points are classified as -1)

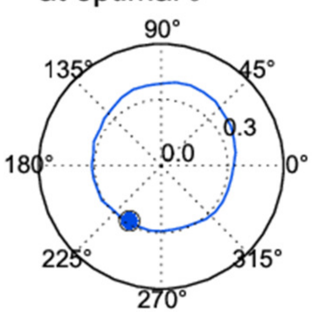
Example 2, Iteration 3

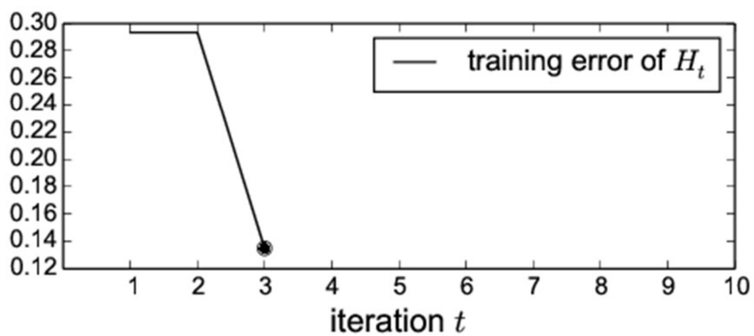






$\epsilon_3(w)$
at optimal b

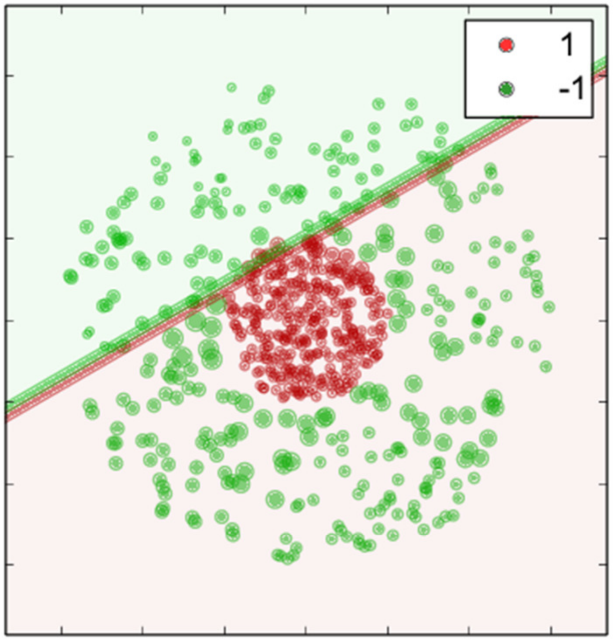


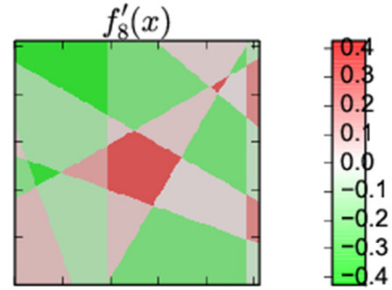


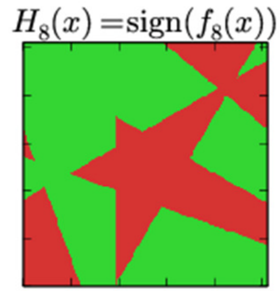
$t = 3$

- h_3 selected which minimizes $\epsilon_3(w)$ (note ● in the polar plot).
- $\alpha_3 = \frac{1}{2}\log(\frac{1-\epsilon_3}{\epsilon_3})$
- distribution re-weighted
- $f_3(x) = \sum_{q=1}^{3} \alpha_q h_q(x)$
- $f_3'(x) = \frac{f(x)}{\sum_{q=1}^{3} \alpha_q}$
- $H_3(x) = \text{sign}(f_3(x))$

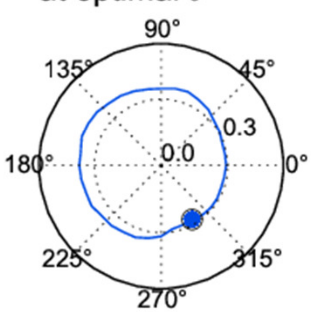
Example 2, Iteration 8

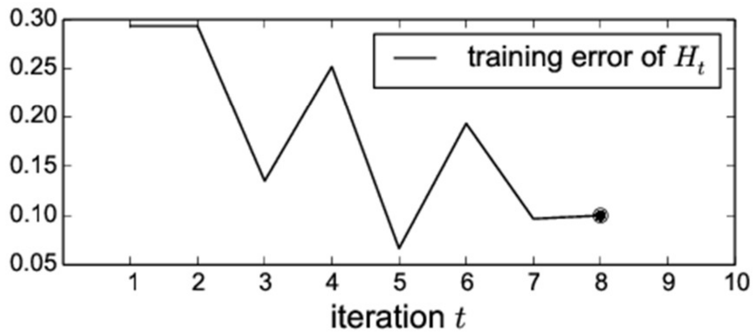






$\epsilon_8(w)$
at optimal b

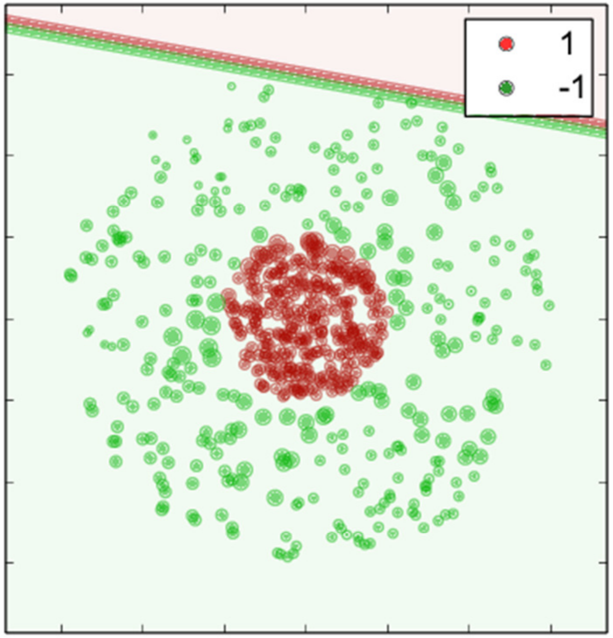


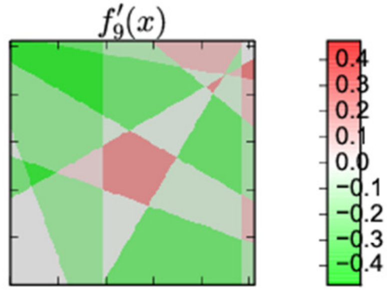


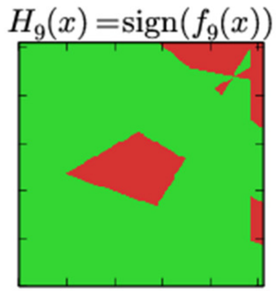
$t = 8$

- h_8 selected which minimizes $\epsilon_8(w)$ (note $\bullet$ in the polar plot).
- $\alpha_8 = \frac{1}{2}\log(\frac{1-\epsilon_8}{\epsilon_8})$
- distribution re-weighted
- $f_8(x) = \sum_{q=1}^{8} \alpha_q h_q(x)$
- $f'_8(x) = \frac{f(x)}{\sum_{q=1}^{8} \alpha_q}$
- $H_8(x) = \text{sign}(f_8(x))$

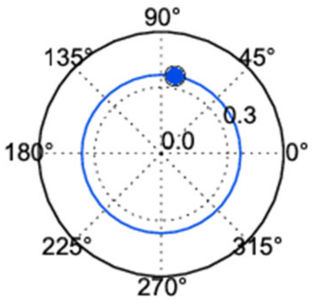
Example 2, Iteration 9

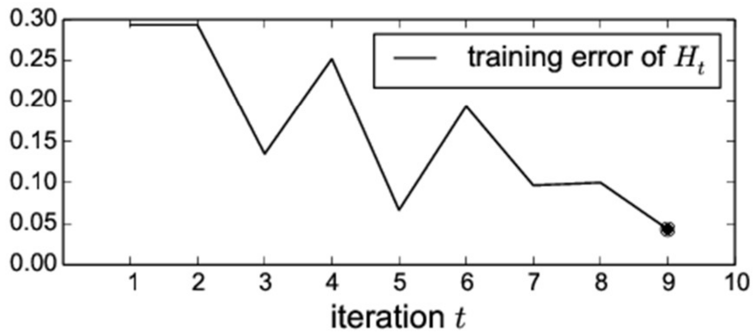






$\epsilon_9(w)$
at optimal b

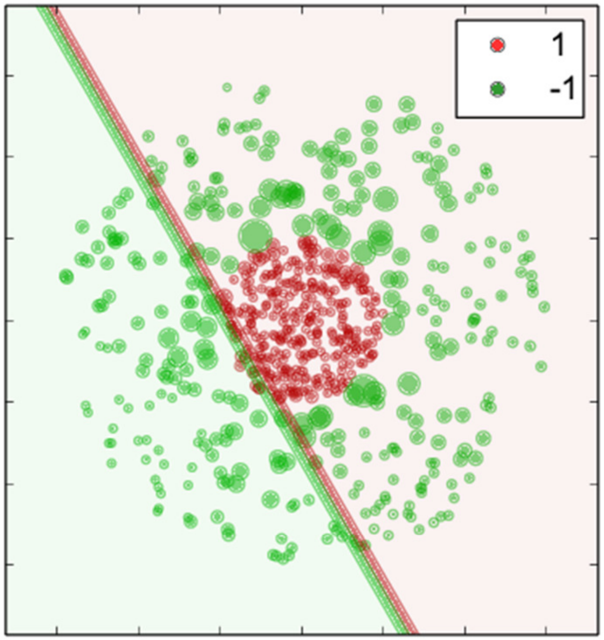


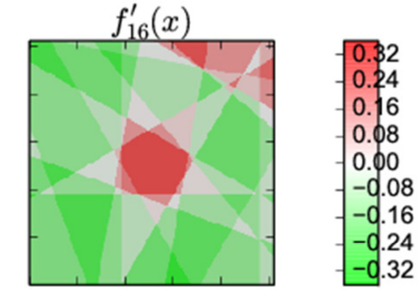


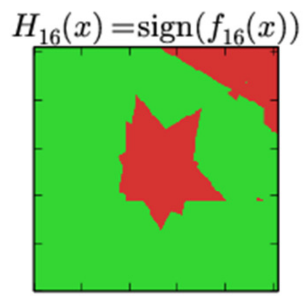
$t = 9$

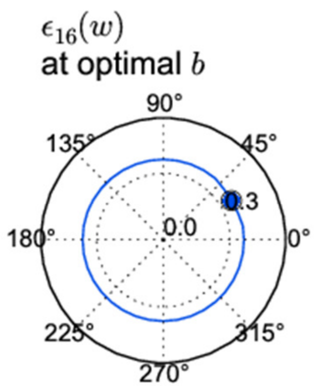
- h_9 selected which minimizes $\epsilon_9(w)$ (note $\bullet$ in the polar plot).
- $\alpha_9 = \frac{1}{2}\log(\frac{1-\epsilon_9}{\epsilon_9})$
- distribution re-weighted
- $f_9(x) = \sum_{q=1}^{9} \alpha_q h_q(x)$
- $f'_9(x) = \frac{f(x)}{\sum_{q=1}^{9} \alpha_q}$
- $H_9(x) = \text{sign}(f_9(x))$

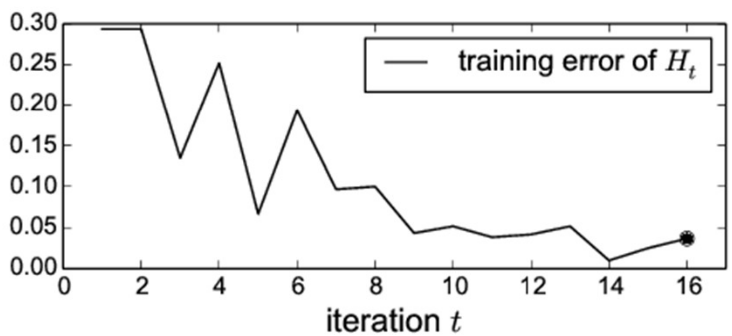
Example 2, Iteration 15







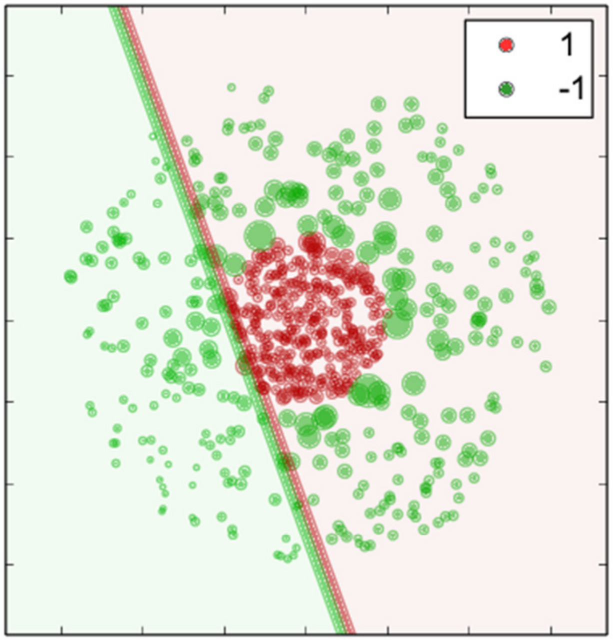


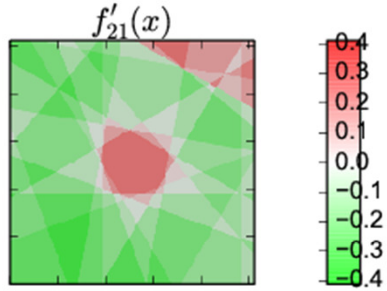


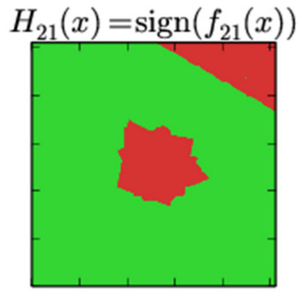
$t = 15$

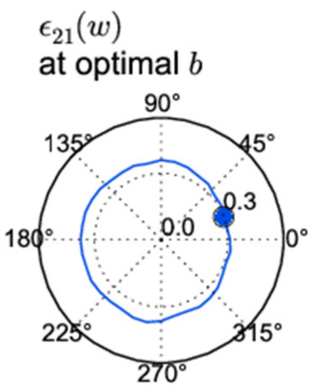
- h_{15} selected which minimizes $\epsilon_{15}(w)$ (note • in the polar plot).
- $\alpha_{15} = \frac{1}{2}\log(\frac{1-\epsilon_{15}}{\epsilon_{15}})$
- distribution re-weighted
- $f_{15}(x) = \sum_{q=1}^{15} \alpha_q h_q(x)$
- $f'_{15}(x) = \frac{f(x)}{\sum_{q=1}^{15} \alpha_q}$
- $H_{15}(x) = \text{sign}(f_{15}(x))$

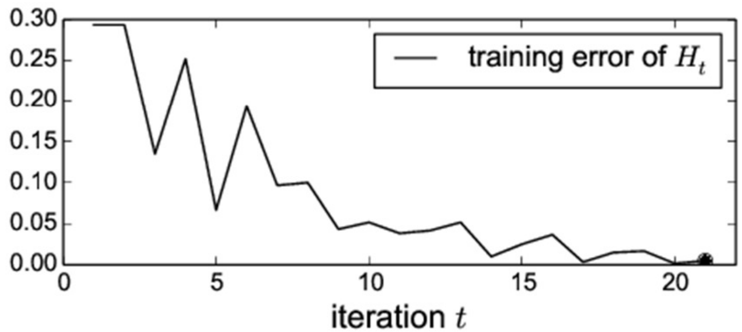
Example 2, Iteration 21







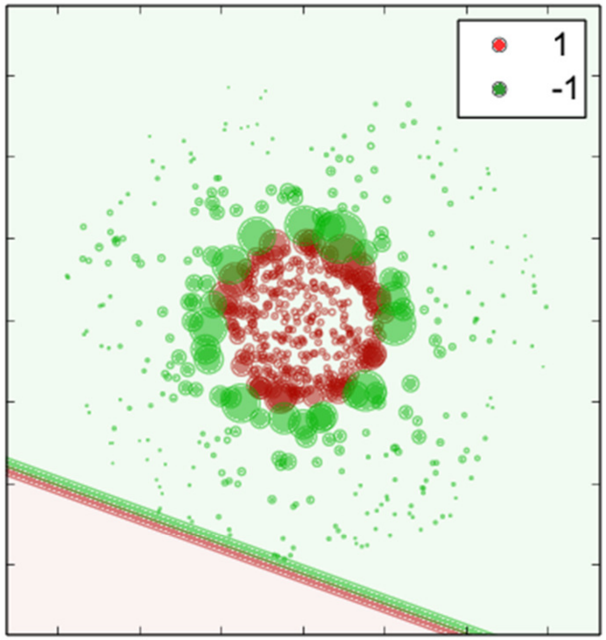


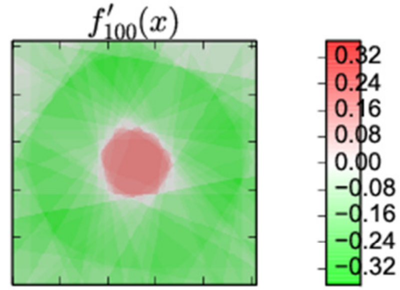


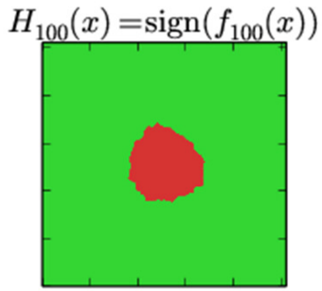
$t = 21$

- h_{21} selected which minimizes $\epsilon_{21}(w)$ (note $\bullet$ in the polar plot).
- $\alpha_{21} = \frac{1}{2}\log(\frac{1-\epsilon_{21}}{\epsilon_{21}})$
- distribution re-weighted
- $f_{21}(x) = \sum_{q=1}^{21} \alpha_q h_q(x)$
- $f'_{21}(x) = \frac{f(x)}{\sum_{q=1}^{21} \alpha_q}$
- $H_{21}(x) = \mathrm{sign}(f_{21}(x))$

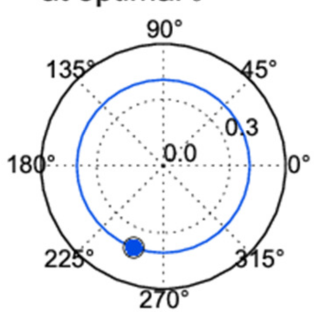
Example 2, Iteration 100

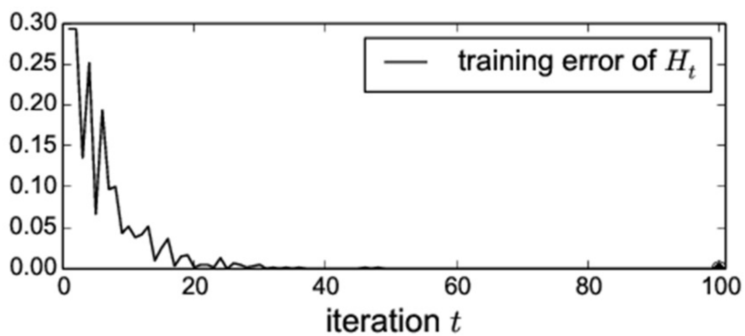






$\epsilon_{100}(w)$
at optimal b

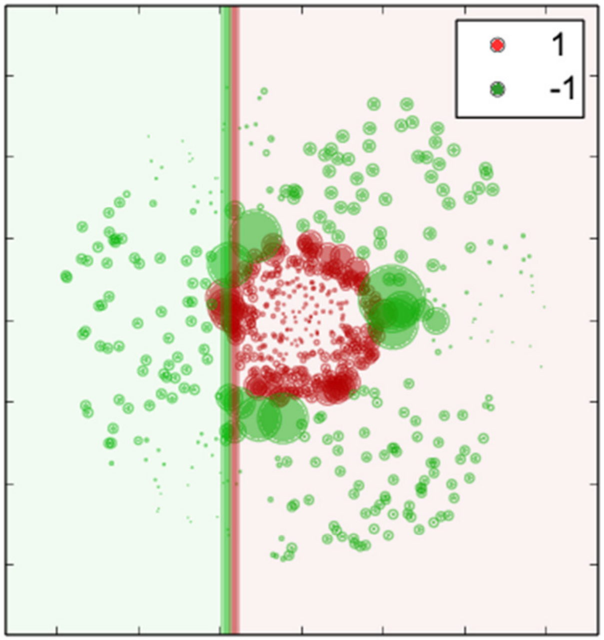


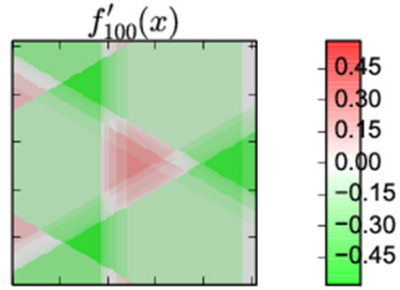


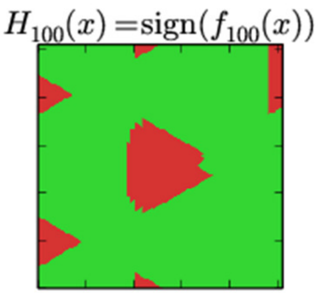
$t = 100$

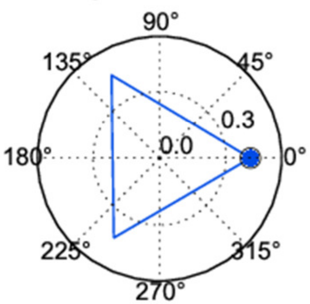
- h_{100} selected which minimizes $\epsilon_{100}(w)$ (note $\bullet$ in the polar plot).
- $\alpha_{100} = \frac{1}{2}\log(\frac{1-\epsilon_{100}}{\epsilon_{100}})$
- distribution re-weighted
- $f_{100}(x) = \sum_{q=1}^{100} \alpha_q h_q(x)$
- $f'_{100}(x) = \frac{f(x)}{\sum_{q=1}^{100} \alpha_q}$
- $H_{100}(x) = \mathrm{sign}(f_{100}(x))$

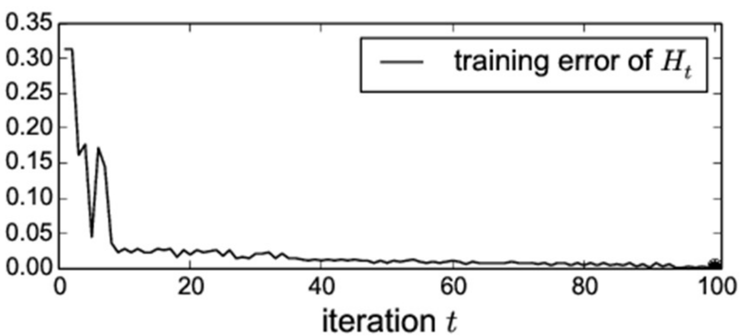
Example 2, *N=3* Proj. Directions, Iteration 100









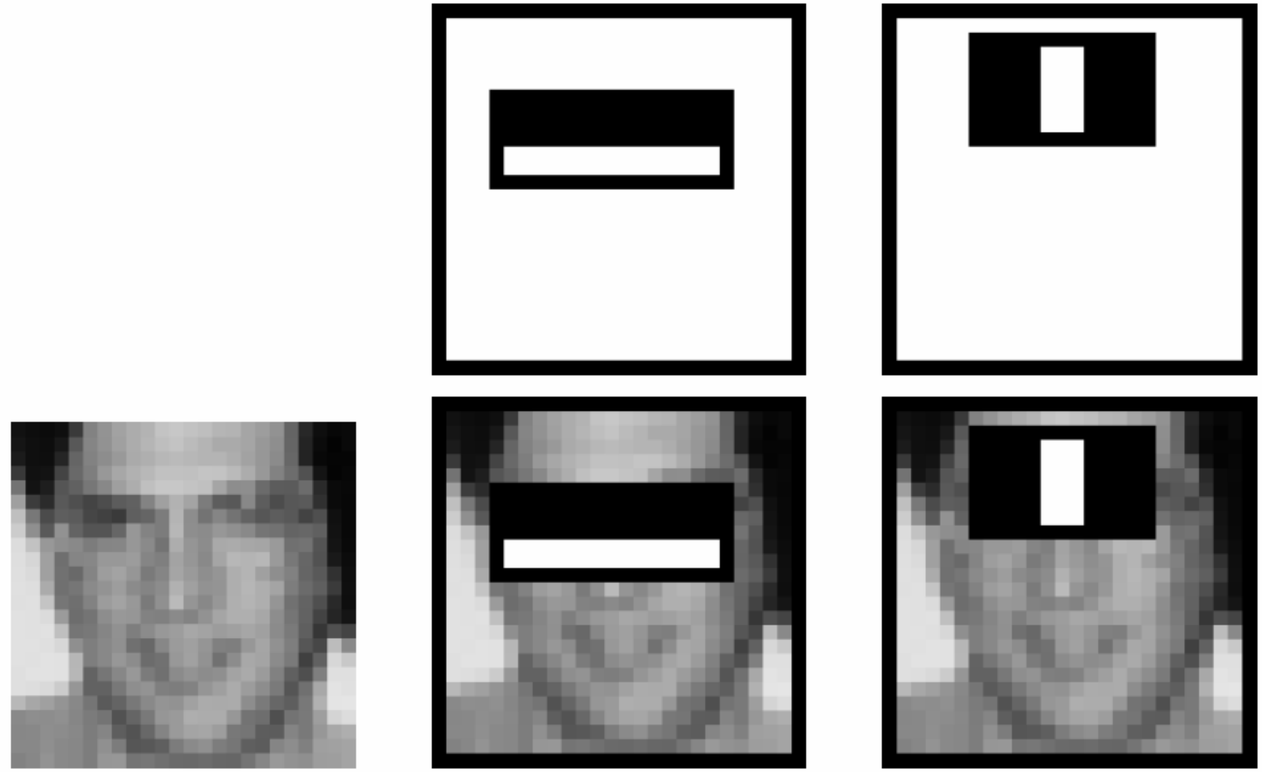


$t = 100$

- h_{100} selected which minimizes $\epsilon_{100}(w)$ (note $\bullet$ in the polar plot).
- $\alpha_{100} = \frac{1}{2}\log(\frac{1-\epsilon_{100}}{\epsilon_{100}})$
- distribution re-weighted
- $f_{100}(x) = \sum_{q=1}^{100} \alpha_q h_q(x)$
- $f'_{100}(x) = \frac{f(x)}{\sum_{q=1}^{100} \alpha_q}$
- $H_{100}(x) = \text{sign}(f_{100}(x))$

Example 3 - Adaboost Detector

- The first two selected classifiers:



The two features have 100% detection rate and 50% false alarm rate

Face Detector, Hard Negative Examples

Images classified as faces by early cascade components

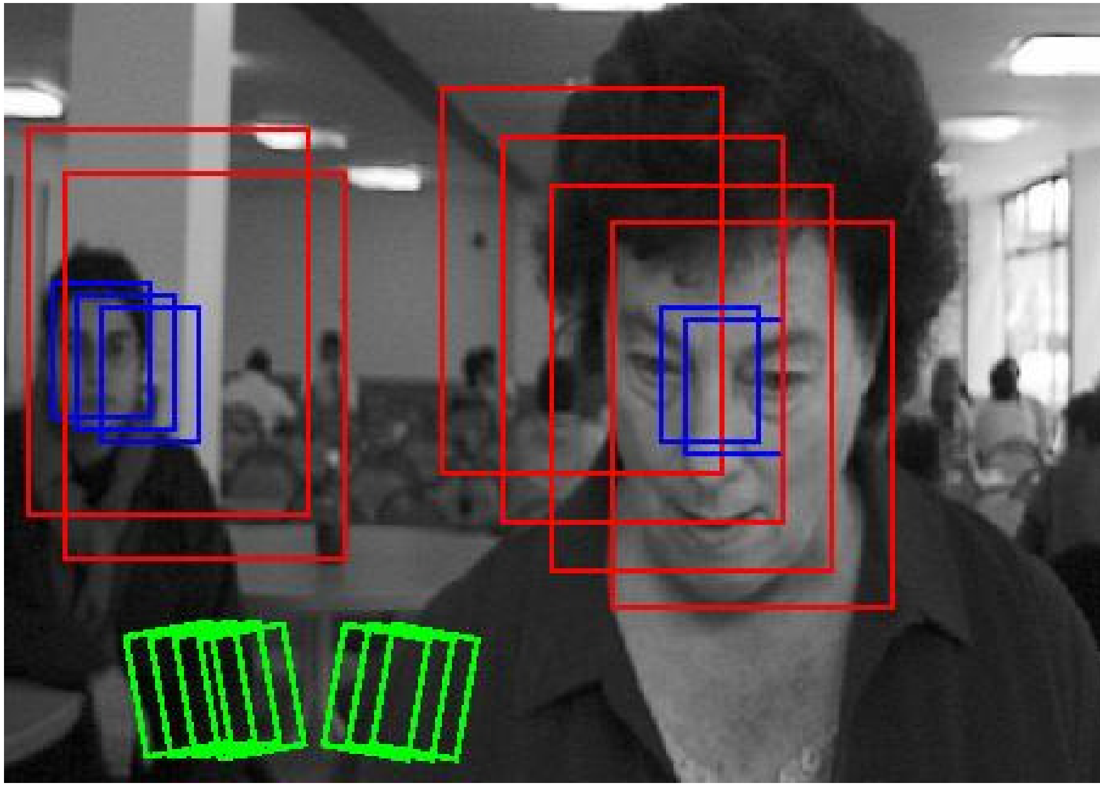
Sampling of Test Window Space

Not every image sub-window must be tested by the classifier.

It is sufficient to use:

- shifts by cca 10% window side
- window side size increments of 15%
- window rotation by +/- 15 deg

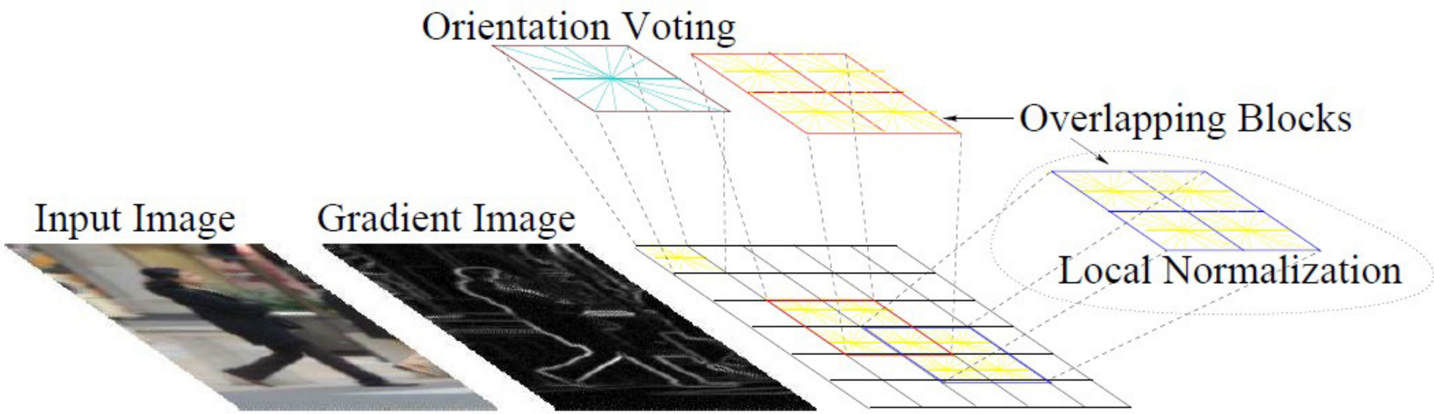
Note:
Total number of sub-windows (thus speed) is determined by the size of smallest face to be detected. Total detection time is the geometric series sum with $q=1/1.15^2$; $s \approx 4t_0$



Historical perspective

- VJ published in 2001.
- Many improvements since then.
- In 2009, implemented in many digital cameras.
- E. g. Waldboost (developed here on CTU) improves the method by addressing the problem of trade-off between speed and accuracy of the Adaboost classifier.

Histogram of Oriented Gradients (HOG) as a Weak Feature



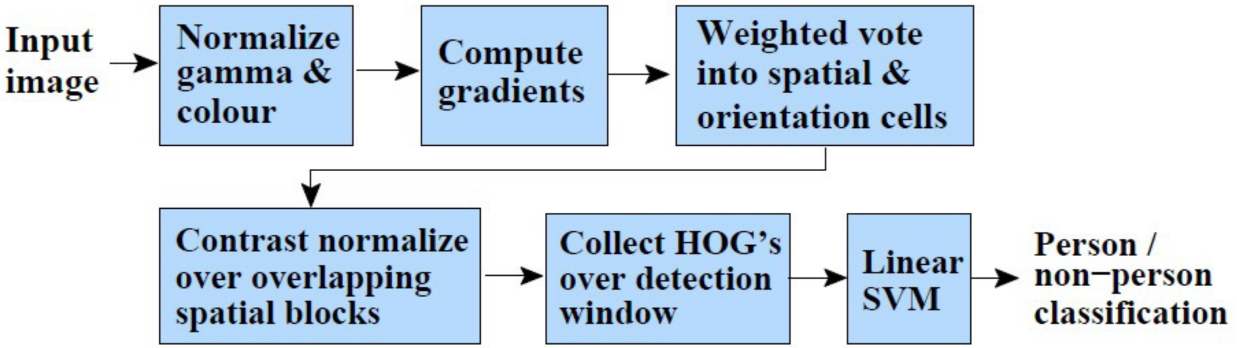


Image courtesy of Dalal & Triggs

- Dalal, Triggs: *Histogram of Oriented Gradients for Human Detection*, CVPR 2005

Other improvements

- Dollar et al: *Fast Pyramids for Object Detection*. PAMI 2014. Contributions (among others): Speed up by using less pyramid levels, interpolation of features from octave-spaced pyramid scales
- Benenson et al: *Pedestrian detection at 100 frames per second*. CVPR 2012. Contributions (among others): Computation of HOG without need of explicitly resizing the image => speedup

CENTER FOR MACHINE PERCEPTION

Thank you for your attention.