Boosting

Peter Veelaert

May 7, 2018

Table of Contents

Table of Contents

Adaboost algorithm

- combines multiple weak classifiers to produce a committee
- AdaBoost is shorthand for Adaptive Boosting
- developed by Freund and Schapire (1996)

Adaboost algorithm

Inputs

- N data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$
- N target labels $t_1, \ldots, t_N$, where $t_i \in \{-1, 1\}$
- a (large) set of weak classifiers $y_k(\mathbf{x})$

Adaboost algorithm

After the $(M-1)$-th iteration the boosted classifier is a linear combination of weak classifiers:

$$Y_{M-1}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M-1} \alpha_m y_m(\mathbf{x})\right)$$

At iteration M we want to improve the classifier by adding one weak classifier y_M:

$$Y_M(\mathbf{x}) = Y_{M-1}(\mathbf{x}) + \alpha_M y_M(\mathbf{x})$$

However, during this process, we also adapt the weights of the data samples. Wrongly classified samples get higher weights.

Adaboost algorithm

Step 1

Initialize weights w_n of data samples by setting

$$w_n^{(1)} = 1/N,$$

for $n = 1, \ldots, N$

Adaboost algorithm

For $m = 1, \dots, M$:

Step 2a

Find a classifier $y_m(\mathbf{x})$ that minimizes the weighted error function

$$J_m = \sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

where

$$I(y_m(\mathbf{x}_n) \neq t_n)$$

is an indicator function that equals one when

$$y_m(\mathbf{x}_n) \neq t_n$$

and 0 otherwise.

Adaboost algorithm

For $m = 1, \ldots, M$:

Step 2a

Find a classifier $y_m(\mathbf{x})$ that minimizes

$$J_m = \sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

Step 2b

Evaluate

$$\epsilon_m = \frac{\sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{M} w_n^{(m)}}$$

and let

$$\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

If $\epsilon_m \geq 0.5$ stop, because improvement is no longer possible.

Adaboost algorithm

For $m = 1, \ldots, M$:

Step 2a

Find a classifier $y_m(\mathbf{x})$ that minimizes

$$J_m = \sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

Step 2b

Evaluate

$$\epsilon_m = \frac{\sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{M} w_n^{(m)}}, \quad \alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

Step 2c

Update weights

$$w_n^{(m+1)} = w_n^{(m)} \exp\left(\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\right)$$

Adaboost algorithm

Step 1

Initialize weights: $w_n^{(1)} = 1/N$, for $n = 1, \ldots, N$

Step 2. For $m = 1, \ldots, M$:

Minimize

$$J_m = \sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

Evaluate

$$\epsilon_m = \frac{\sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{M} w_n^{(m)}}, \quad \alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

$$w_n^{(m+1)} = w_n^{(m)} \exp\left(\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\right)$$

Step 3

Compose final model

$$Y_m(\mathbf{x}) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

Meaning of α_m

One can show that

$$\alpha_m = \ln\left(\frac{\sum_{y_i(\mathbf{x})=t_i} w_i^{(m)}}{\sum_{y_i(\mathbf{x})\neq t_i} w_i^{(m)}}\right)$$

$\alpha_m > 0$ if

sum of weights correctly classified samples
is larger than
sum of weights incorrectly classified samples

Exponential error function

One can show that AdaBoost tries to minimize the **exponential error function**

$$E = \sum_{n=1}^{M} \exp\left(t_n f_m(\mathbf{x})\right)$$

where

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(\mathbf{x})$$

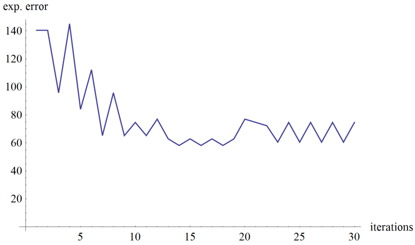
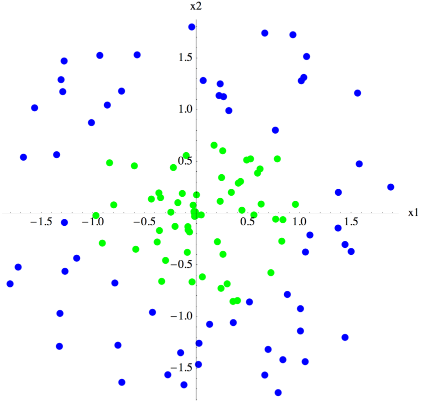
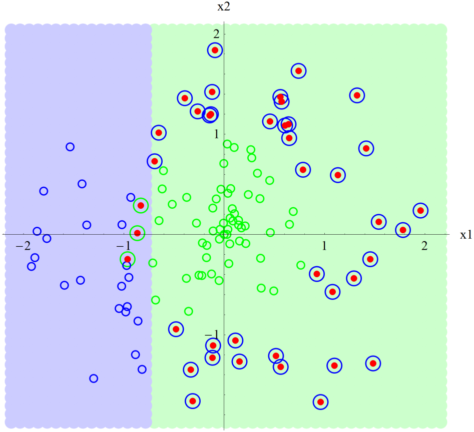


Table of Contents

Synthetic data set

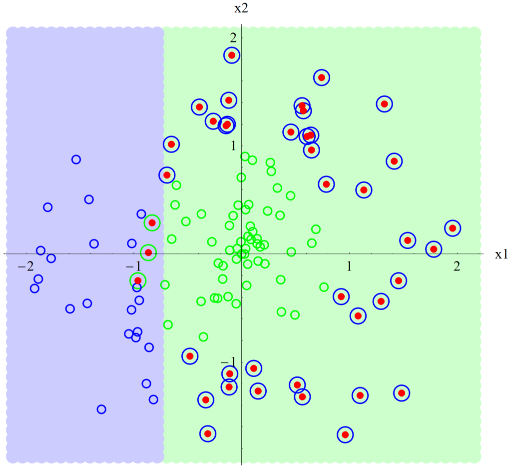


Example



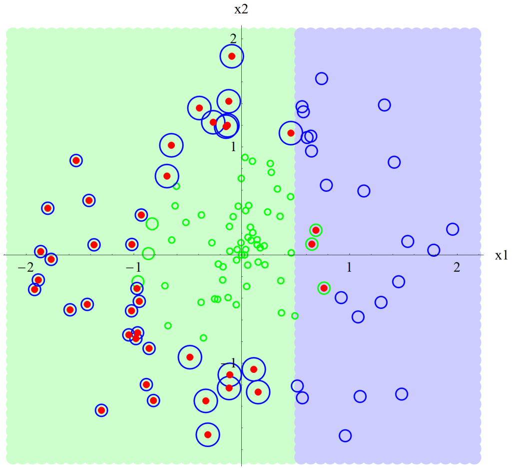
Weights are indicated by circle radii
Blue and green regions correspond to current model

Example



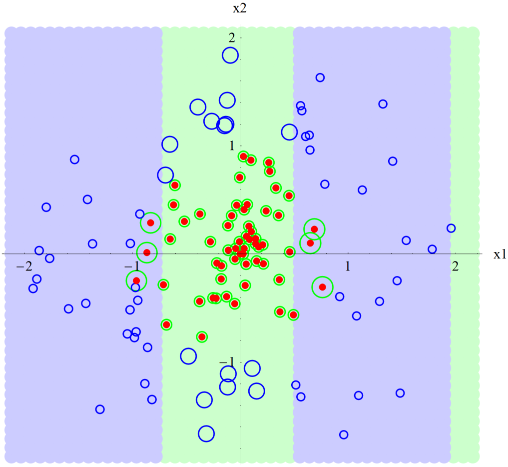
First iteration

Example



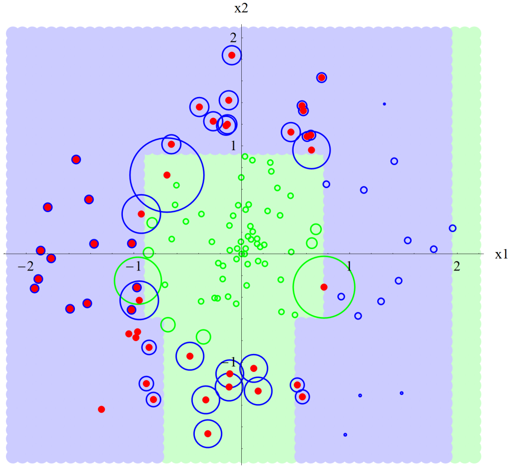
Second iteration: effect of second classifier is not yet seen in class regions (stays below threshold in current model)

Example



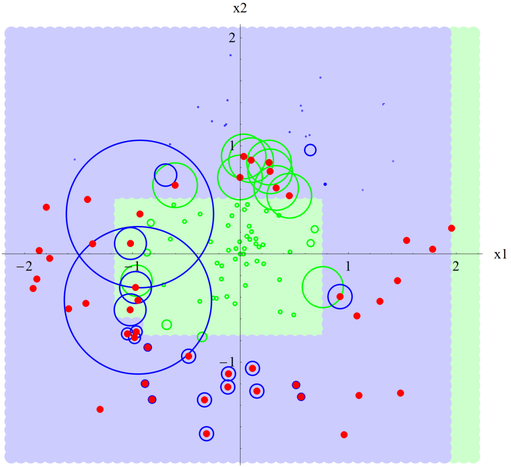
Third iteration

Example



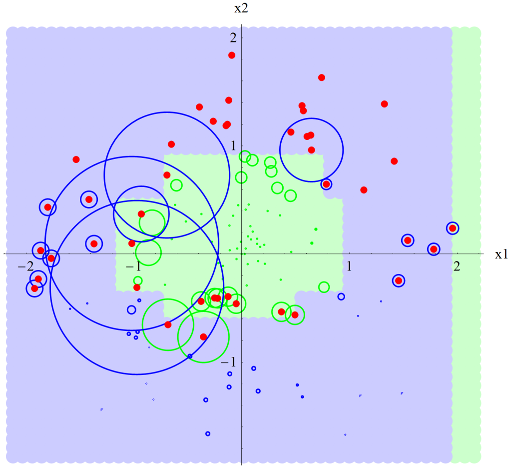
After 10 iterations

Example



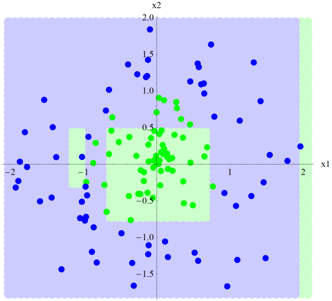
After 20 iterations

Example



After 30 iterations

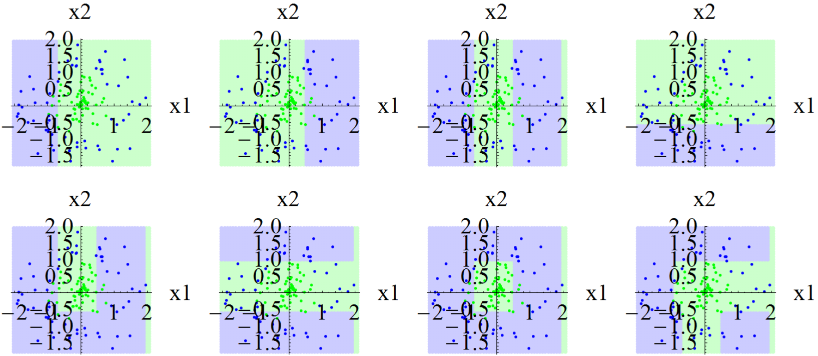
Example



Final model (after 30 iterations)

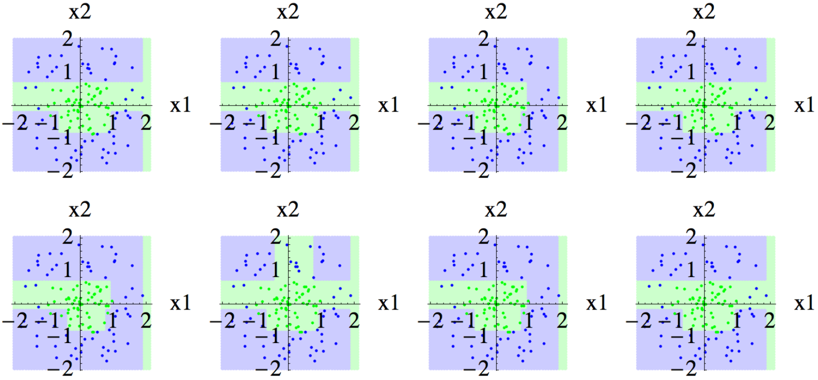
$$\mathrm{sgn}\Big(1.336\,\mathrm{sgn}(-1.000x1 - 0.9572) + 0.3425\,\mathrm{sgn}(0.5046 - 1.000x1) + 0.8882\,\mathrm{sgn}(0.7483 - 1.000x1) + 1.214\,\mathrm{sgn}(0.9919 - 1.000x1) + 7.113\,\mathrm{sgn}(x1 - 1.966) + 1.681\,\mathrm{sgn}(x1 + 0.7135) + 0.7662\,\mathrm{sgn}(x1 + 0.9572) + 1.174\,\mathrm{sgn}(x1 + 1.201) + 5.595\,\mathrm{sgn}(-1.000x2 - 0.3569) + 1.273\,\mathrm{sgn}(0.5252 - 1.000x2) + 1.819\,\mathrm{sgn}(0.9662 - 1.000x2) + 5.722\,\mathrm{sgn}(x2 + 0.3569) + 0.4938\,\mathrm{sgn}(x2 + 0.5774) + 1.012\,\mathrm{sgn}(x2 + 0.7979) \Big)$$

Example



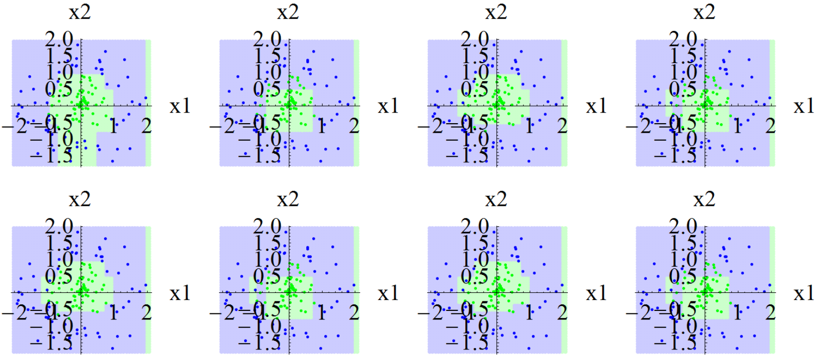
Overview of iterations 1-8

Example



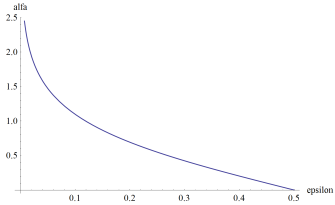
Overview of iterations 9-16

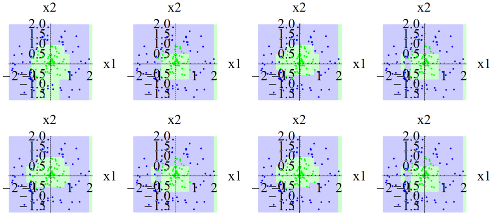
Example



Last 8 iterations

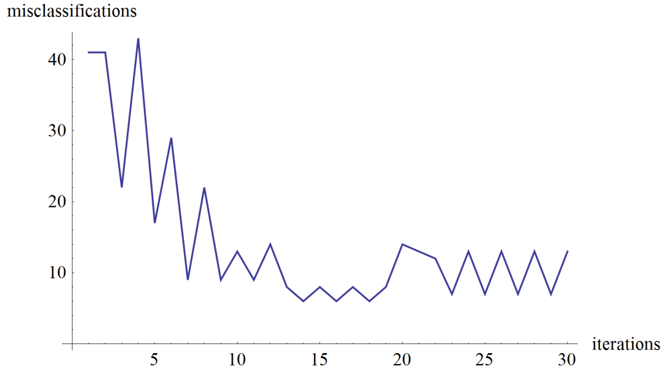
Example





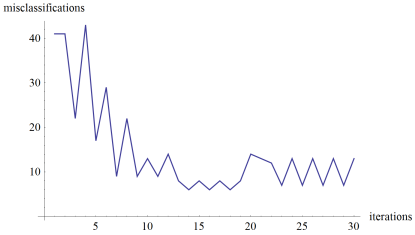
- $(0.5 - \epsilon_m)$ measures the relative improvement of each weak classifier
- as long as $(0.5 - \epsilon_m) > 0$ a weak classifier can improve the model
- classifiers that add significant improvement get larger α_m

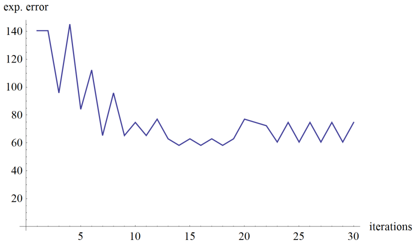
Misclassifications



Number of misclassifications of model after *m* iterations

Exponential error function

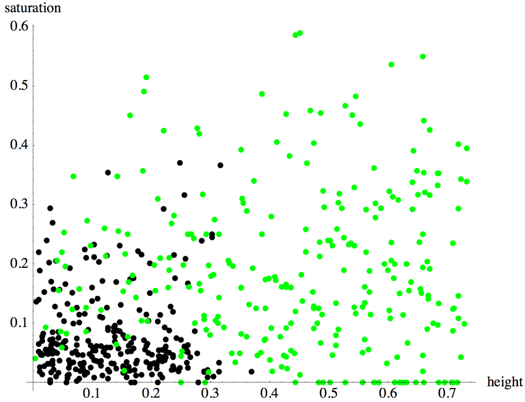




Exponential error function closely follows number of misclassifications

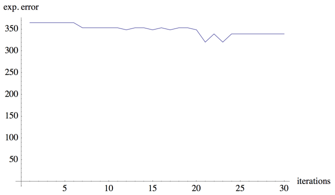
Table of Contents

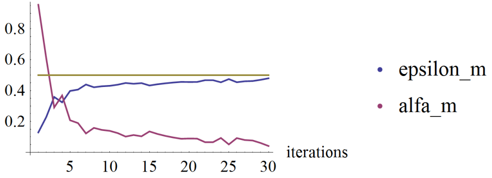
A more difficult case



Data from road/environment: saturation and height of pixel

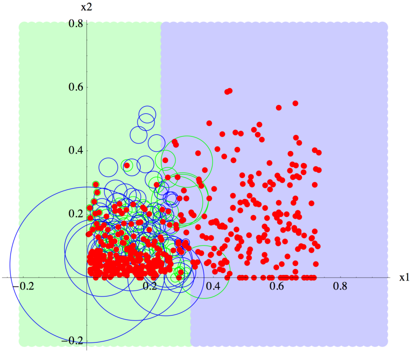
A more difficult case

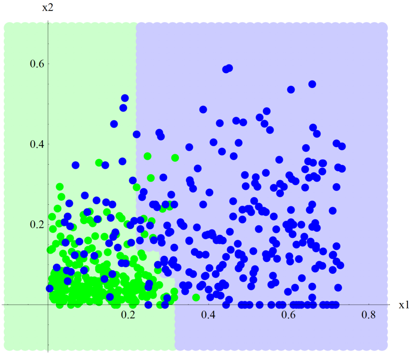




- Exponential error does not decrease significantly
- after 10 iterations improvement by weak classifiers is minimal: $(0.5 - \epsilon_m) \approx 0$

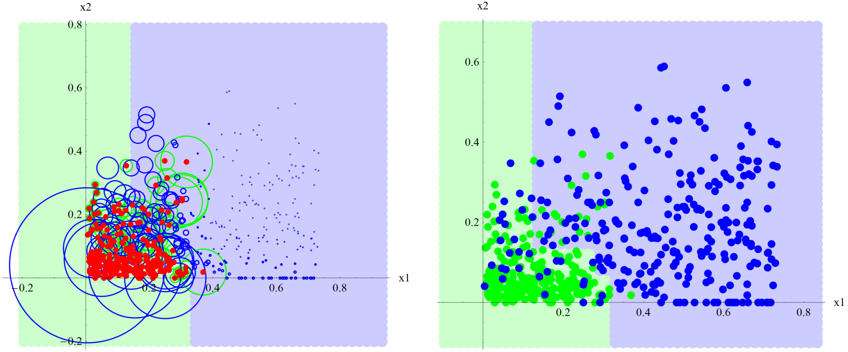
A more difficult case





After 30 iterations

A more difficult case



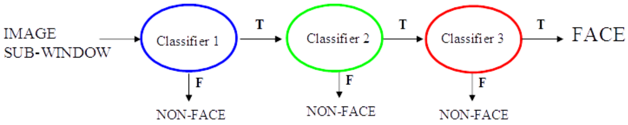
Using a larger set of weak classifiers helps

Remarks

- We used a fixed set of weak learners (=weak classifiers).
- At each iteration one could also train a new weak learner that minimizes $J_m = \sum_{n=1}^{M} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$ and add it to the set of weak classifiers.
- The weak learner must be able to handle weights.
- Possible weak learners: Perceptrons, Classification and Regression Trees (CART), ...

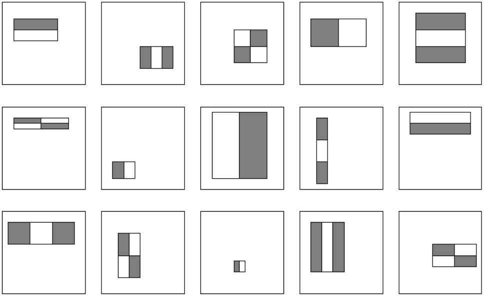
Table of Contents

Attentional cascade for face recognition



- proposed by Paul Viola and Michael Jones in 2004
- uses AdaBoost to train each classifier
- uses lower AdaBoost threshold to maximize detection rate (instead of minimizing misclassification)

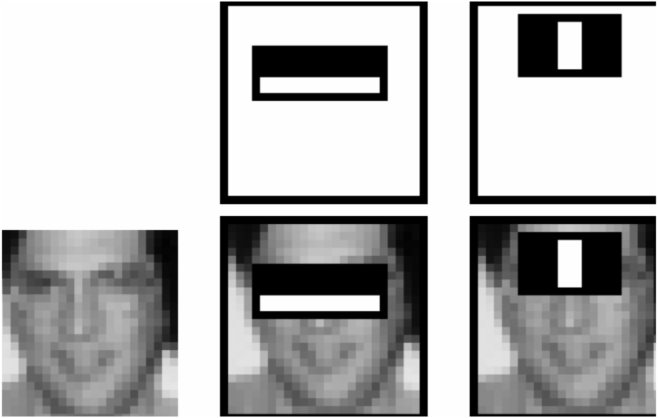
Haar features



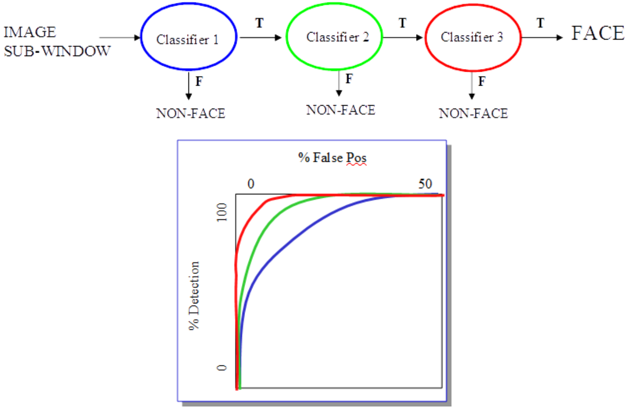
(Figures from Viola-Jones)

- Haar feature: simple rectangular black and white template
- can be computed efficiently with integral images

Haar features



Attentional cascade



Training the attentional cascade

- trained with database of 5000 faces
- faces are normalized (scale, translation)
- 300 million non-faces, 9500 non-face images
- 60000 possible Haar features to choose from
- final detector: 38 layers in cascade, 6060 features
- complexity of features increases along the cascade