Boosting algorithm

Diane Lingrand and Frédéric Precioso

UNIVERSITÉ
CÔTE D'AZUR
Master Data Science M1

2021 - 2022

Context

- Classification or regression algorithms
  - SVM, ANN, Decision Tree, Bayes Networks
- Ensemble algorithms
  - bagging : voting or averaging
    - bootstrapping - features sampling - Random Forests
  - boosting : today's topic
  - stacking

Outline

Outline

Outline

Context

- Supervised learning
  - learning dataset
  - test dataset
- Binary classification

Binary classification

- Separation between data corresponding to some criterions and data not corresponding.

Main idea of boosting

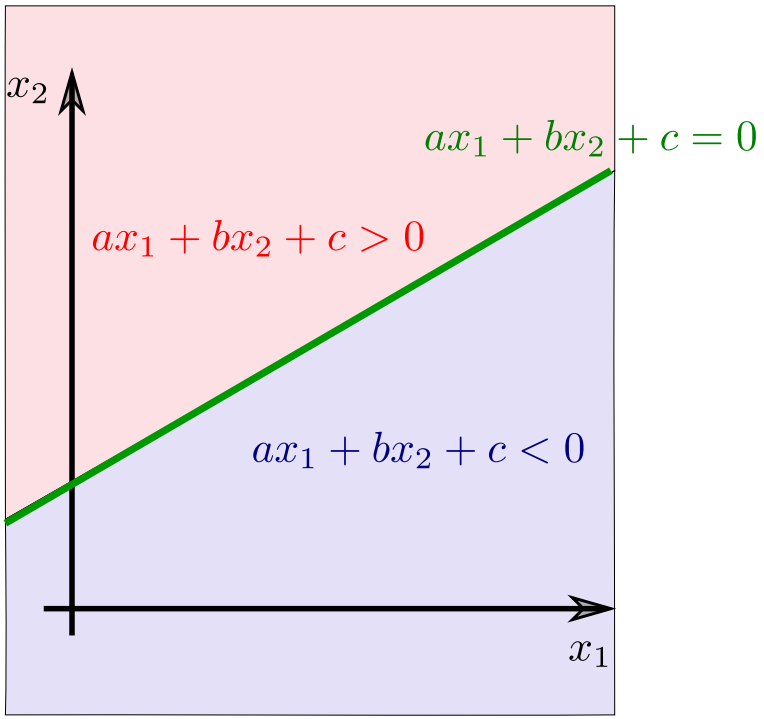
- Build a strong learner from weak learners
- Example :
  - Green corresponds to fir tree.
  - Vertical shapes corresponds to fir tree.
  - Shapes with bottom larger than top corresponds to fir tree.
  - When dominant color is red, it is not a fir tree.
  - ....
- Example of sport bet and heuristics.
  - We ask experts simple rules that are true more than 50%
  - We focus on examples for which a rule fails and we ask other rules to experts.
  - And loop the process

Origin of boosting

- Mathematician Kearns asks : "Is it possible to make as good as possible a weak learner" (that is to say better than random) ?
- Schapire, answered in 90 : " Yes ! " , and exhibited the first elementary boosting algorithm which shows that a weak binary classifier can always improve by being trained on 3 subsamples well chosen.
- The choice of the weak learner does not have any importance (a decision tree, a bayesian classifier, a SVM, a Neural Network, etc.), but one has to choose the 3 training subsamples with respect to its performance.

An example for a weak classifier : line in 2d plane

- points lie in a 2d plane
- the weak classifier is defined by a line of equation $ax_1 + bx_2 + c = 0$
  - divides the plane into 2 area :
    - $(x_1; x_2)$ for which $ax_1 + bx_2 + c > 0$
    - $(x_1; x_2)$ for which $ax_1 + bx_2 + c < 0$
  - learning means to find coefficients a, b, and c.



Outline

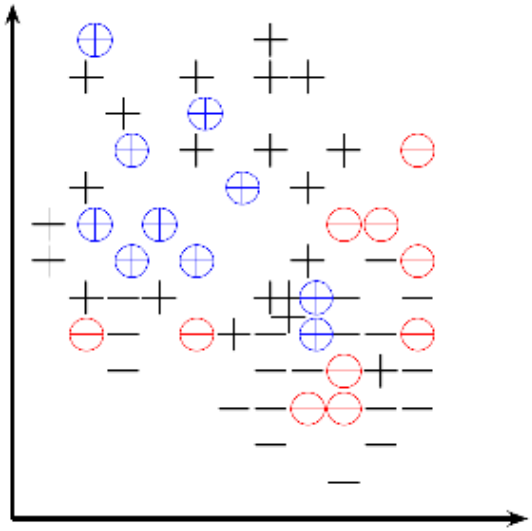
Elementary boosting algorithm

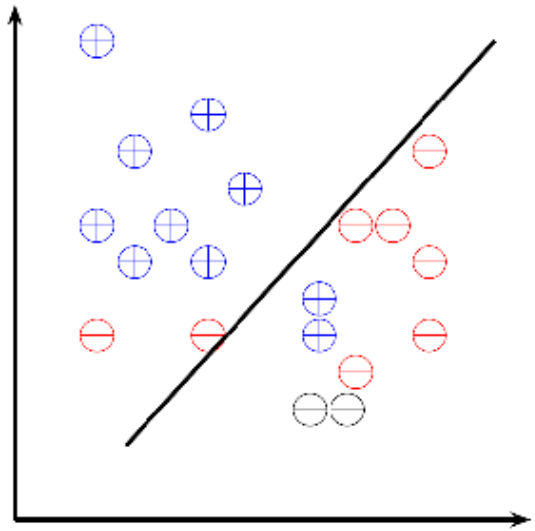
S : learning data set of m elements

1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.

A toy example (1)

Classifiers are lines. Learning a classifier corresponds to find the linear separation of data.





S : set of + and -
S_1 : reds and blues

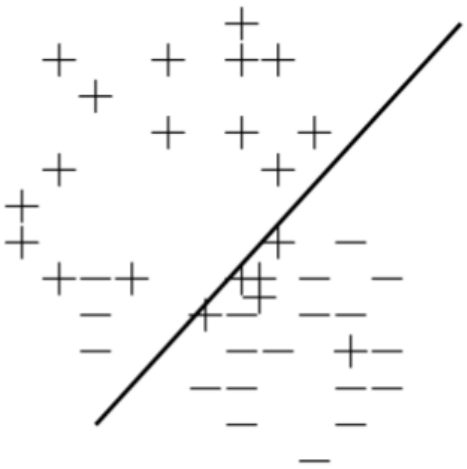
Classifier h_1 learned on S_1.

Elementary boosting algorithm

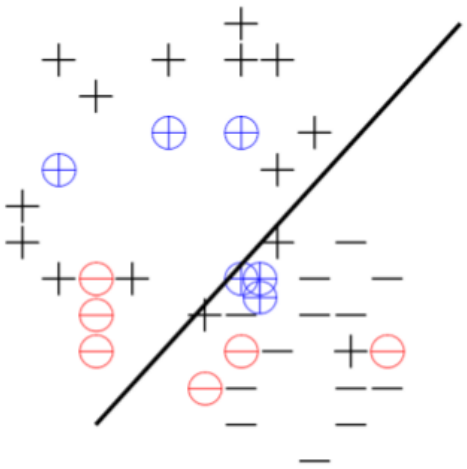
S : learning data set of m elements

1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.

A toy example (2)



$S \setminus S_1$ and h_1



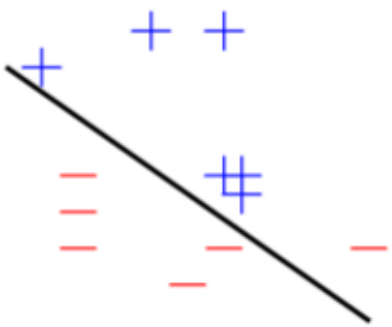
$S_2 \subset S \setminus S_1$: reds and blues

Elementary boosting algorithm

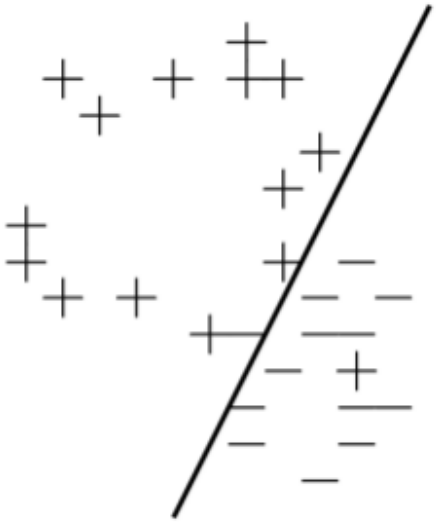
S : learning data set of m elements

1. Learn the classifier h_1 on subset $S_1 \subset S$. Test of h_1 on $S \setminus S_1$.
2. Learn the classifier h_2 on subset $S_2 \subset S \setminus S_1$ with half of elements of S_2 wrongly classified by h_1.
3. Learn classifier h_3 on subset $S_3 \subset S \setminus S_1 \setminus S_2$: contains elements for which rules h_1 and h_2 answer differently.
4. H : Majority vote between answers of h_1, h_2 and h_3.

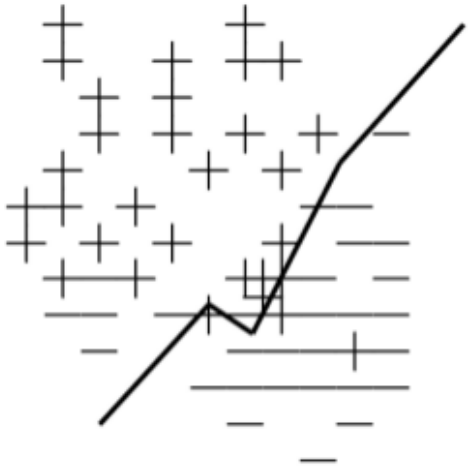
A toy example (3)



h_2 learned on S_2



S_3 and h_3



S and H

Outline

Probabilistic booosting

3 main ideas :

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.

General algorithm

- S learning dataset
- Initialisation : all samples have same weights $(D_1(i), i \in \{1 \dots m\})$.
- Iterations : for $t \in \{1 \dots T\}$
  - **learn** h_t by minimisation of an **error**
  - **compute weight** α_t and **weights** $D_t(i)$ (or distribution)
- Compute strong classifier :

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

Goal of Adaboost

- Minimisation of exponential loss : $E_{x,y}\left[e^{-yH(x)}\right]$
- Iteration t : $H_t(x) = H_{t-1}(x) + \alpha_t(x)h_t(x)$

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

Minimum when

$$-e^{-\alpha_t}P(y = h_t(x)) + e^{\alpha_t}P(y \neq h_t(x)) = 0$$

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

Adaboost : **Ada**ptative **boost**ing

- $S = \{(x_1, y_1), \dots (x_m, y_m)\}$ with $x_i \in X$ and $y_i \in \{-1, +1\}$
- Initialisation : $D_1(i) = \frac{1}{m}$ with $i \in \{1 \dots m\}$
- For $t \in \{1 \dots T\}$:
  - **find** $h_t : X \to \{-1, +1\}$ minimizing error ϵ_t defined by :

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)[y_i \neq h_t(x_i)]$$

  - compute **weights** :

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

  - compute **distribution** :

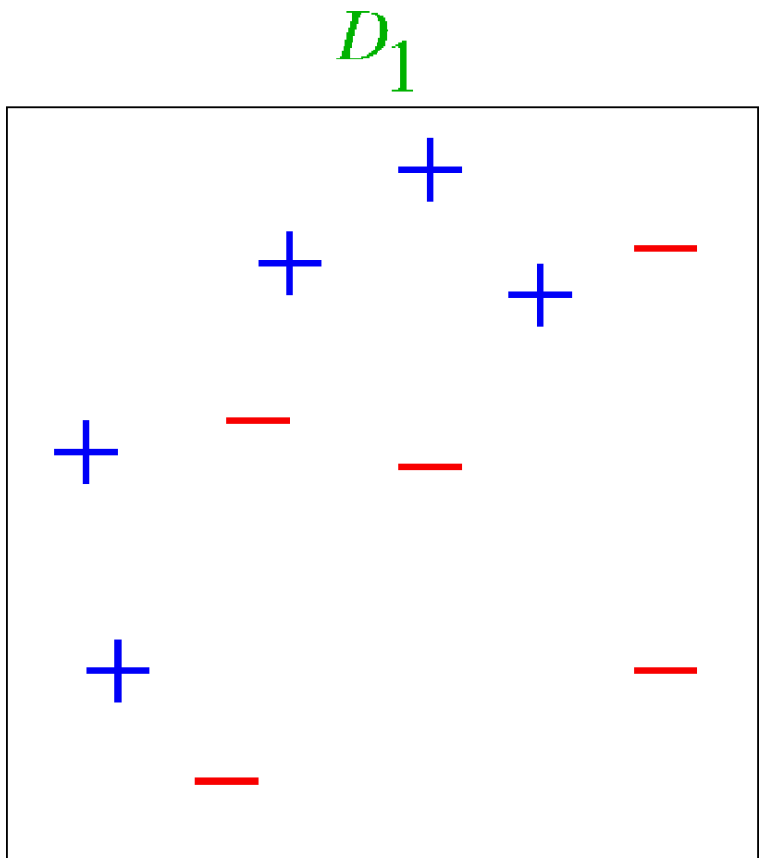
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \, y_i \, h_t(i))}{Z_t}$$

    where Z_t is for normalisation : $1 = \sum_{i=1}^{m} D_t(i)$
- Compute **strong classifier** :

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

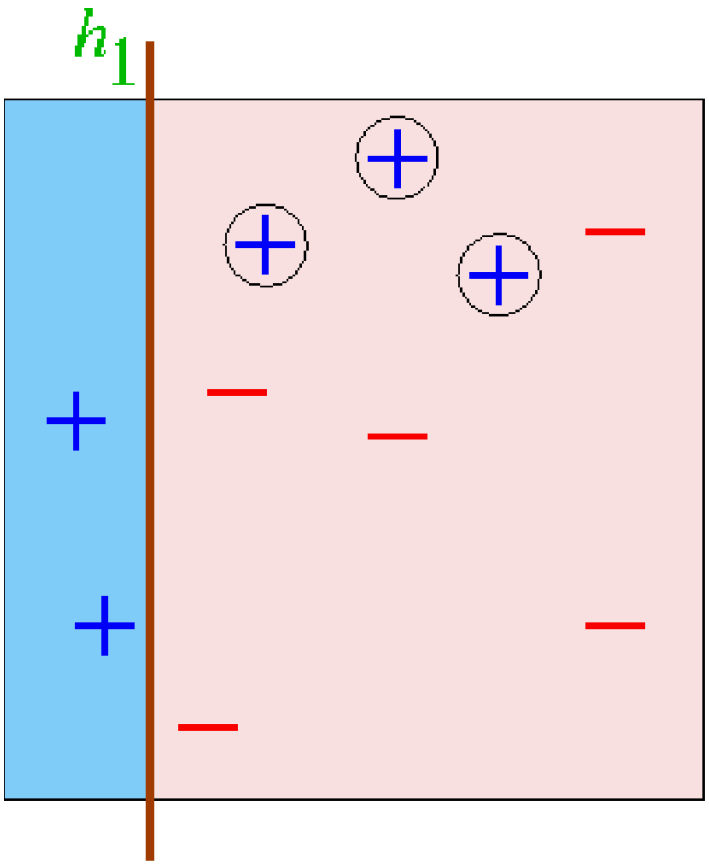
Outline

Initialisation



D_1 :	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Step 1



$$\epsilon_1 = 0.30$$

$$\alpha_1 = \frac{1}{2}\ln(\frac{1-\epsilon_1}{\epsilon_1}) = \frac{1}{2}\ln(\frac{0.7}{0.3}) = 0.42$$

$$D_2(1) = \frac{1}{Z_1} D_1(1)\, e^{\alpha_1} = \frac{0.152}{Z_1}$$

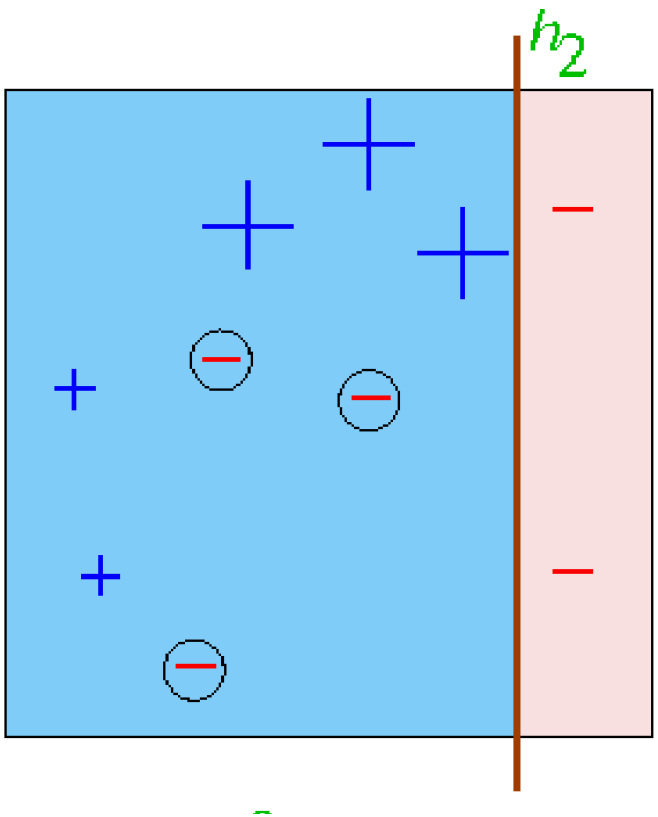
$$D_2(4) = \frac{1}{Z_1} D_1(4)\, e^{-\alpha_1} = \frac{0.065}{Z_1}$$

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

D_2 :

0.167	0.167	0.167	0.071	0.071	0.071	0.071	0.071	0.071	0.071

Step 2



$$\epsilon_2 = 0.21$$

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-\epsilon_2}{\epsilon_2}\right) = 0.65$$

$$D_3(1) = \frac{D_2(1)}{Z_2}\, e^{-\alpha_2} = 0.167\, e^{-0.65} = \frac{0.0876}{Z_2}$$

$$D_3(4) = \frac{D_2(4)}{Z_2}\, e^{-\alpha_2} = 0.071\, e^{-0.65} = \frac{0.036}{Z_2}$$

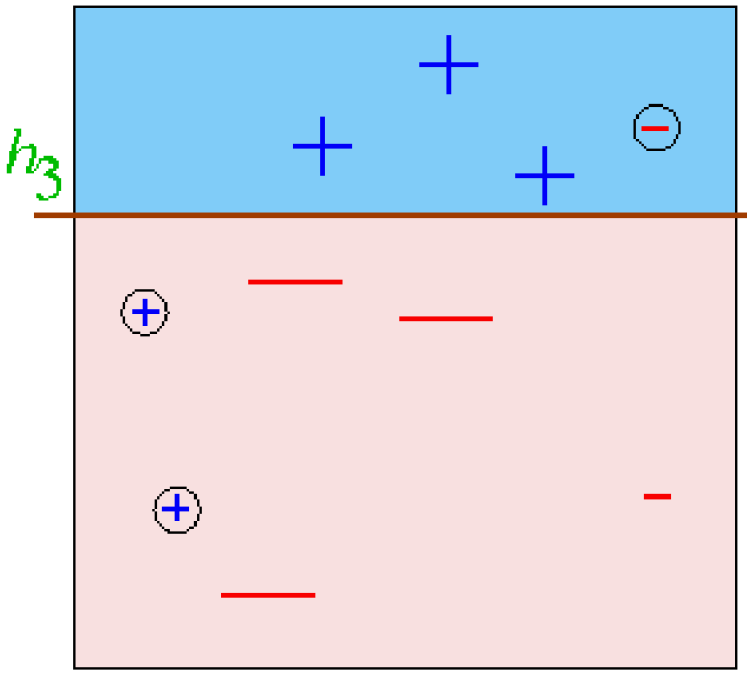
$$D_3(8) = \frac{D_2(8)}{Z_2}\, e^{+\alpha_2} = 0.071\, e^{+0.65} = \frac{0.1357}{Z_2}$$

$$Z_2 = 3\times 0.0876 + 4\times 0.036 + 3\times 0.1357 = 0.814$$

D_3 :

0.11	0.11	0.11	0.044	0.044	0.044	0.044	0.17	0.17	0.17

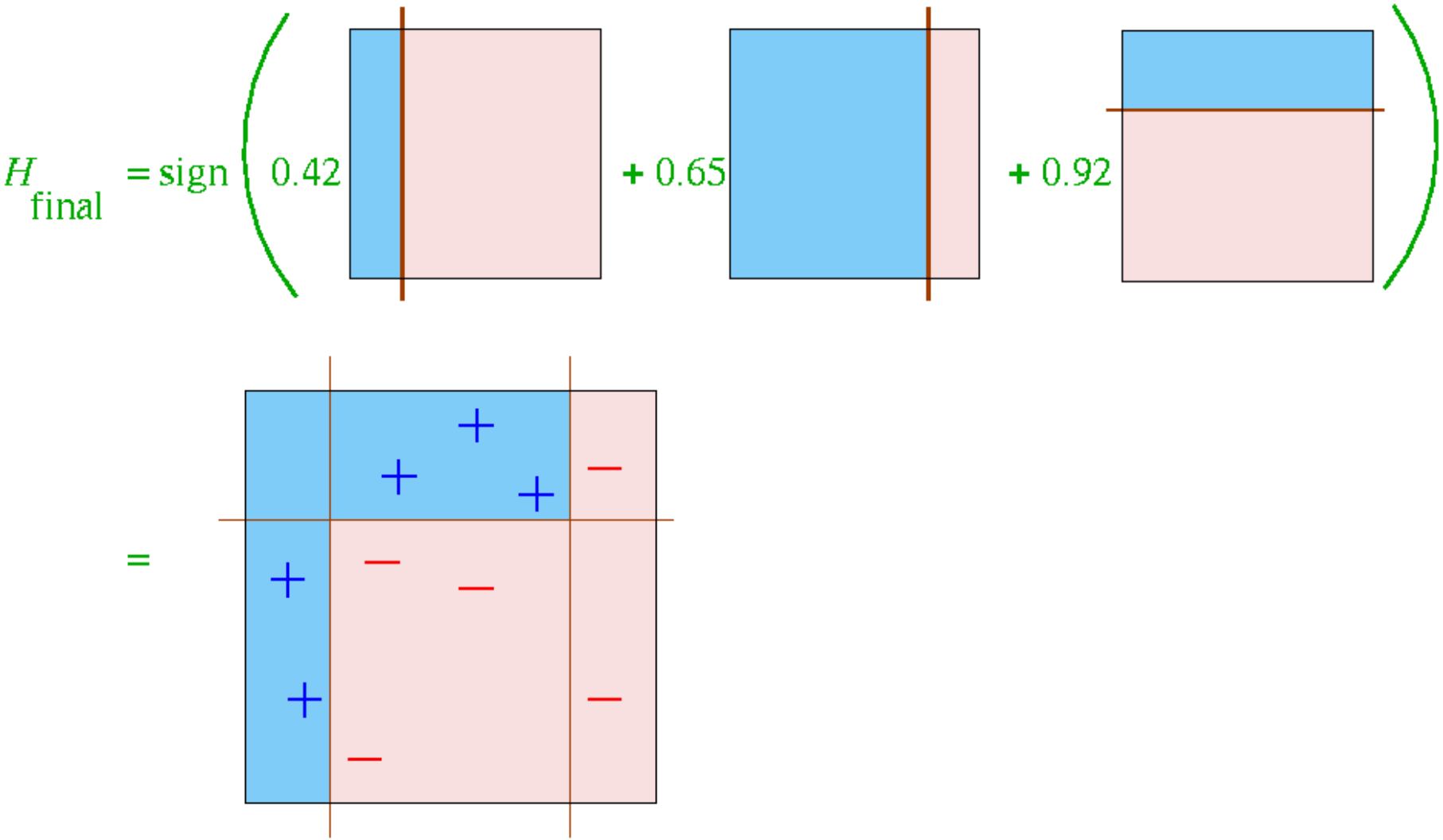
Step 3



$$\epsilon_3 = 3 \times 0.044 = 0.13$$

$$\alpha_3 = 0.92$$

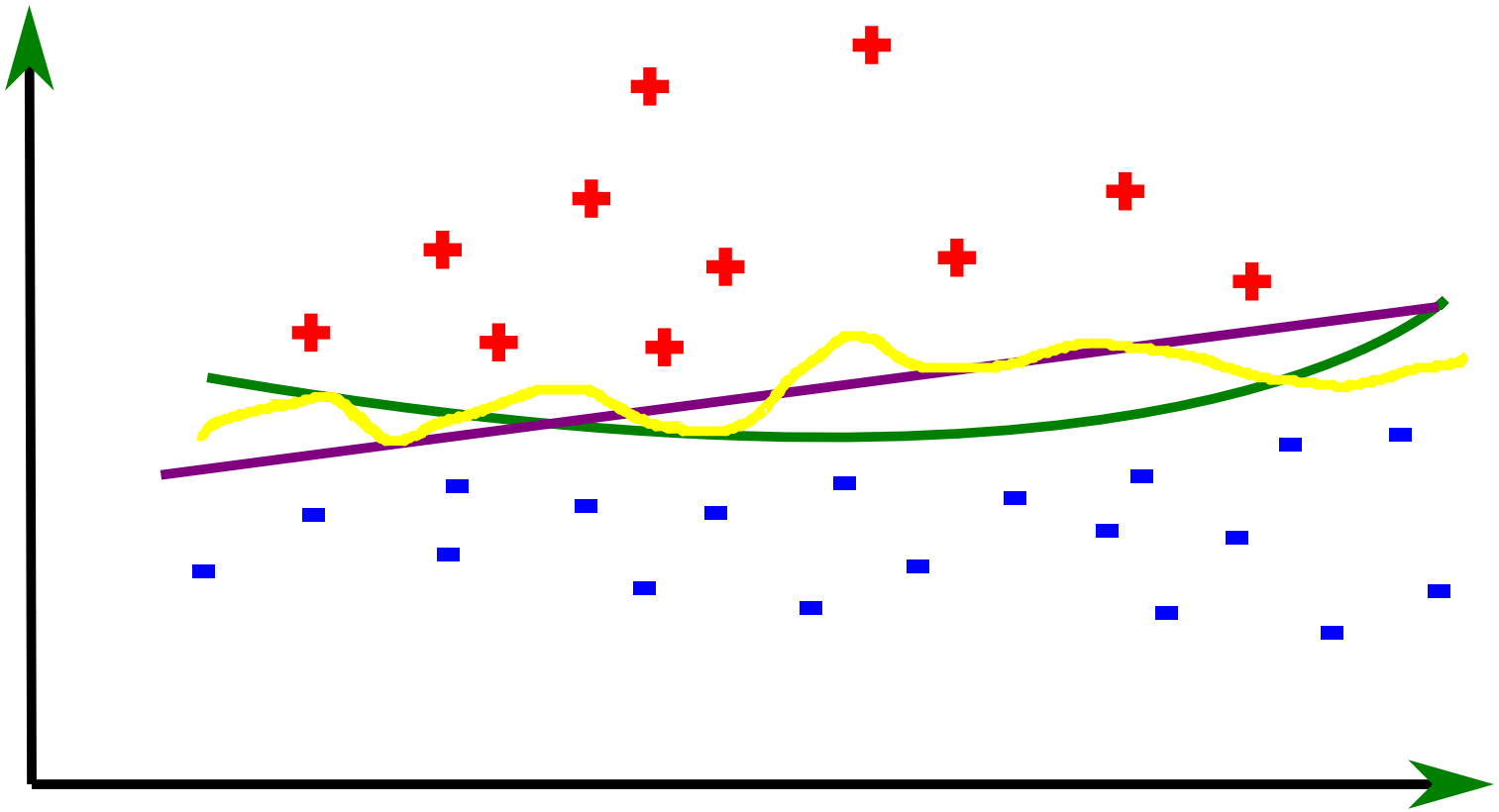
Final classifier



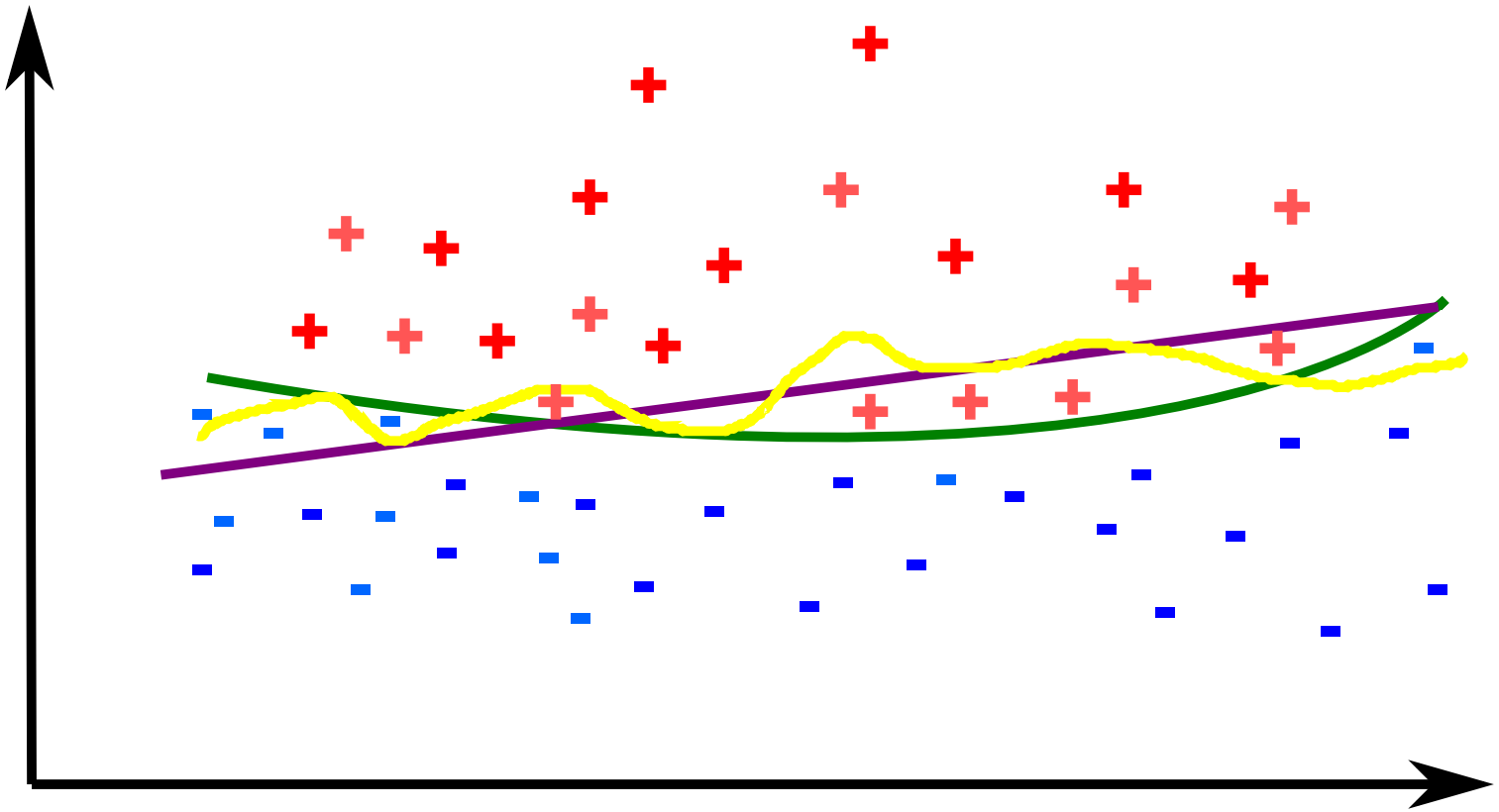
Advantages and drawbacks

- Advantages :
  - Fast
  - Easy to implement
  - only 1 parameter : number of boosting iterations
  - extensible to multi classes classification
  - able to detect outliers
- Drawbacks :
  - depends on learning data and weak classifiers
  - could fail if :
    - the weak classifier is too complex
    - low margins $\rightarrow$ overfitting
    - the weak classifier is too weak
    - underfitting
  - noise sensitive

Overfitting and Underfitting



Overfitting and Underfitting



Boosting 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
  - AdaBoost SAMME (2006)

Multi-class boosting : AdaBoost SAMME

- $S = \{(x_1, y_1), \dots (x_m, y_m)\}$ with $x_i \in X$ and $y_i \in \{1 \dots k\}$
- Initialisation : $D_1(i) = \frac{1}{m}$ with $i \in \{1 \dots m\}$
- For $t \in \{1 \dots T\}$:
  - **find** $h_t : X \to \{1 \dots k\}$ minimizing error ϵ_t defined by :

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) g(y_i, h_t(x_i)) \text{ with } g(a, b) = 1 \text{ if } a = b \text{ else } 0$$

  - compute **weights** :

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t}) + \frac{1}{2} \ln(k - 1)$$

  - compute **distribution** :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \, g(h_t(x_i), y_i))}{Z_t} \text{ where } 1 = \sum_{i=1}^{m} D_t(i)$$

- Compute **strong classifier** :

$$\arg\max_k \left(\sum_{t=1}^{T} \alpha_t \, g(h_t(x), k) \right)$$

Boosting applications

- Detection, tracking and face recognition
- Spam, Zip Code OCR
- Text classification : Schapire and Singer
- OCR : Schwenk and Bengio (neural netwoks)
- 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

Example using scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
boosting = AdaBoostClassifier(n_estimators=20, algorithm='SAMME.R')
boosting.fit(X_train, y_train)
y_pred = boosting.predict(X_test)
print('f1 score: ', f1_score(y_pred,y_test,average=None))
```

Outline

Face detection

Face detection

How to ?


A deeper view of Deep Learning
Mélanie Ducoffe, Frédéric Precioso
Laboratoire I3S - UNS CNRS UMR 7271
06/07/2016

How to ?


A deeper view of Deep Learning
Mélanie Ducoffe, Frédéric Precioso
Laboratoire I3S - UNS CNRS UMR 7271
06/07/2016

How to ?


A deeper view of Deep Learning
Mélanie Ducoffe, Frédéric Precioso
Laboratoire I3S - UNS CNRS UMR 7271
06/07/2016

How to ?


A deeper view of Deep Learning
Mélanie Ducoffe, Frédéric Precioso
Laboratoire I3S - UNS CNRS UMR 7271
06/07/2016
cnrs
i3s

How to ?


A deeper view of Deep Learning
Mélanie Ducoffe, Frédéric Precioso
Laboratoire I3S - UNS CNRS UMR 7271
06/07/2016

Problem description

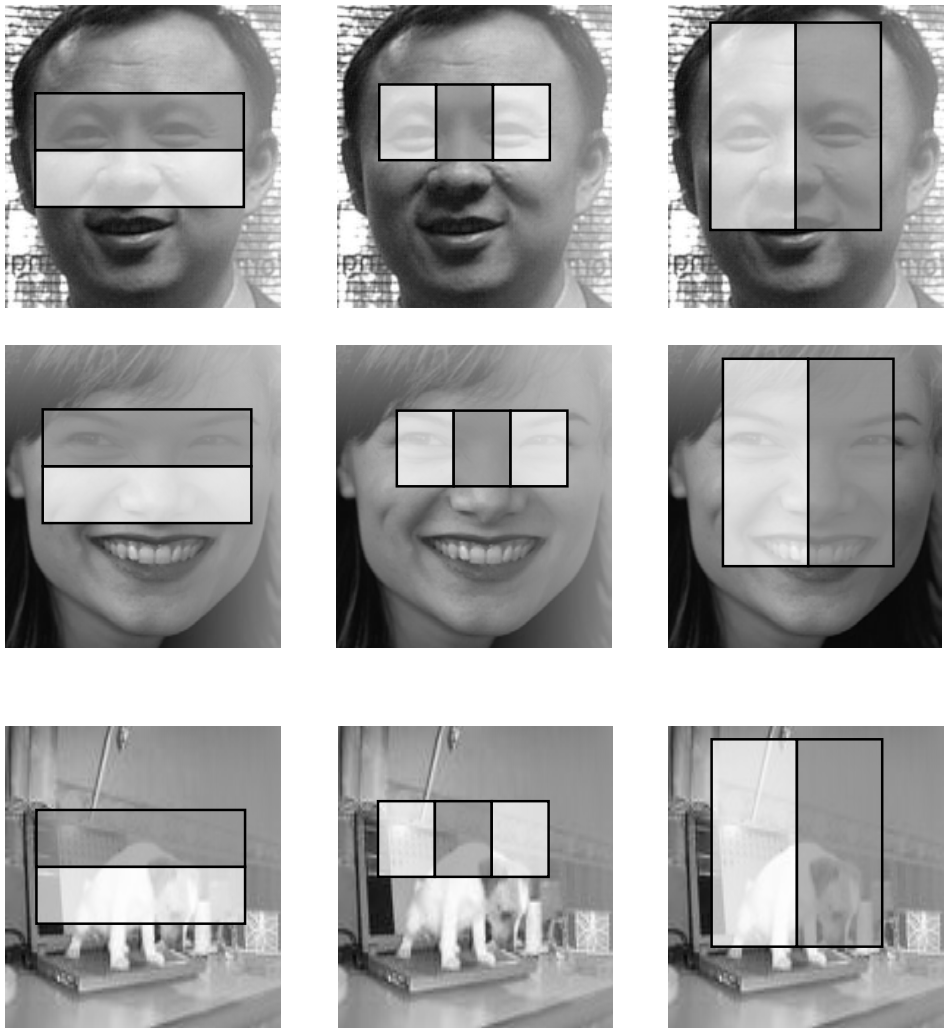
- slide a window across image and evaluate a face model at every location.
- sliding window detector must evaluate tens of thousands of location/scale combinations.
- faces are really rare compare to the number of evaluations
  - for computational efficiency, we should try to spend as little time as possible on the non-face windows
  - false positive rate should be very low ($< 10^{-6}$ for an image of 1 Mpixels)

Viola and Jones face detector

- real-time object detection
- slow training but fast detection
- 3 main ideas :
  - integral images for fast feature evaluation
  - boosting for feature selection
  - cascade classifiers for fast rejection of non face

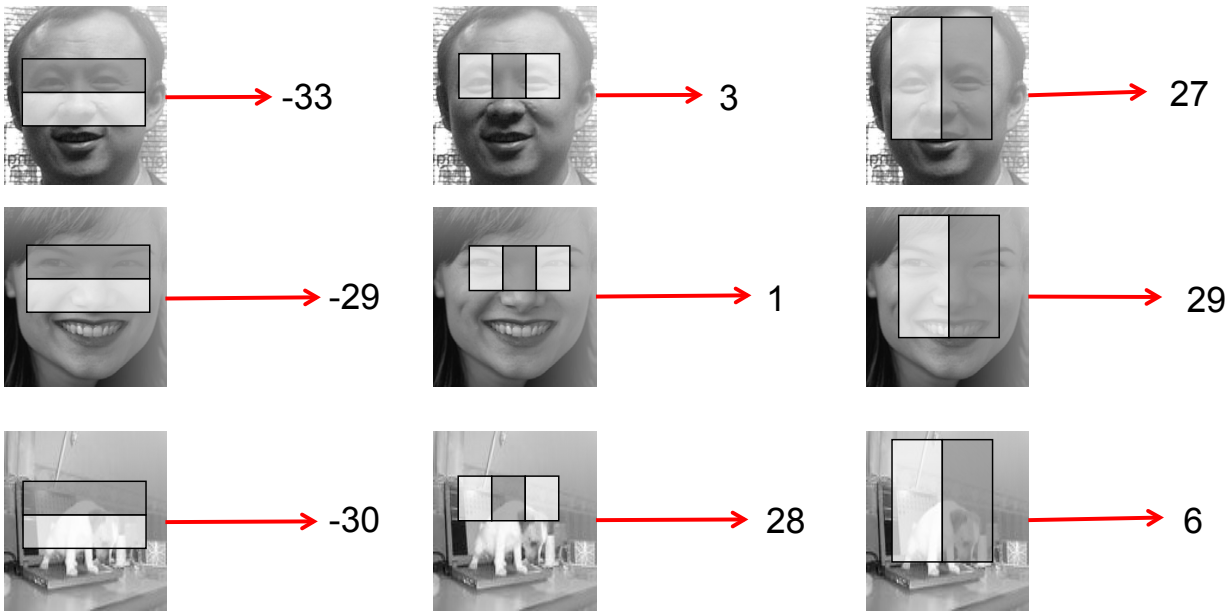
[CVPR 2001] : P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features".
[IJCV 2004] : P. Viola and M. Jones. "Robust real-time face detection", 57(2).

Finding weak classifiers for faces



$$f_t(W) = \sum_{\text{white}} \text{pixels} - \sum_{\text{black}} \text{pixels}$$

How to learn classifiers ?



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

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

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

Generalisation :

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

where the parity p_t equals ± 1, threshold θ_t has to be learned.

Boosting algorithm

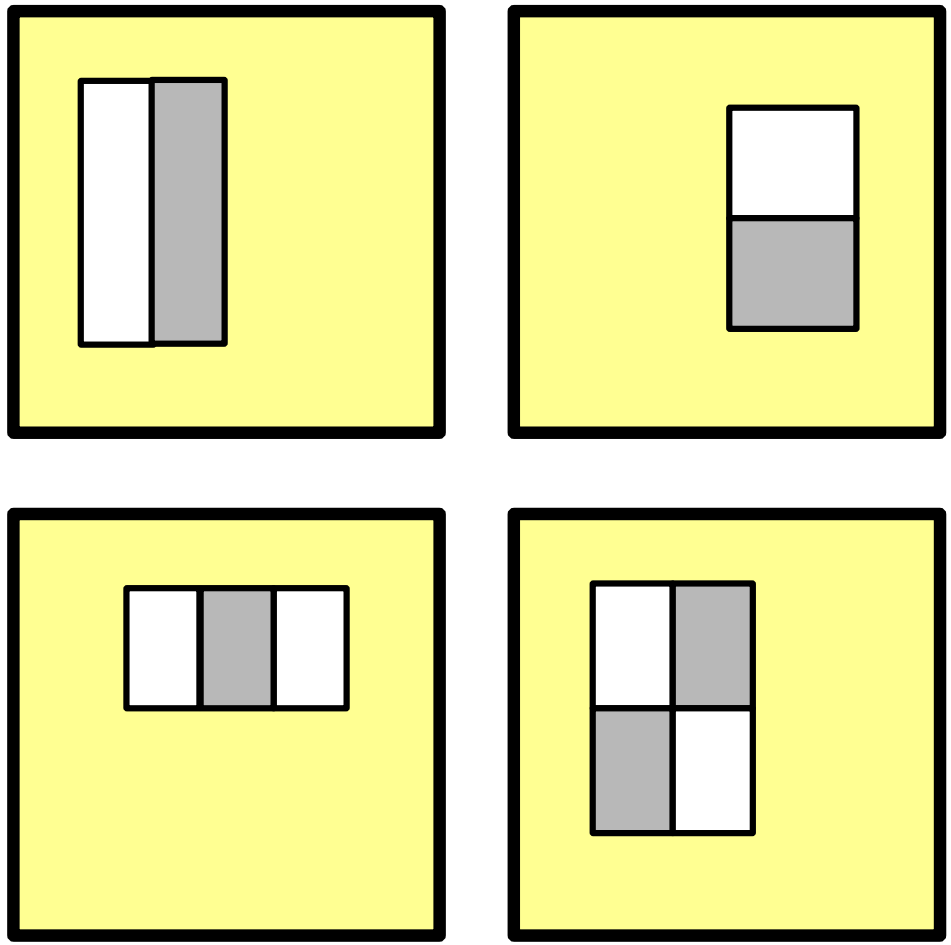
- S learning dataset : faces and non faces
- Initialisation : same weights for faces ($\frac{1}{m_v}$) and same weights for non faces ($\frac{1}{m_{nv}}$).
- Iterations : for $t \in \{1 \dots T\}$
  - for all available filters :
    - learn threshold θ for all learning samples
  - select best filter h_t with threshold θ_t
  - compute weights α_t and $D_t(i)$
- compute strong classifier

K : number of filters

Learning complexity : O(T.(nb. samples).K)

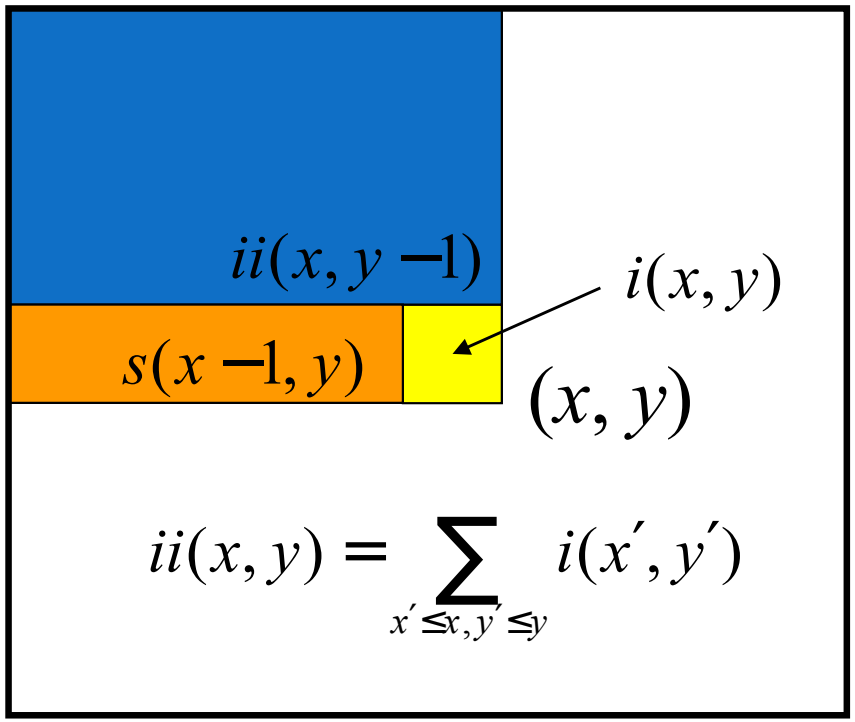
Weak classifiers : rectangular filters

- based on Haar filters
- difference between sum of white related pixels and sum of black related pixels
- 3 types : 2, 3 or 4 rectangles
- combinaisons (rectangles, sizes, positions) for a 24x24 window : 160 000 filters



Integral Image (II) computation

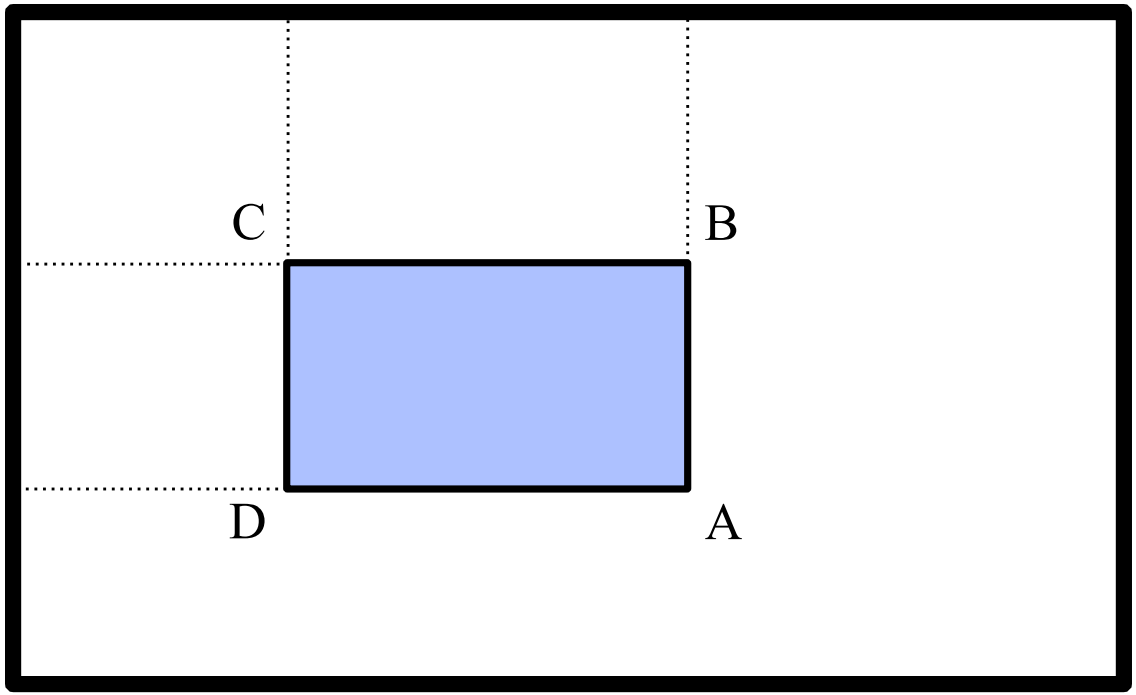
- i is the initial image ; ii is the integral image
- Each pixel (x,y) of the integral image represents the sum of all pixels above and to the left of (x,y), from the initial image.
- Integral image is computed in one pass using a temporary structure representing the sum of pixels on the same row, at left : s.



$$
\begin{aligned}
s(x, y) &= s(x-1, y) + i(x, y) \\
ii(x, y) &= ii(x, y-1) + s(x, y)
\end{aligned}
$$

Use of integral image

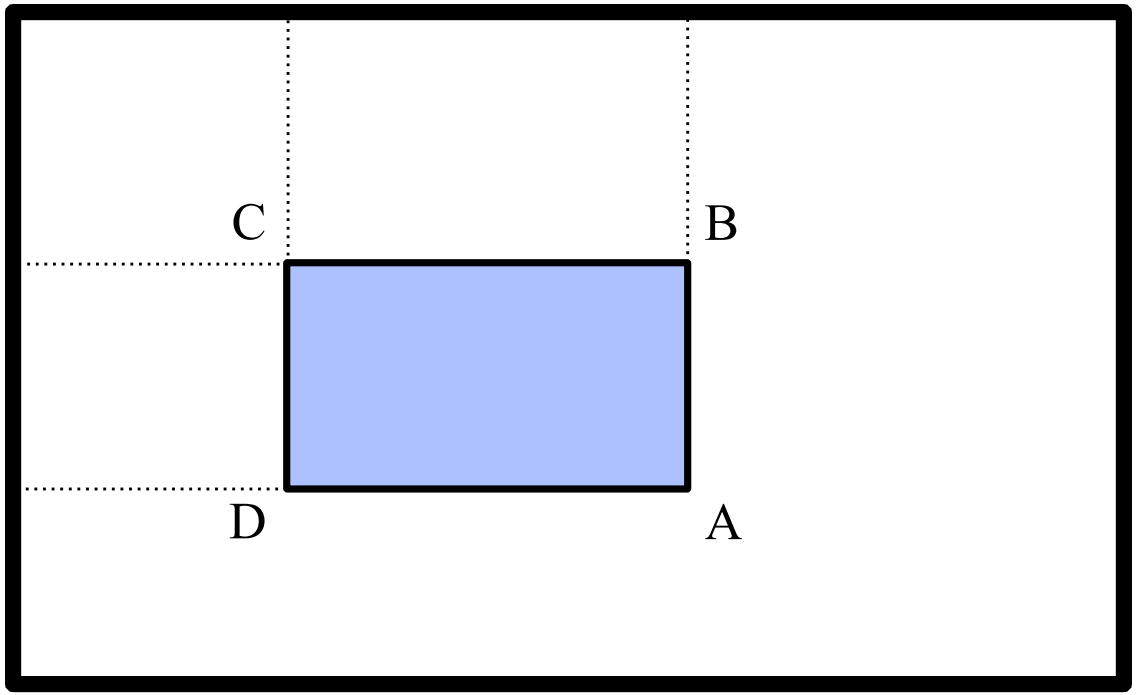
Compute the sum of pixels inside a rectangular area ABCD :



$$S = \sum_{x=x_C}^{x_A} \sum_{y=y_D}^{y_C} i(x,y) = ii_A - ii_B + ii_C - ii_D$$

Use of integral image

Compute the sum of pixels inside a rectangular area ABCD :



$$S = \sum_{x=x_C}^{x_A} \sum_{y=y_D}^{y_C} i(x,y) = ii_A - ii_B + ii_C - ii_D$$

Only 3 operations are needed !

Other optimisations :

- Different scales :
  - Instead of changing samples scales, we change the filters scales.
- Normalisation of images computed from integral images and squared integral images :

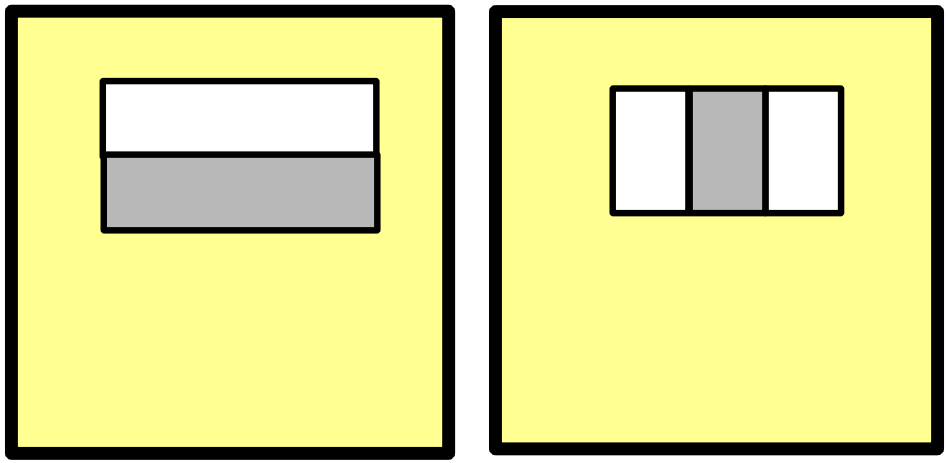
$$\sigma^2 = m^2 - \frac{1}{N}\sum x^2$$

In practice

Learning dataset :

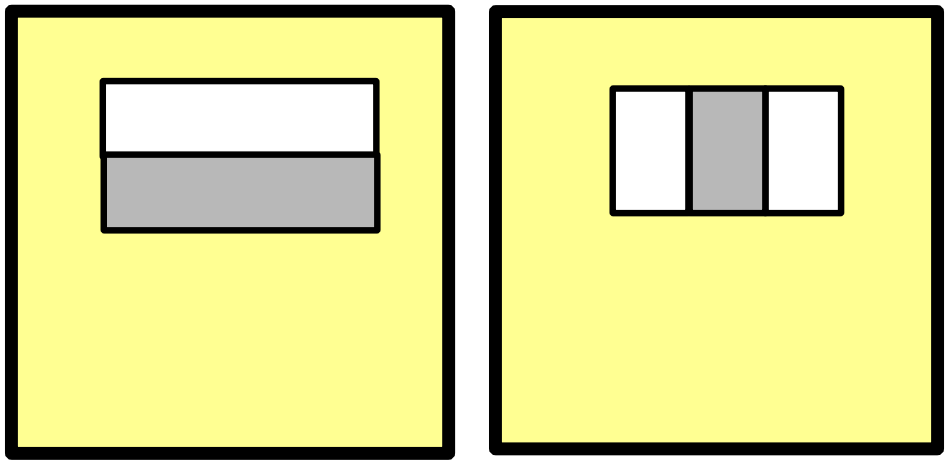
- 5000 faces
  - front oriented
  - size 24x24 pixels
  - normalised
- 300 millions of non faces
  - taken from 9500 images without faces
- variability :
  - between individuals
  - illumination
  - pose

Selected filters by *boosting* : the first 2 features



- conduct to :
  - 100% detection rate (ratio true positives over positives)
  - 50% false positives rate (ratio false positives over negatives)
- able to eliminate non faces very fastly

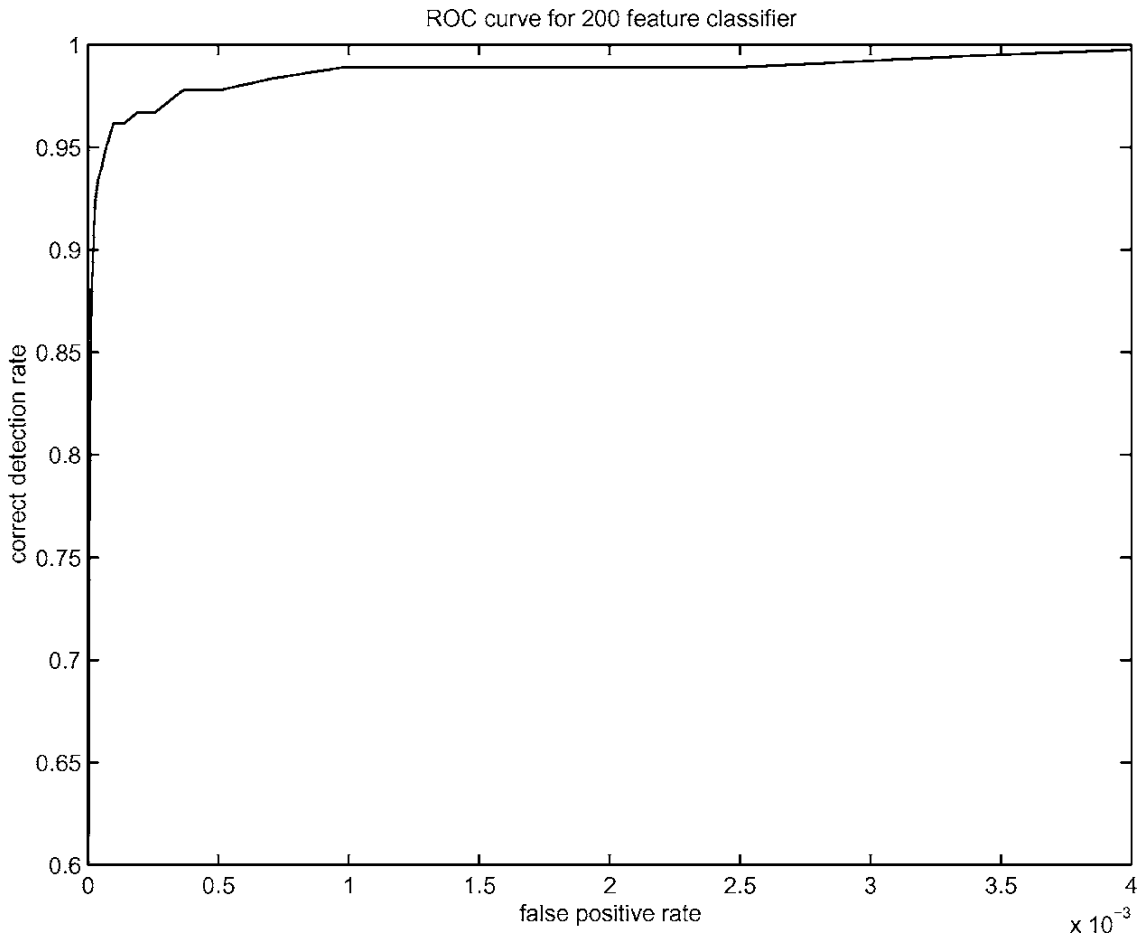
Selected filters by *boosting* : the first 2 features



- conduct to :
  - 100% detection rate (ratio true positives over positives)
  - 50% false positives rate (ratio false positives over negatives)
- able to eliminate non faces very fastly

$\Rightarrow$ Fast elimination of non faces and increase of filters complexity

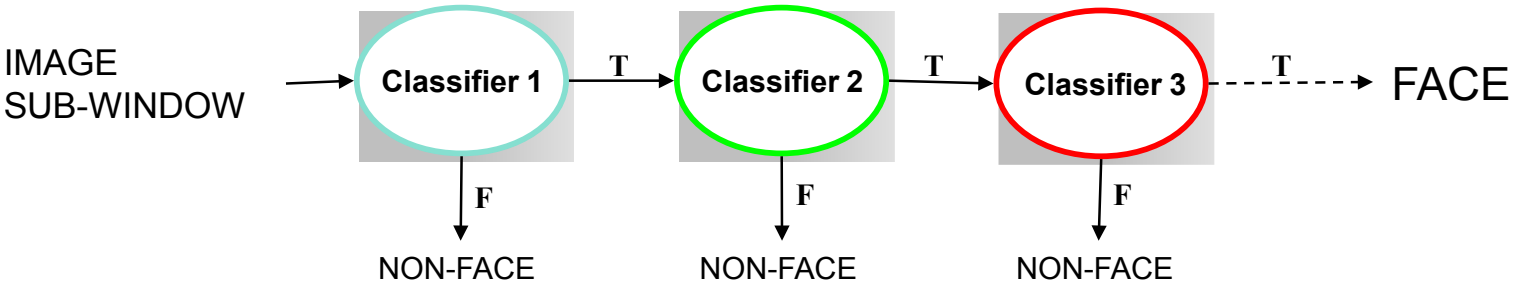
The 200 best filters selected : ROC curve



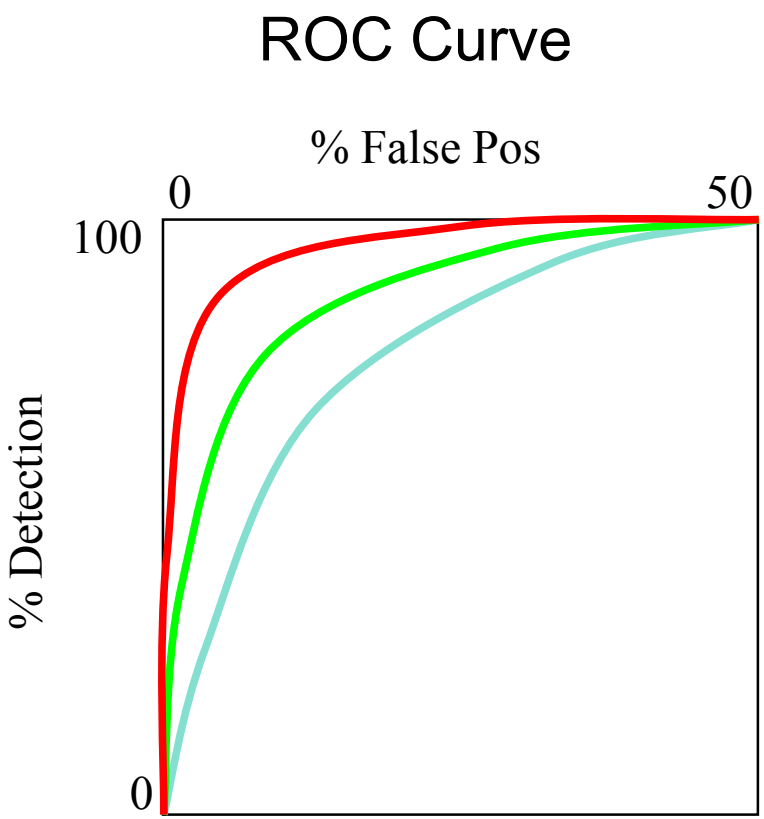
A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084. To be practical for real application, the false positive rate must be closer to 1 in 1,000,000.

Attentional Cascade

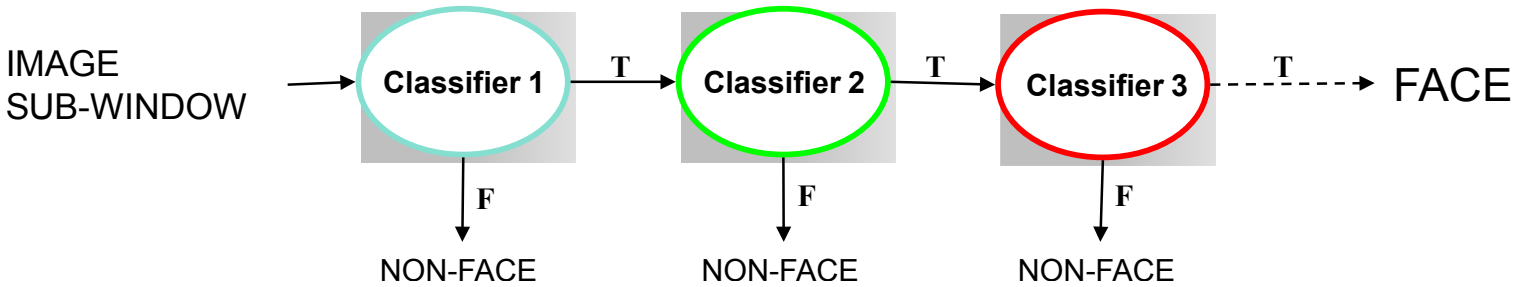
- We start with simple classifiers which reject many of the negative sub-windows while detecting almost all positive sub-windows
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on
- A negative outcome at any point leads to the immediate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



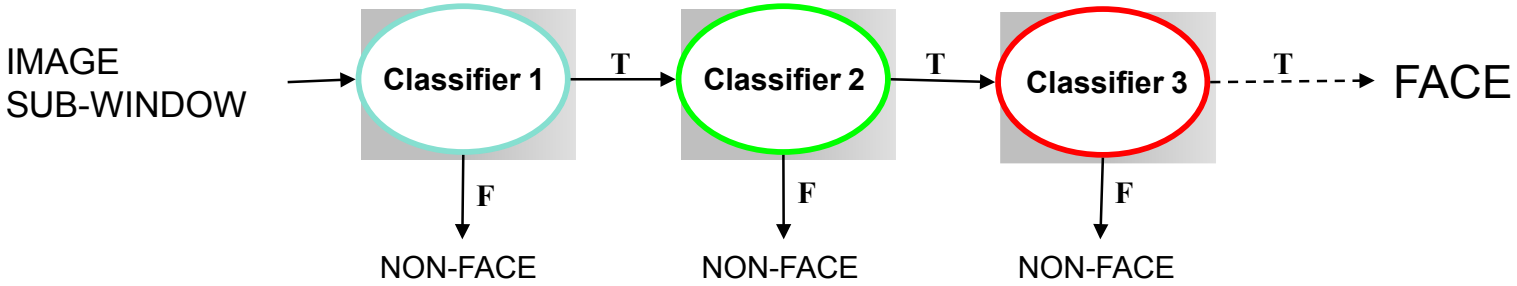
Attentional Cascade



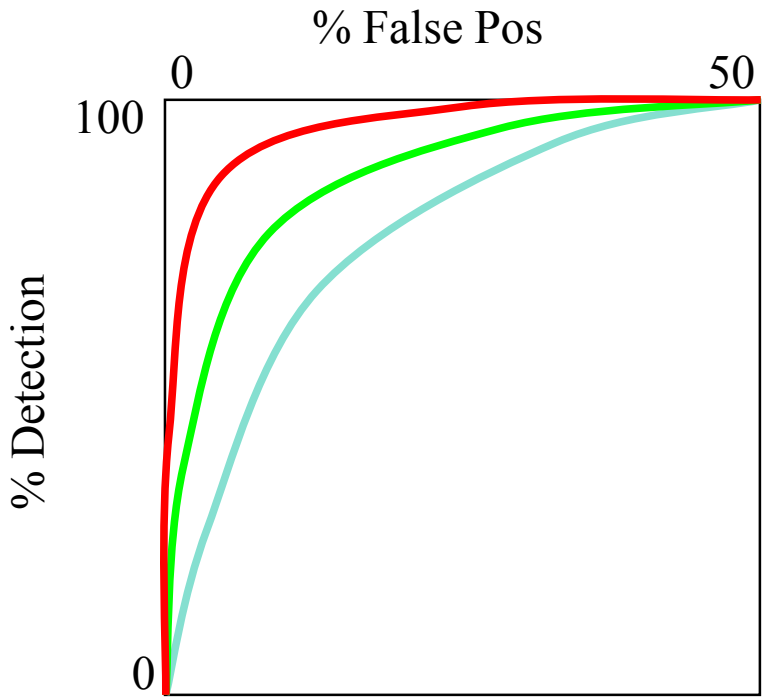
Chain classifiers that are progressively more complex and have lower false positive rates



Cascade structure



ROC Curve



$$F = \prod_{i=1}^{K} f_i$$

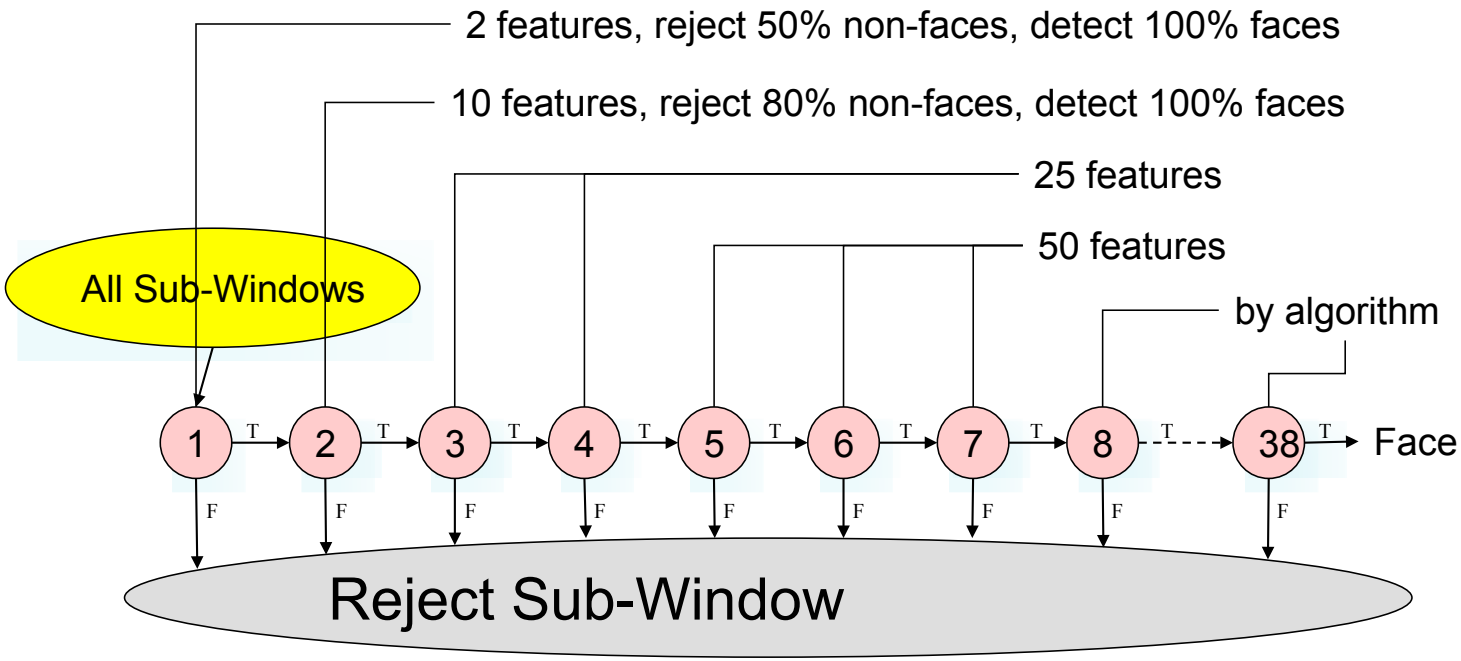
$$D = \prod_{i=1}^{K} d_i$$

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 ($0.3^{10} \approx 6\,10^{-6}$)

Algorithm for determining the stages of the cascade

- choose $f = \max_i f_i$ and $d = \min_i d_i$
- choose F
- P : set of faces
- N : set of non faces
- initialisations : $F_0 = 1.0 \quad D_0 = 1.0 \quad i = 0$
- while $F_i > F$
  - add a new stage : $i \leftarrow i + 1$
  - $n_i = 0 \quad F_i = F_{i-1}$
  - while $F_i > f\, F_{i-1}$:
    - add filters : $n_i \leftarrow n_i + 1$
    - boosting with n_i filters
    - evaluation of cascade classifier on validation data : F_i and D_i
    - lower the threshold of classifier i for having a dectection rate at least : $d.D_{i-1}$ (impacts also F_i)
  - empty N
  - false positives go to N for the next step

The final cascade



- Consists in combining in cascade more and more complex classifiers
  - 38 stages
  - total of 6060 classifiers
- Historical performances :
  - on a Pentium III, 700 MHz, processing of a 384x288 pixels images in 0.067 seconds (15 Hz)

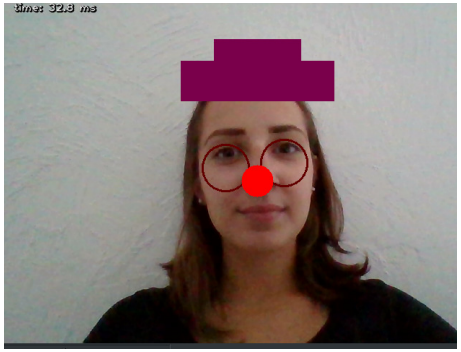
Extensions

- Other filters (rotation 45 deg)
- Nested cascades :
  - eyes detection
  - nose detection
  - smile detection

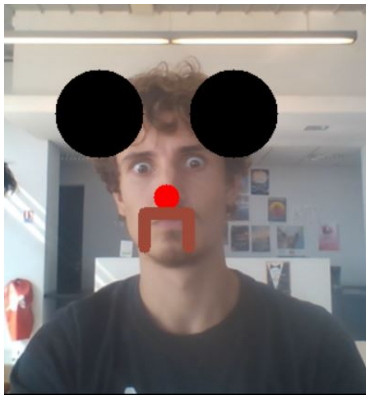
How to escape to facedetect ?

see https://cvdazzle.com/

Let's play with facedetect !



Alexia



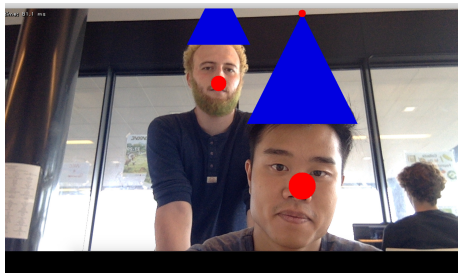
Erwan

Anthony

Alexis G.

Yilei

Simon



Vincent

Maxime

Mariana

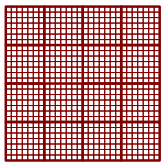
Tests on MNIST database

- 28x28 grey level images : 784 pixels
- Haar filters : `'type-2-x'` and `'type-2-y'` : 158760 filters
- if we remove filters of diagonal size $< \sqrt{50}$ and take only 1 over 128 filters : 859 filters
- only two classes : 4 and 8 ; confusion matrix $\begin{bmatrix} 980 & 2 \\ 6 & 968 \end{bmatrix}$
- with all classes

$$\begin{bmatrix} 887 & 0 & 15 & 7 & 7 & 7 & 11 & 1 & 38 & 7 \\ 1 & 1105 & 8 & 3 & 0 & 5 & 1 & 1 & 11 & 0 \\ 6 & 10 & 784 & 52 & 44 & 9 & 67 & 5 & 52 & 3 \\ 2 & 2 & 76 & 805 & 2 & 26 & 2 & 11 & 48 & 36 \\ 5 & 1 & 13 & 7 & 829 & 5 & 12 & 7 & 10 & 93 \\ 3 & 2 & 16 & 104 & 1 & 666 & 23 & 13 & 42 & 22 \\ 12 & 4 & 27 & 0 & 31 & 13 & 867 & 0 & 4 & 0 \\ 1 & 9 & 24 & 12 & 9 & 7 & 0 & 820 & 17 & 129 \\ 56 & 12 & 21 & 86 & 10 & 28 & 18 & 8 & 680 & 55 \\ 4 & 8 & 3 & 20 & 120 & 4 & 0 & 138 & 24 & 688 \end{bmatrix}$$

Tests on MNIST database

- only two classes : 4 and 8



  - haar features with resolution quarter w(=28) and quarter h
  - size of descriptor : 168 (data 784 pixels)
  - confusion matrix

$$\begin{bmatrix} 971 & 11 \\ 12 & 962 \end{bmatrix}$$

- with all classes
  - confusion matrix

$$\begin{bmatrix} 825 & 0 & 18 & 4 & 10 & 49 & 40 & 2 & 31 & 1 \\ 0 & 1042 & 6 & 2 & 0 & 1 & 3 & 17 & 64 & 0 \\ 15 & 45 & 812 & 11 & 22 & 4 & 20 & 17 & 80 & 6 \\ 52 & 60 & 51 & 653 & 4 & 46 & 1 & 27 & 82 & 34 \\ 5 & 4 & 1 & 3 & 838 & 5 & 15 & 11 & 21 & 79 \\ 42 & 19 & 17 & 116 & 27 & 467 & 15 & 8 & 123 & 58 \\ 46 & 4 & 45 & 0 & 20 & 22 & 757 & 5 & 58 & 1 \\ 1 & 22 & 22 & 12 & 30 & 21 & 0 & 767 & 22 & 131 \\ 27 & 19 & 29 & 58 & 28 & 26 & 15 & 8 & 744 & 20 \\ 5 & 15 & 5 & 22 & 439 & 8 & 2 & 40 & 37 & 436 \end{bmatrix}$$

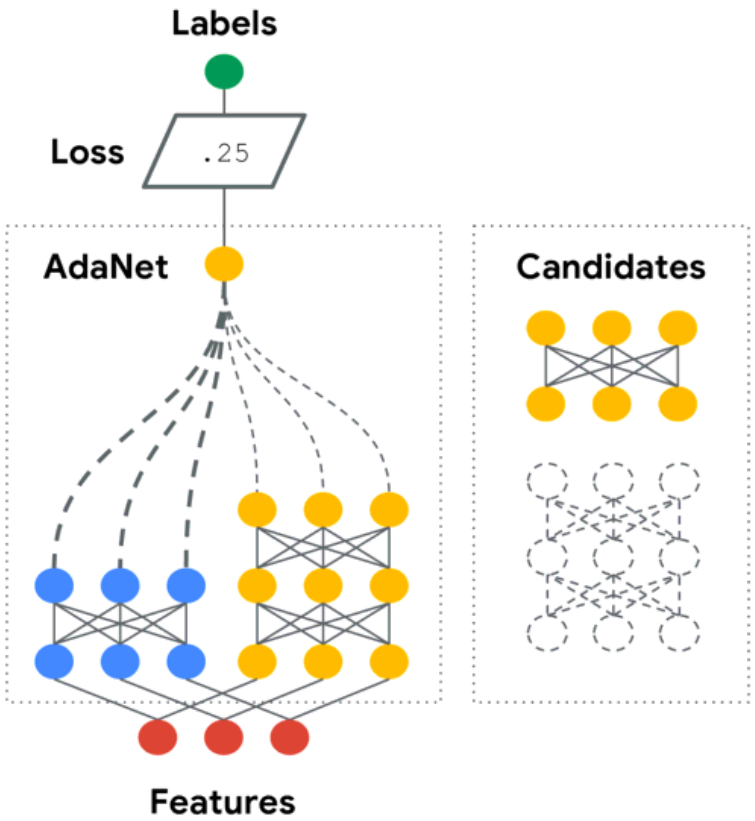
Pixels as image feature

- Confusion matrix

$$\begin{bmatrix} 883 & 0 & 25 & 3 & 4 & 28 & 23 & 3 & 3 & 8 \\ 0 & 1070 & 3 & 8 & 3 & 1 & 4 & 25 & 21 & 0 \\ 30 & 35 & 596 & 32 & 18 & 8 & 208 & 25 & 75 & 5 \\ 28 & 32 & 19 & 678 & 2 & 92 & 30 & 33 & 74 & 22 \\ 4 & 2 & 17 & 14 & 708 & 16 & 10 & 80 & 35 & 96 \\ 29 & 32 & 6 & 122 & 27 & 526 & 22 & 19 & 71 & 38 \\ 20 & 10 & 35 & 6 & 26 & 32 & 822 & 1 & 6 & 0 \\ 7 & 16 & 23 & 8 & 14 & 7 & 1 & 804 & 20 & 128 \\ 40 & 48 & 11 & 91 & 15 & 34 & 22 & 18 & 661 & 34 \\ 9 & 11 & 23 & 32 & 161 & 18 & 1 & 169 & 34 & 551 \end{bmatrix}$$

- does it means sens ?
- is it better ? worse ?

boosting and deep learning



AdaNet : `https://github.com/tensorflow/adanet`