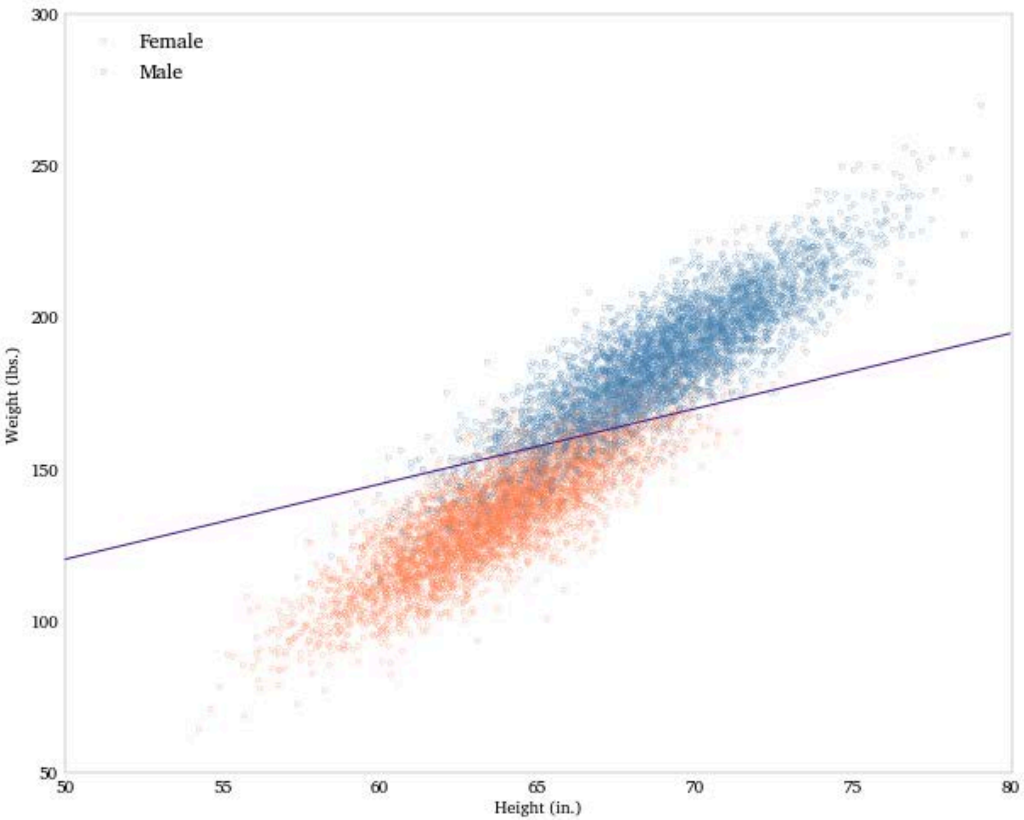
Boosting

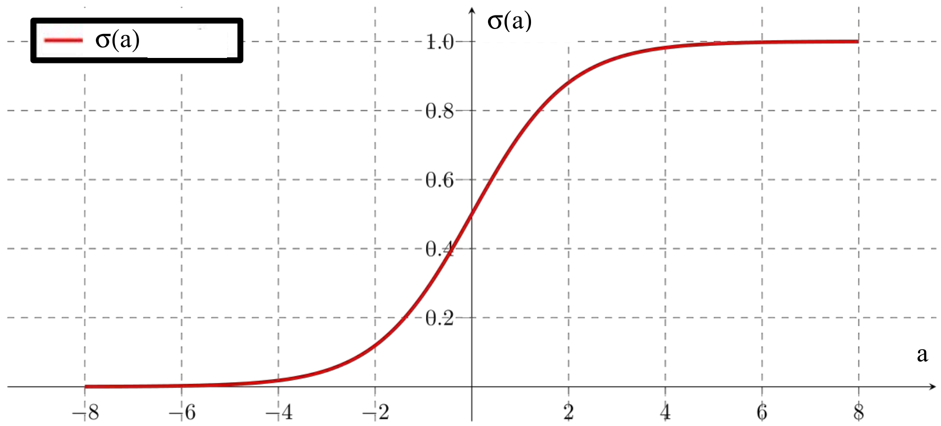
Pascal Fua
IC-CVLab

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Logistic Regression



$$y(\mathbf{x}; \mathbf{w}, w_0) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$$
$$\approx p(t = 1, \mathbf{x})$$

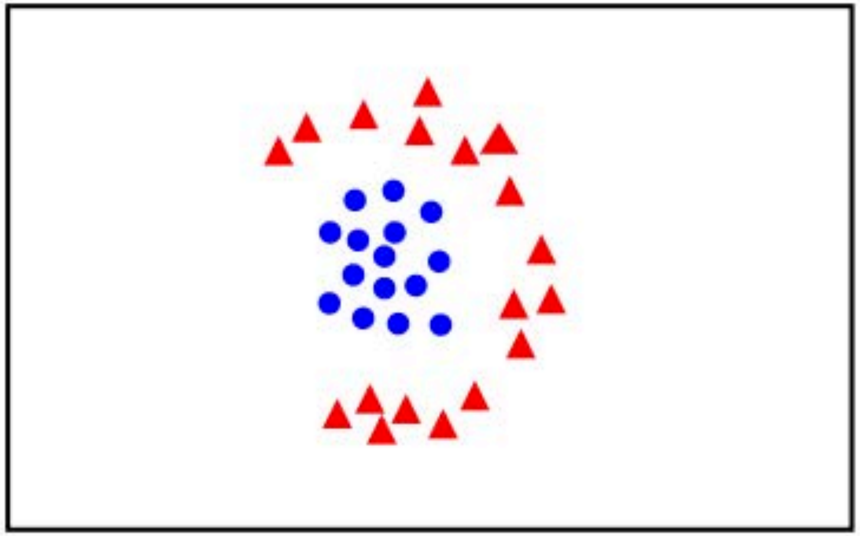


Given the training set $\{(x_n, t_n)_{1 \leq n \leq N}\}$, choose a $\mathbf{w}$ that minimizes

$$E(\mathbf{w}, w_0) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \approx -\ln(p(\mathbf{t}|\mathbf{w}, w_0)) \ .$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Classifying Non Linearly Separable Data

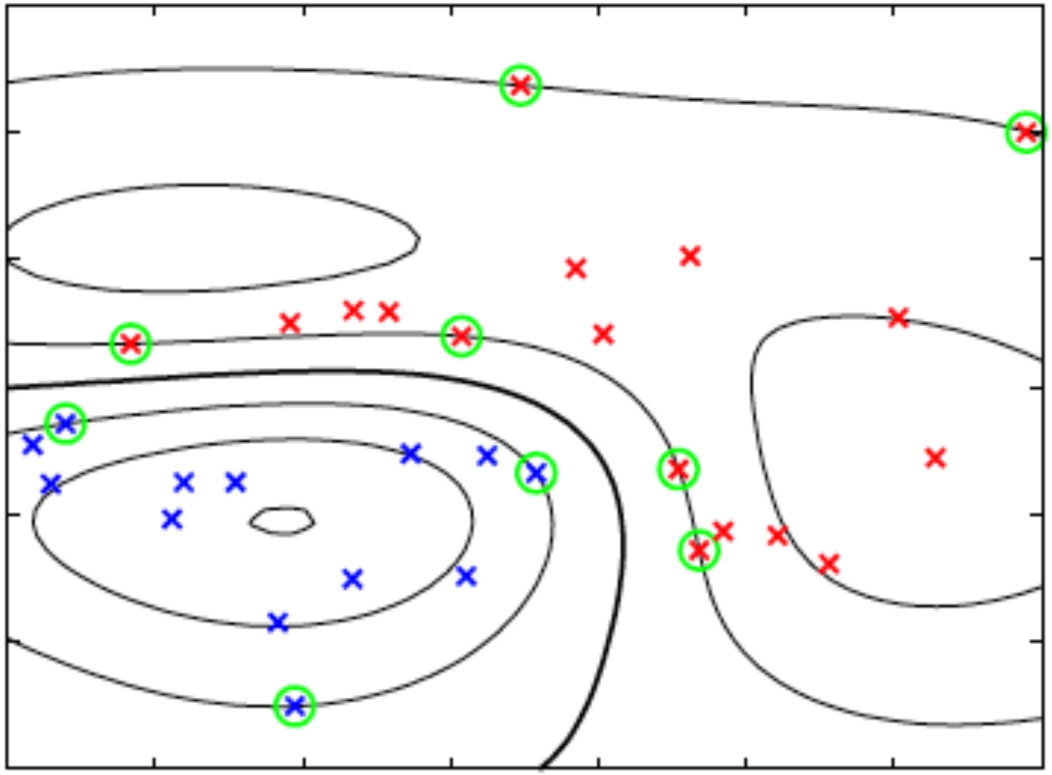


Map it to a higher dimension!

$$f(\mathbf{x}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}) + w_0)$$

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$$

Kernel Trick



$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b ,$$

- Only for a subset of the data points is a_n is non zero.
- The feature vector $\phi(\mathbf{x})$ does **not** appear explicitly anymore.
- The kernel function k(.) can be understood as a similarity measure.
- Training accuracy can be traded for larger margins.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Very elegant BUT complexity in $O(N^3)$.

Fashions in Science

Support Vector Machines (SVMs) are a relatively new concept in supervised learning, but since the publication of [4] in 1995 they have been applied to a wide variety of problems. In many ways the application of SVMs to almost any learning problem mirrors the enthusiasm (and fashionability) that was observed for neural networks in the second half of the 1980's.

C. Williams'08

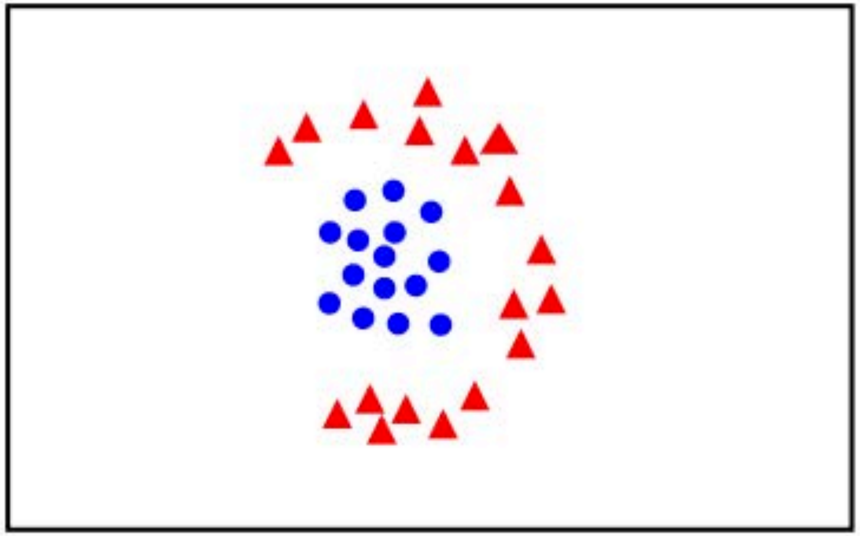
In many ways the application of Deep Neural Networks to almost any learning problem mirrors the enthusiasm (and fashionability) that was observed for SVMs in the second half of the 1990's.

😏 '18

In many ways the application of XXXX to almost any learning problem mirrors the enthusiasm (and fashionability) that was observed for neural networks in the second half of the 2010's.

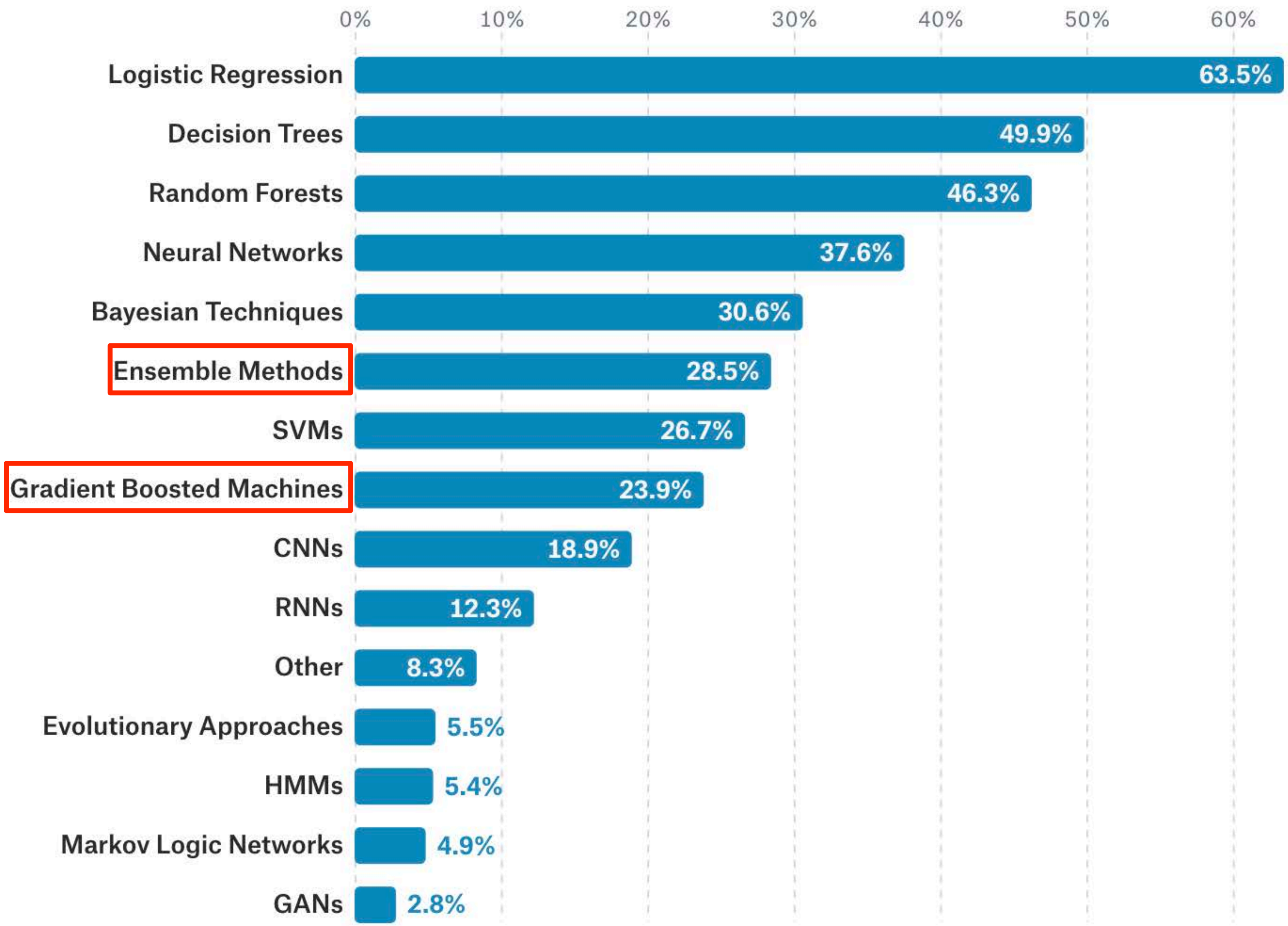
🤔 '38

Classifying Non Linearly Separable Data



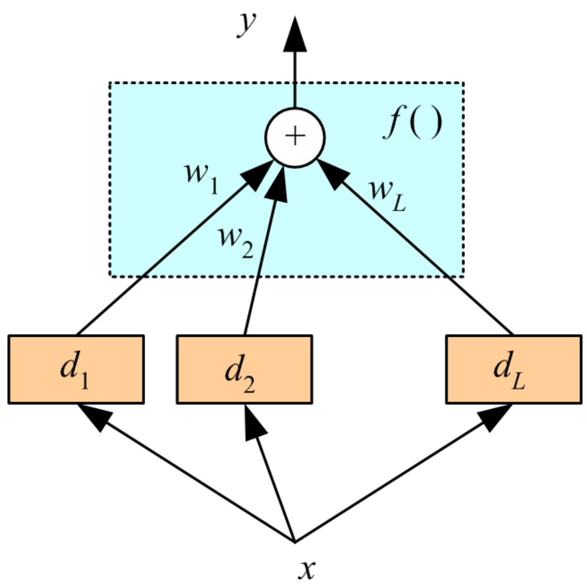
- Map it to a higher dimension.
- Combine multiple linear classifiers.
- Use decision trees.
- Use deep networks.

Boosting Methods



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Combining Linear Classifiers



- Use the linear classifiers as “weak” classifiers, that is, classifiers operating only slightly better than chance.
- Write a strong classifier as a weighted sum of weak ones.

Ada Boost

Iteratively building a weighted sum of weak classifiers:

$$y(\mathbf{x}) = \alpha_1 y_1(\mathbf{x}) + \alpha_2 y_2(\mathbf{x}) + \alpha_3 y_3(\mathbf{x}) + \ldots$$

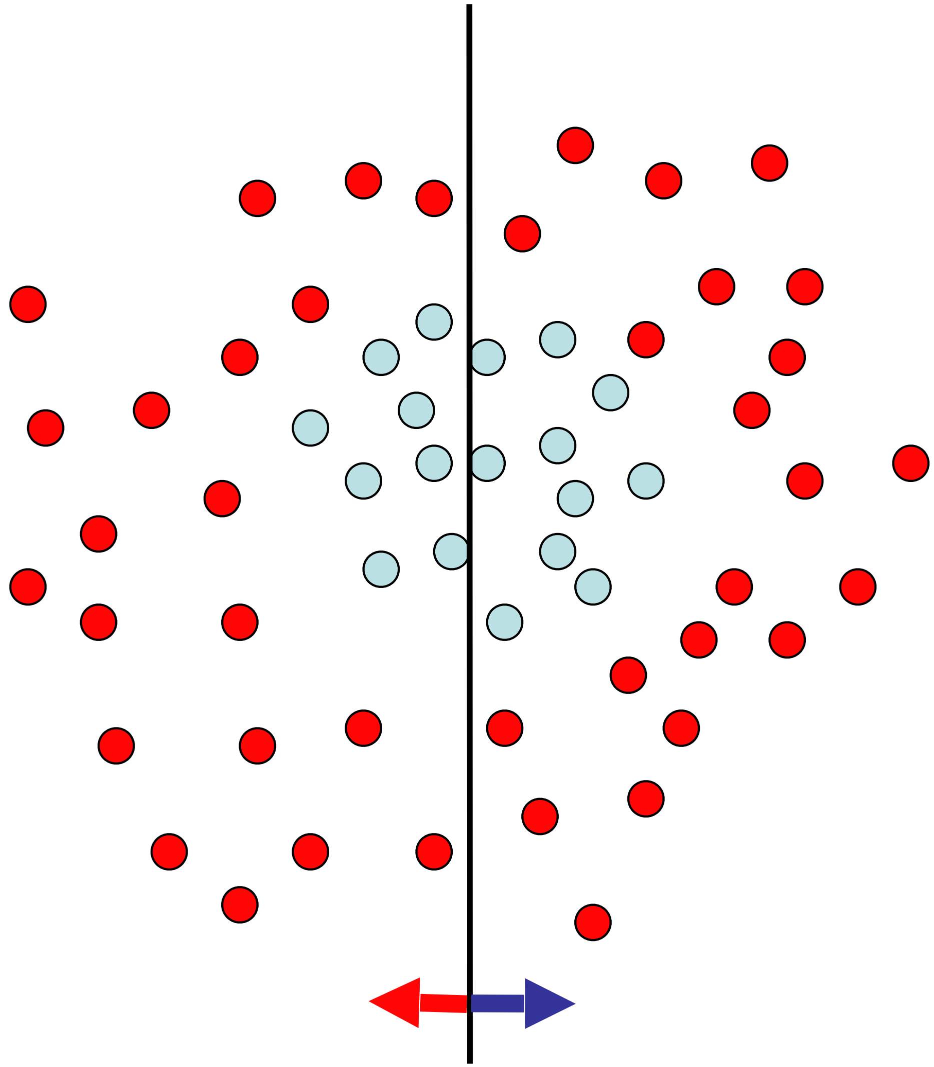
Weak classifier

Strong classifier

Weight

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Toy Example



Each data point has a class label:

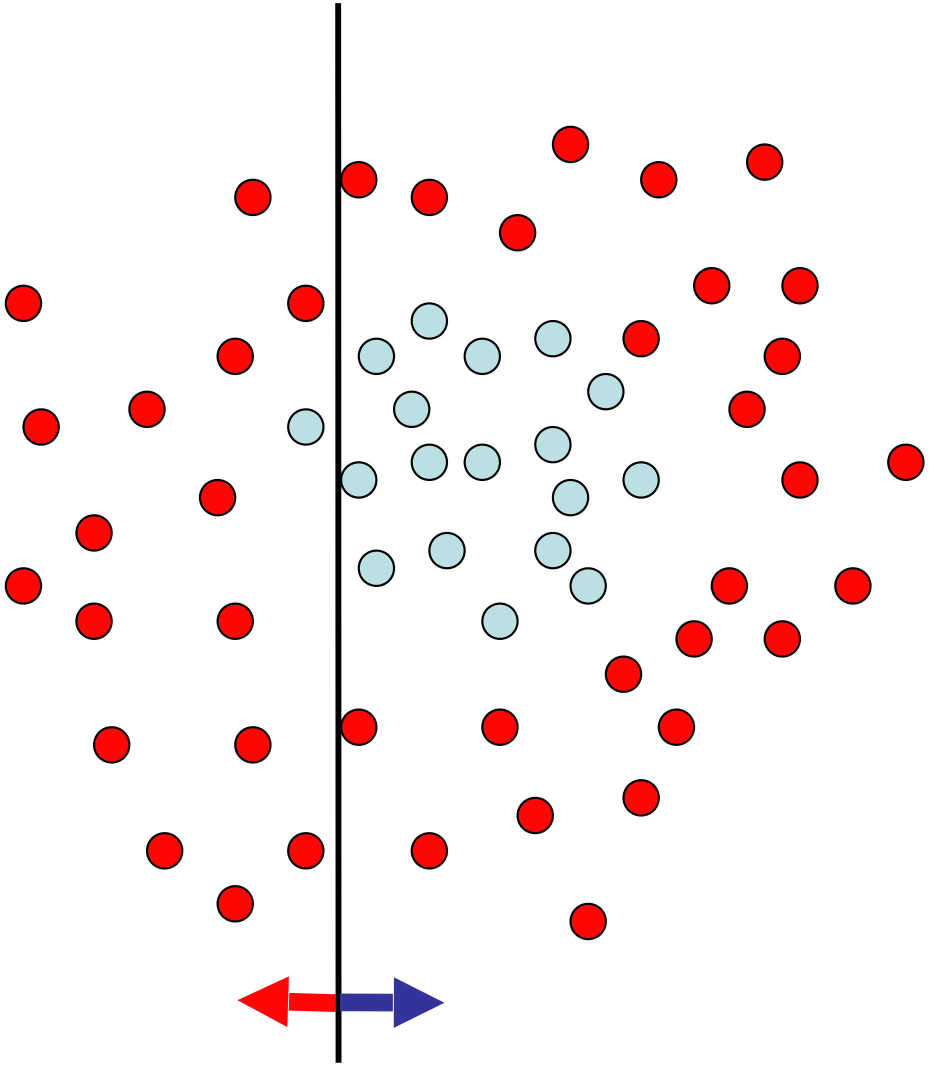
$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{light blue}) \end{cases}$$

and a weight:

$$d_t = 1$$

Classifier is roughly at chance.

Toy Example



Each data point has a class label:

$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{light blue}) \end{cases}$$

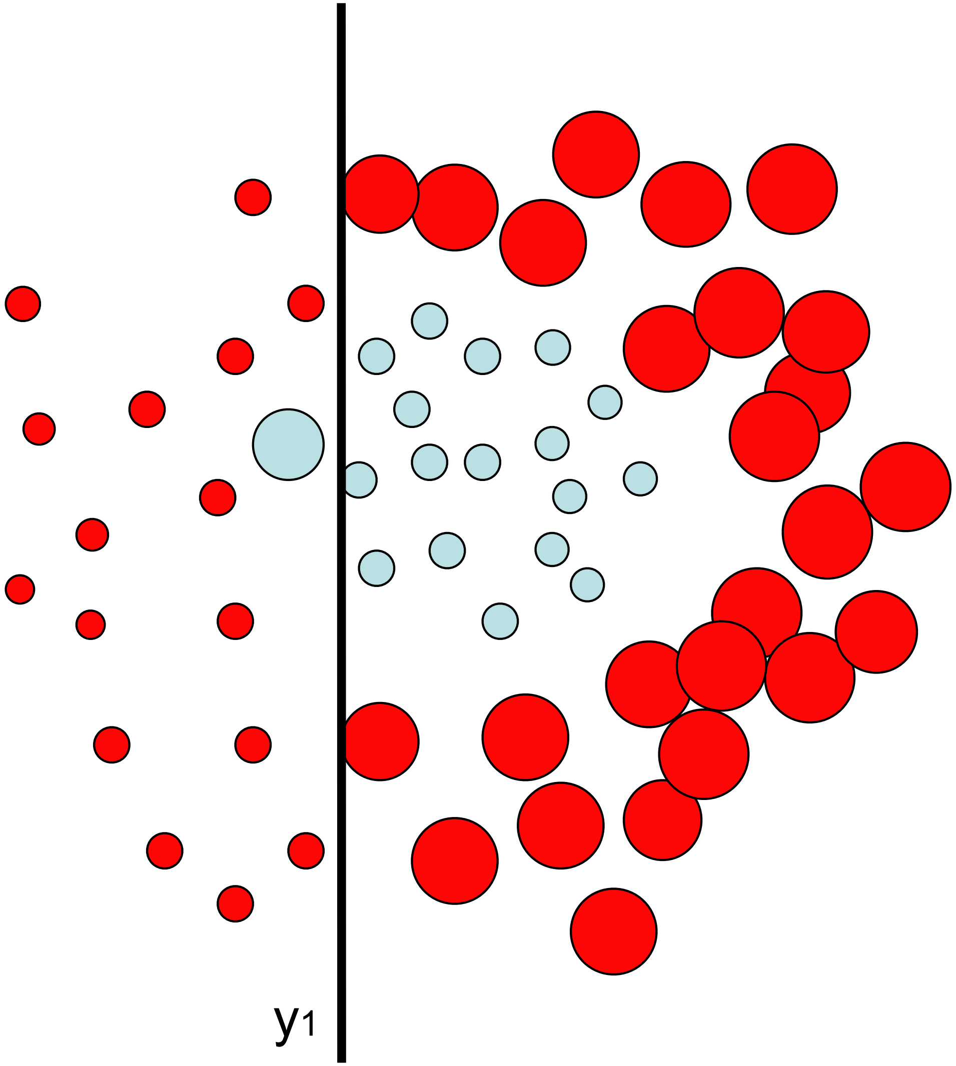
and a weight:

$d_t = 1$

Classifier is now slightly better than chance.
It becomes y_1.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Toy Example



Each data point is given a new class label

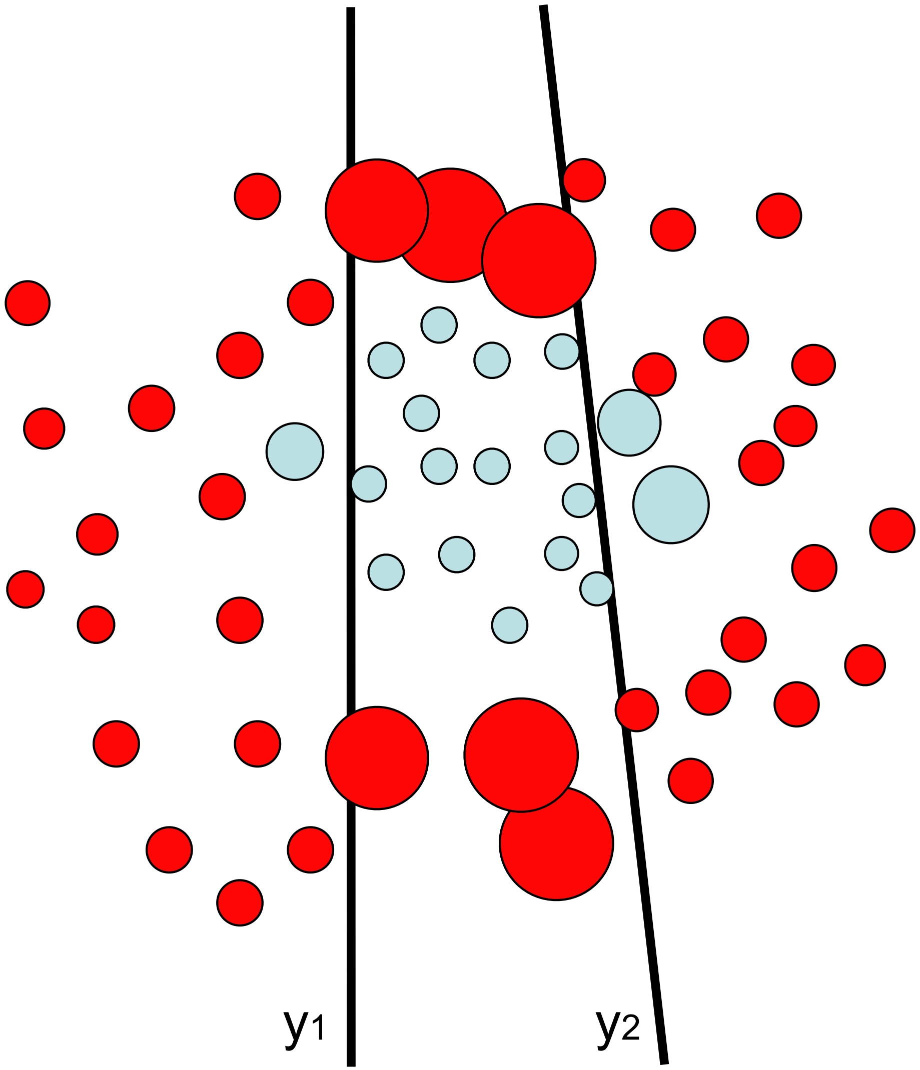
$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{light blue}) \end{cases}$$

and a new weight

d_t

d_t is chosen so that the classifier operates at chance again.

Toy Example



Each data point is given a new class label

$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{light blue}) \end{cases}$$

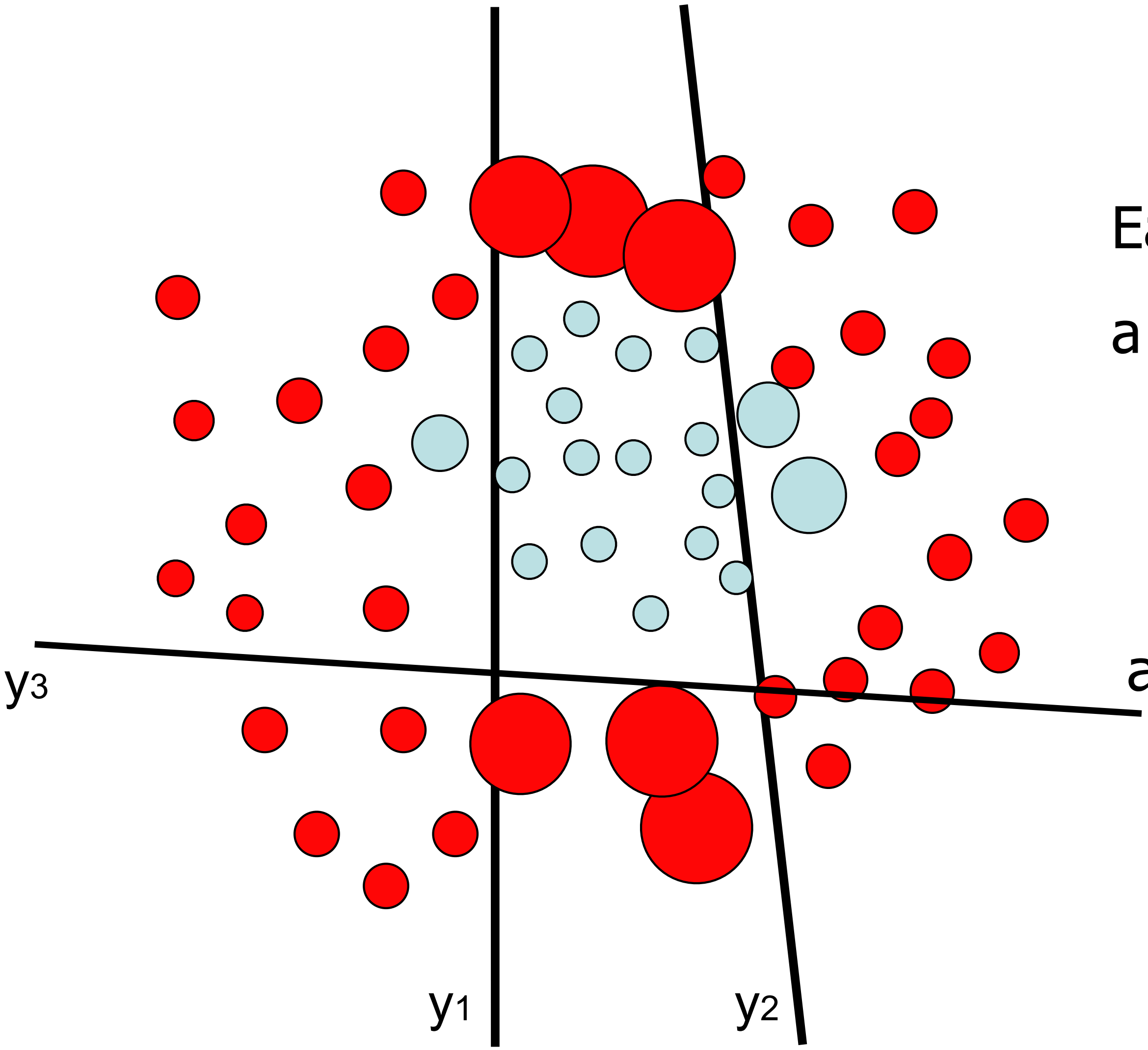
and a new weight

d_t

Find a new classifier y2 and reset the weights again.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Toy Example



Each data point is given a new class label

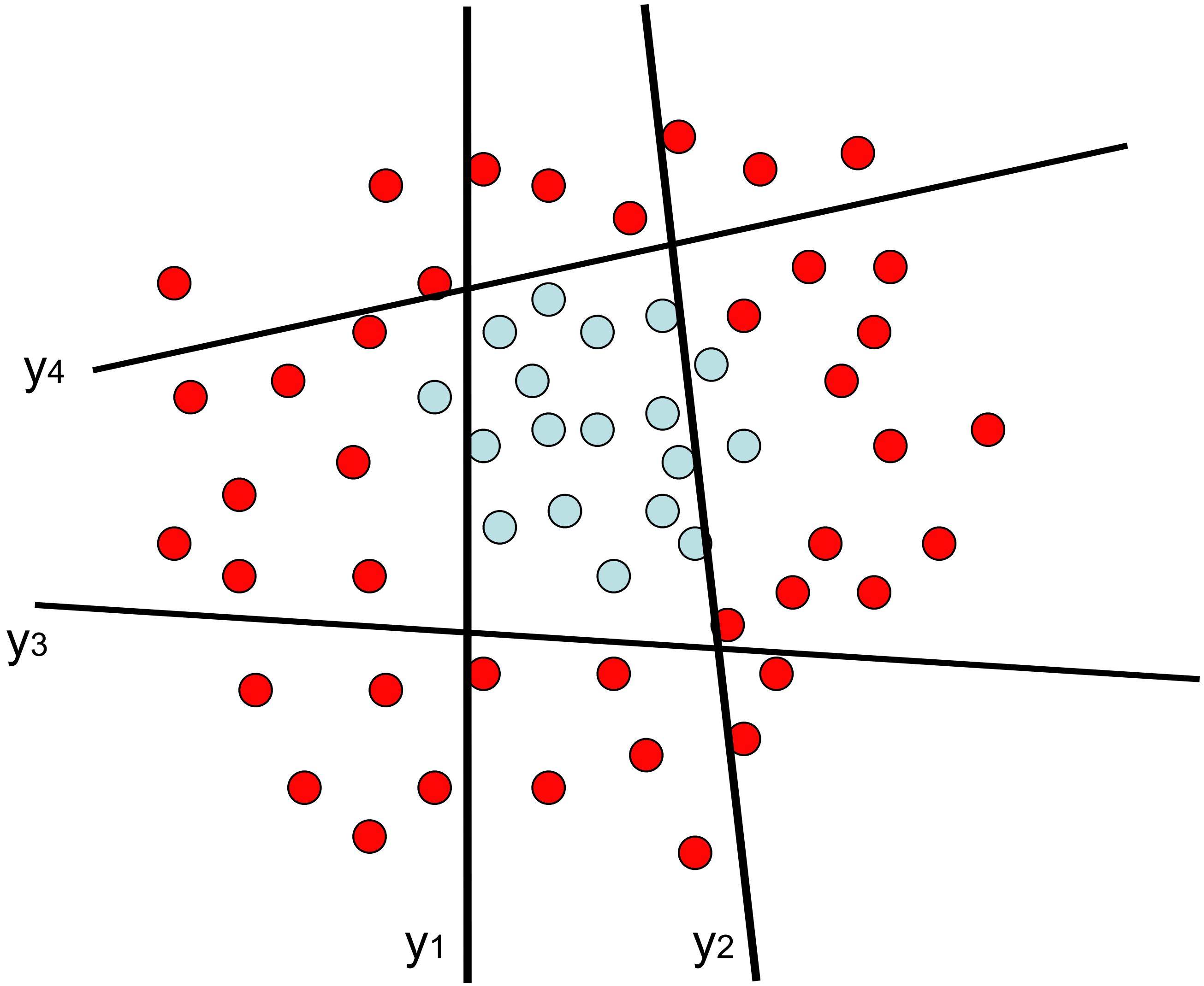
$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{light blue}) \end{cases}$$

and a new weight

$\mathbf{d_t}$

Find a new classifier y3 and reset the weights again.

Toy Example



$$y(\mathbf{x}) = \alpha_1 y_1(\mathbf{x}) + \alpha_2 y_2(\mathbf{x}) + \alpha_3 y_3(\mathbf{x}) + \alpha_4 y_4(\mathbf{x})$$

Adaboost Algorithm

For a training set $\chi = \{\mathbf{x}_n, t_n\}$ where $t_n \in$ {-1,1} for $1 \leq n \leq$N:

1. Initialize data weights: $\forall n, w_n^1 = 1/N$.
2. For $t = [1, \ldots, T]$:
   (a) Find classifier $y_t : \chi \to \{-1, 1\}$ that minimizes weighted error $\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$.
   (b) Evaluate

$$\epsilon_t = \frac{\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t}{\sum_{n=1}^{N} w_n^t}$$

Inferior to 0.5 if y_t operates at better than chance.

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

Positive if y_t operates at better than chance.

   (c) Update weights

$$w_n^{t+1} = w_n^t \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n)))$$

The weight of misclassified samples is increased.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

→ **Final classifier:** $Y(\mathbf{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t y_t(\mathbf{x}))$

Exponential Loss

$$E_t = \sum_{n=1}^{N} \exp(-t_n f_t(\mathbf{x}_n))$$

$$f_t(\mathbf{x}) = \frac{1}{2} \sum_{s=1}^{t} \alpha_s y_s(\mathbf{x})$$

Adaboost is designed to minimize this loss.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Proof Sketch (1)

At iteration t, given $y_1, \ldots, y_{t-1}$ and $\alpha_1, \ldots, \alpha_{t-1}$, minimize

$$E_t = \sum_{n=1}^{N} \exp(-t_n(f_{t-1}(\mathbf{x}_n) + \frac{1}{2}\alpha_t y_t(\mathbf{x}_n))))$$

$$= \sum_{n=1}^{N} w_n^t \exp(-\frac{\alpha_t}{2} t_n y_t(\mathbf{x}_n))$$

with respect to y_t and α_t.

—> Adaboost performs a form of gradient descent on the exponential loss.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Proof Sketch (2)

Minimizing $\sum_{n=1}^{N} w_n^t \exp(-\frac{\alpha_t}{2} t_n y_t(\mathbf{x}_n))$ w.r.t. to y_t and α_t yields:

$$y_t \quad \text{must minimize} \quad \sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$$

$$\alpha_t = \log(\frac{1-\epsilon_t}{\epsilon_t}) \text{ with } \epsilon_t = \frac{\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t}{\sum_{n=1}^{N} w_n^t}$$

$$w_n^{t+1} = w_n^t \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n)))$$

—> Adaboost performs a form of gradient descent on the exponential loss.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Adaboost Algorithm

For a training set $\chi = \{\mathbf{x}_n, t_n\}$ where $t_n \in$ {-1,1} for 1 $\leq n \leq$N:

1. Initialize data weights: $\forall n, w_n^1 = 1/N$.
2. For $t = [1, \ldots, T]$:
   (a) Find classifier $y_t : \chi \to \{-1, 1\}$ that minimizes weighted error $\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$.
   (b) Evaluate

$$\epsilon_t = \frac{\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t}{\sum_{n=1}^{N} w_n^t}$$

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

   (c) Update weights

$$w_n^{t+1} = w_n^t \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n)))$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

→ **Final classifier:** $Y(\mathbf{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t y_t(\mathbf{x}))$

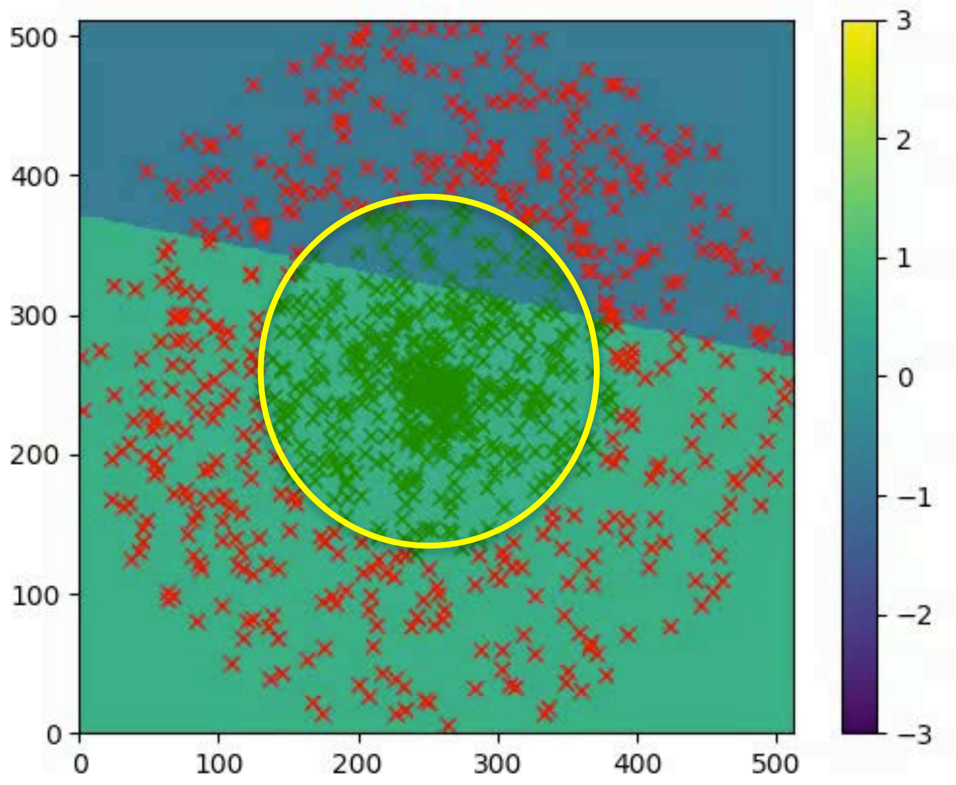
Adabost in Python

```
def fit(self,nit=10):
    # Initialize weights and list of classifiers
    self.weakCls  = []
    bestAcc = 0.0
    self.datCoeffs = np.ones(self.ns,dtype=np.float)/self.ns
    # Find nit weak classifiers and update weights each time.
    for m in range(nit):
        weakC=self.getWeakC()
        self.weakCls.append(weakC)
        weakC.alpha=self.updateWeights(weakC)
```

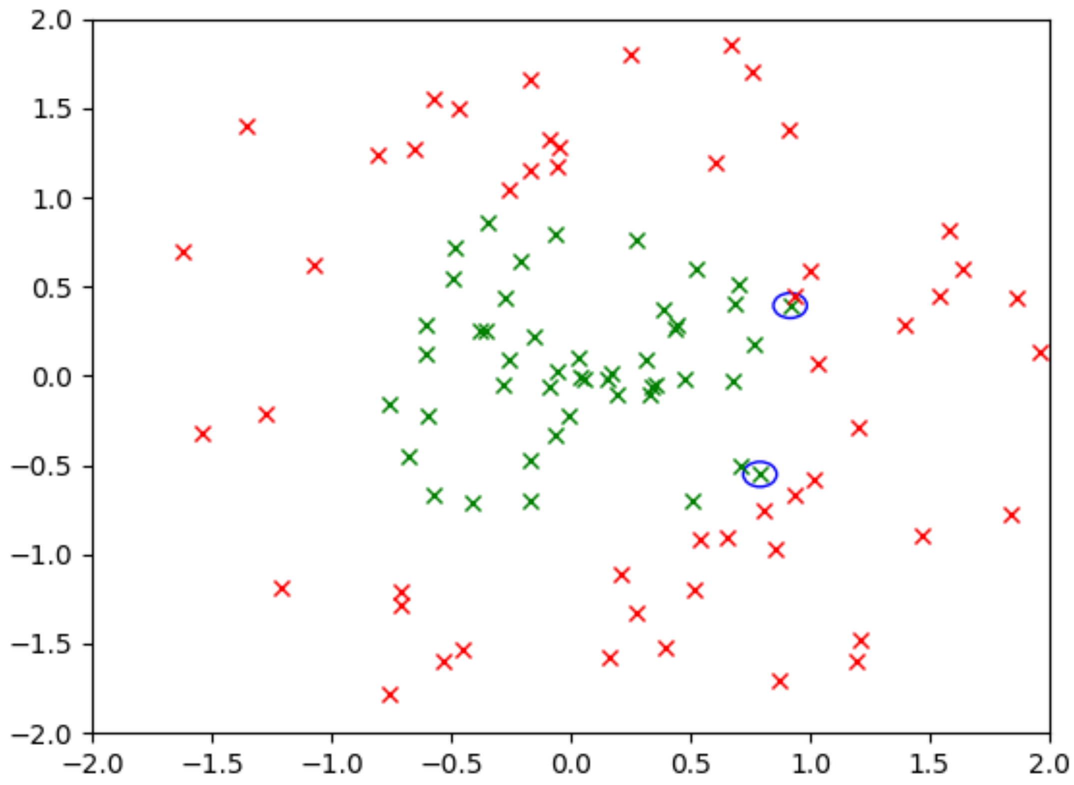
```
def updateWeights(self,weakC):
    # Compute alpha
    err,_ = self.weakClassError(weakC)
    alpha = np.log(1.0/max(1e-10,err)-1.0)
    # Compute numbers of misclassified samples.
    nerrs = np.logical_not(weakC.predict(self.xs)==self.ys)
    # Update and normalize weights.
    self.datCoeffs *=  np.exp(alpha*nerrs)
    self.datCoeffs /=  sum (self.datCoeffs)
    return alpha
```

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Circular Distribution



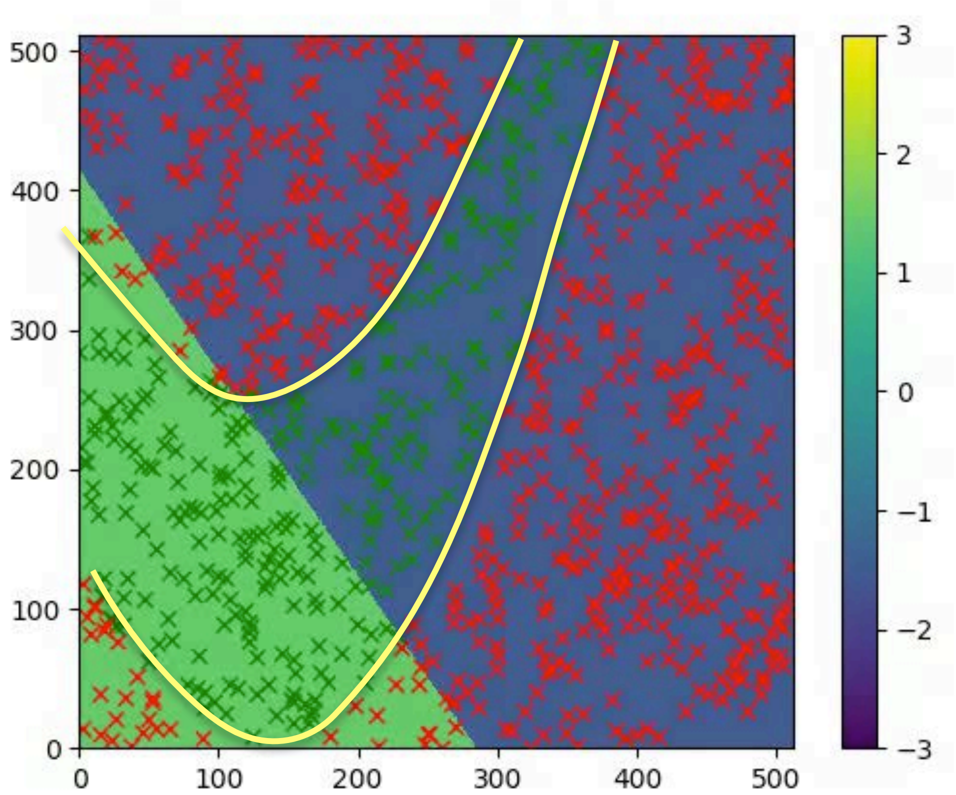
Training (100 iterations)



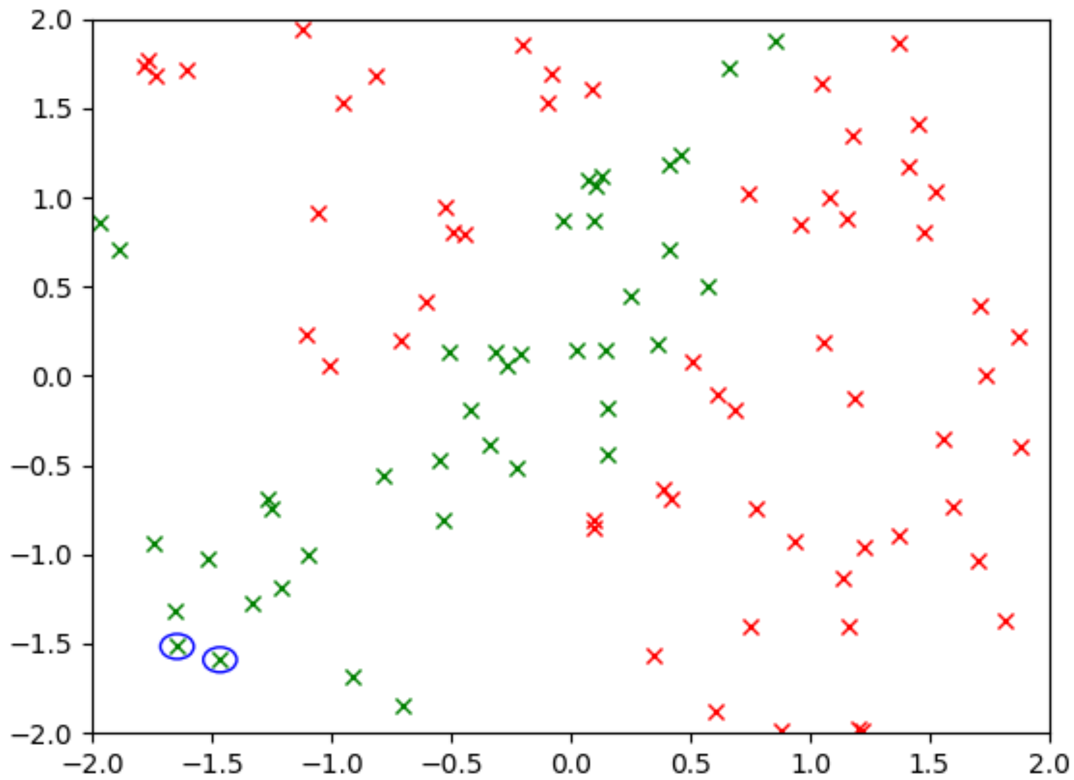
Validation (98% accuracy)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Rosenbrock Function



Training (100 iterations)



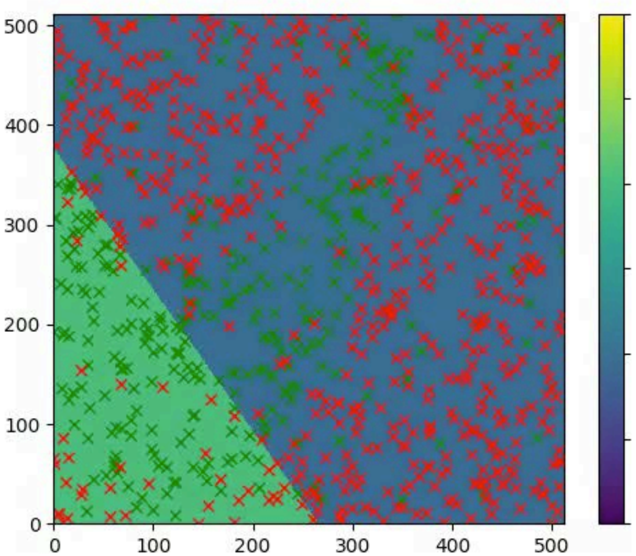
Validation (98% accuracy)

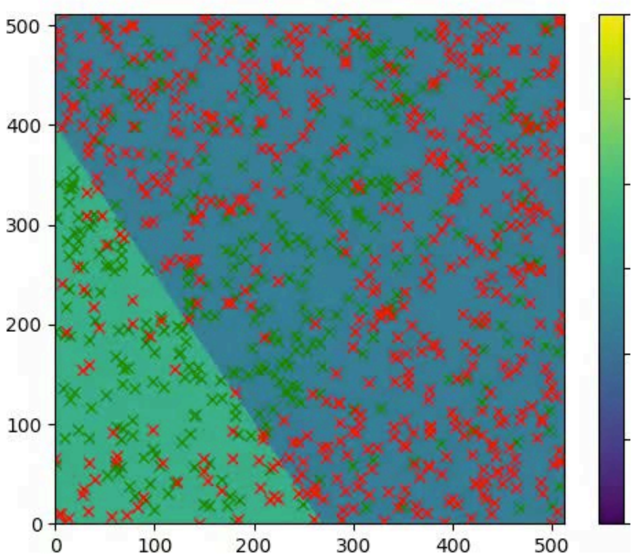
$$r(x,y) = 100 * (y - x^2)^2 + (1-x)^2$$

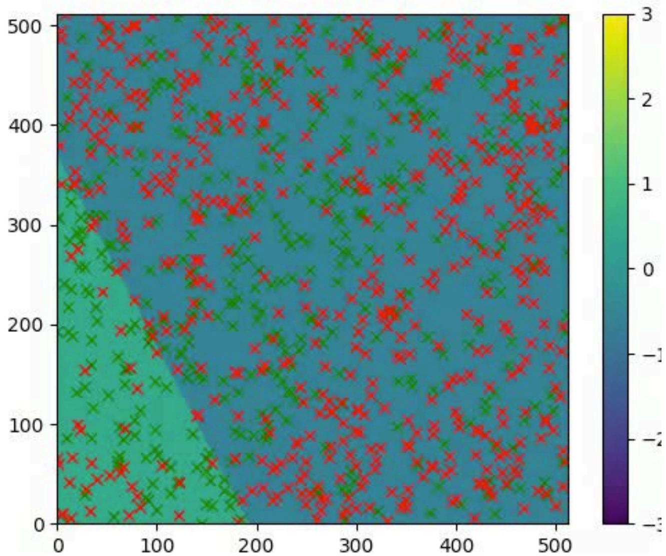
$$f(x,y) = \begin{cases} \text{-1} & \text{if } r(x,y) < \text{T} \\ 1 & \text{otherwise} \end{cases}$$

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Noisy Labels



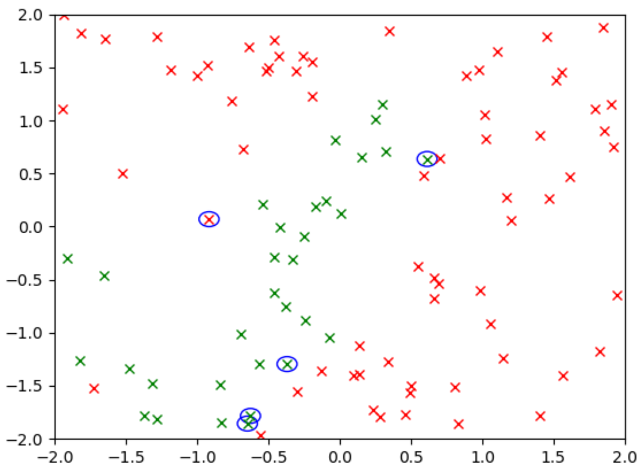


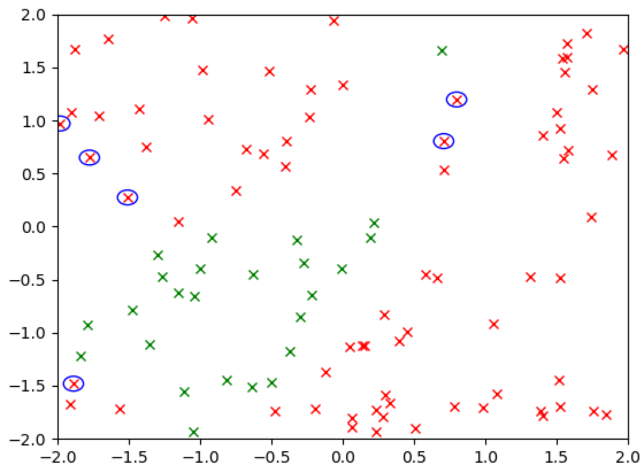


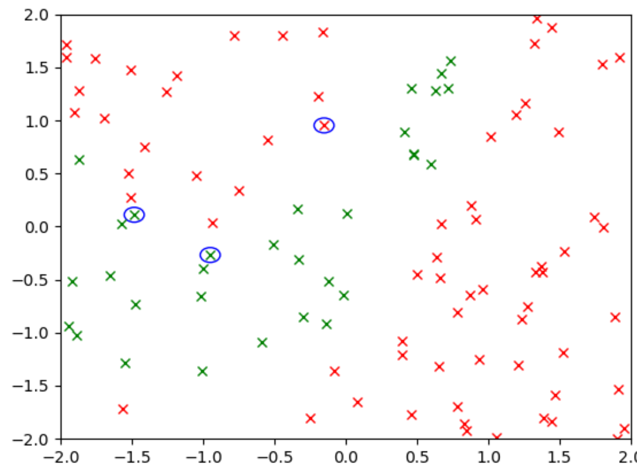
10% mislabeled

20% mislabeled

30% mislabeled



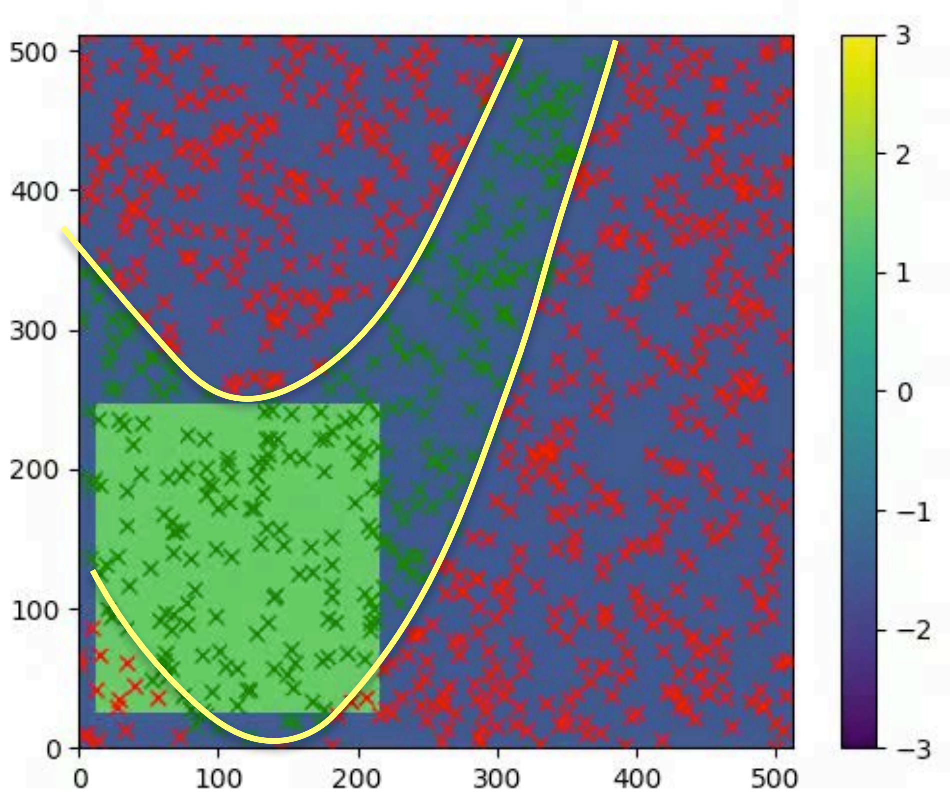




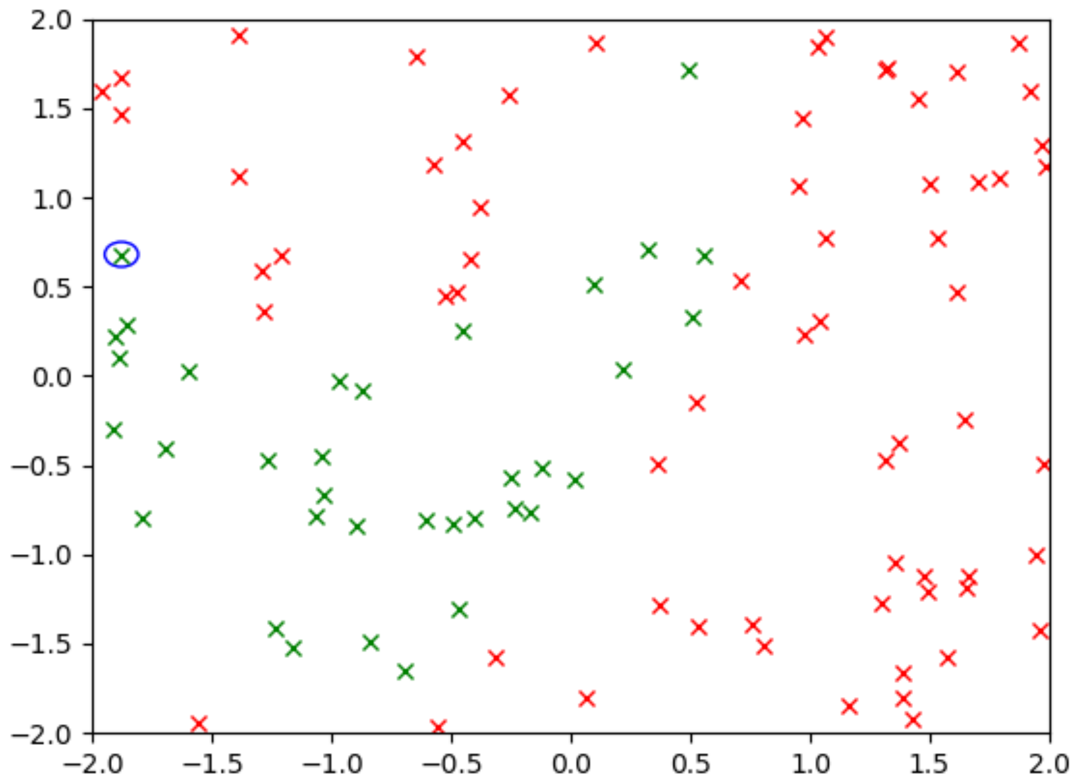
The incorrect labels have relatively little impact because they are randomly distributed, but they could.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Changing the Weak Learners



Training (100 iterations)



Validation (99% accuracy)

$$y(\mathbf{x};\mathbf{w}) = \begin{cases} 1 & \text{if} \quad x_0 < \mathbf{x}[1] < x_1 \quad \text{and} \quad y_0 < \mathbf{x}[2] < y_1 \ , \\ -1 & \text{otherwise.} \end{cases}$$

$$\mathbf{w} = (x_0, y_0, x_1, y_1)$$

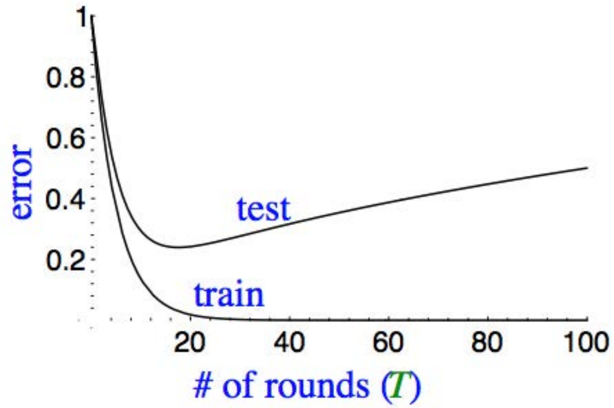
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Training and Testing Errors

- The training error goes down exponentially fast if the weighted errors ε_t of the component classifiers is always strictly inferior to 0.5.

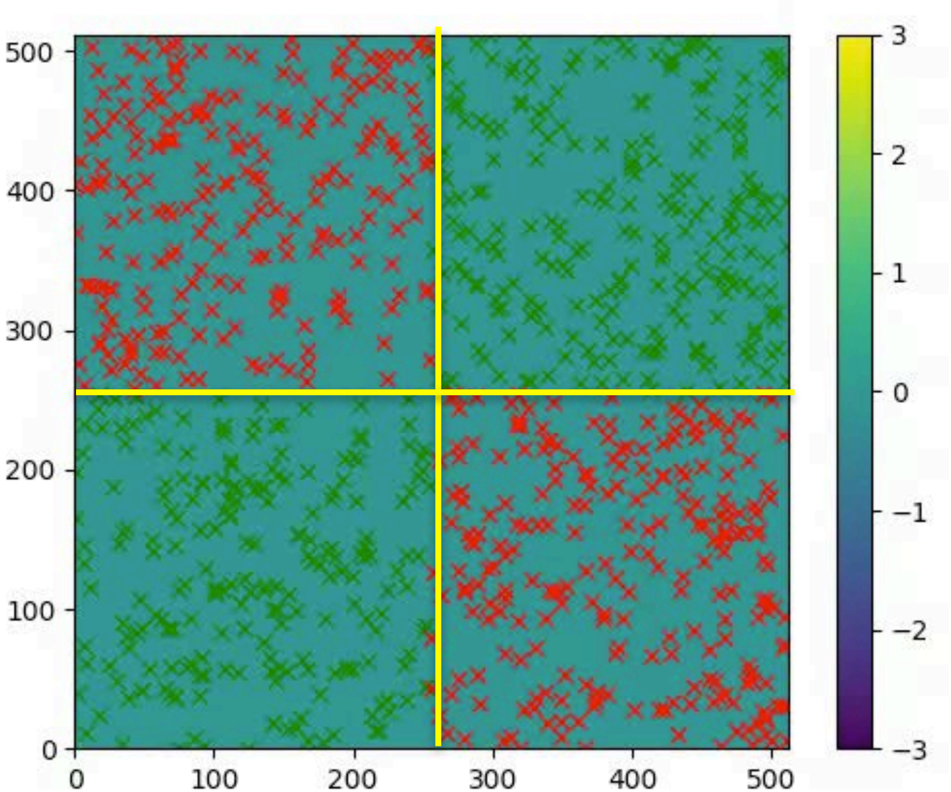
$$\frac{1}{N}\sum_n [t_n \neq h(\mathbf{x}_n)] < \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}$$

- The testing error may eventually go up due to overfitting.

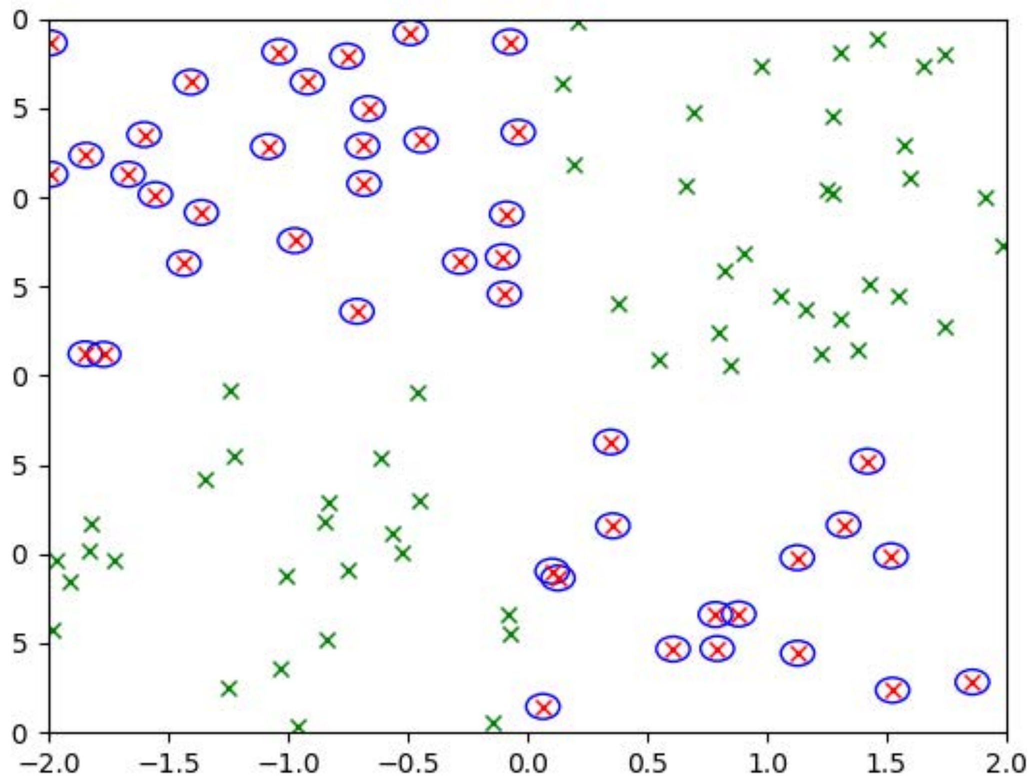


—> Use a validation set.

Failure Mode



Training (100 iterations)



Validation (56% accuracy)

Individual weak classifiers cannot do better than chance!

Adaboost in Python

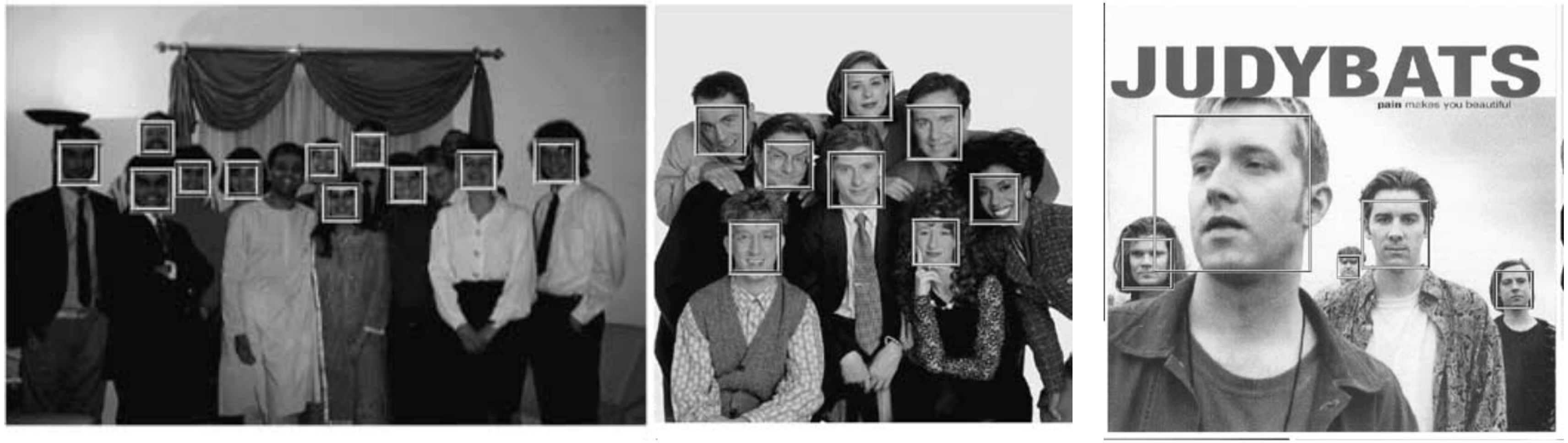
```
def fit(self,nit=10):
    # Initialize weights and list of classifiers
    self.weakCls  = []
    bestAcc = 0.0
    self.datCoeffs = np.ones(self.ns,dtype=np.float)/self.ns
    # Find nit weak classifiers and update weights each time.
    for m in range(nit):
        weakC=self.getWeakC()
        self.weakCls.append(weakC)
        weakC.alpha=self.updateWeights(weakC)
```

```
def updateWeights(self,weakC):
    # Compute alpha
    err,_ = self.weakClassError(weakC)
    alpha = np.log(1.0/max(1e-10,err)-1.0)
    # Compute numbers of misclassified samples.
    nerrs = np.logical_not(weakC.predict(self.xs)==self.ys)
    # Update and normalize weights.
    self.datCoeffs *=  np.exp(alpha*nerrs)
    self.datCoeffs /=  sum (self.datCoeffs)
    return alpha
```

- A strikingly simple algorithm that works well.
- The weak classifiers do not have to be linear classifiers.

—>Versatile and generic.

Face Detection

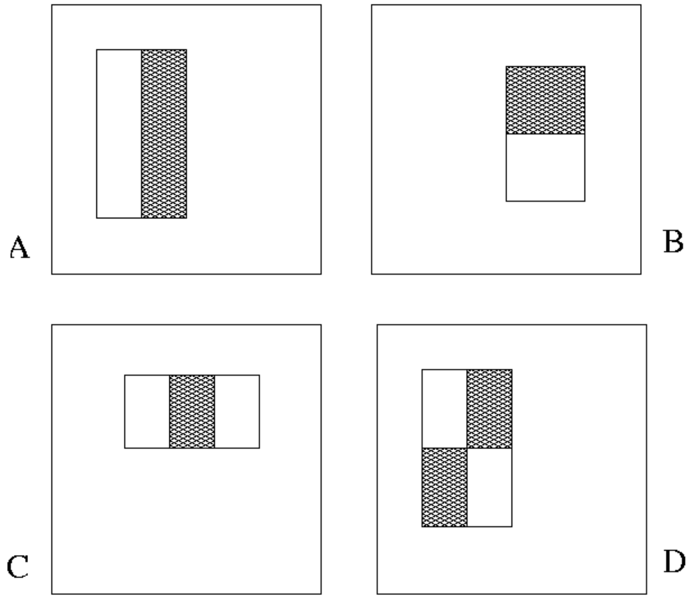


Viola & Jones, `Rapid Object Detection using a Boosted Cascade of Simple Features’, CVPR 2001:

- First reliable, real-time face detection system.
- Used in commercial products, such as digital cameras.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Weak Learners for Images



Value = ∑ (pixels in white area) – ∑ (pixels in black area)

Rectangle filters:

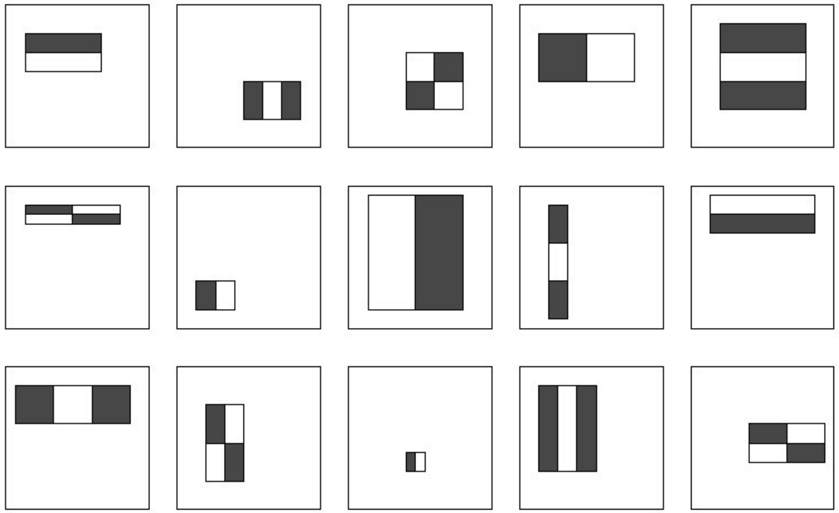
- Fast to compute (4 operations per rectangle).
- 180’000 possibilities for a 24x24 window.

—> Use adaboost to choose a good subset.

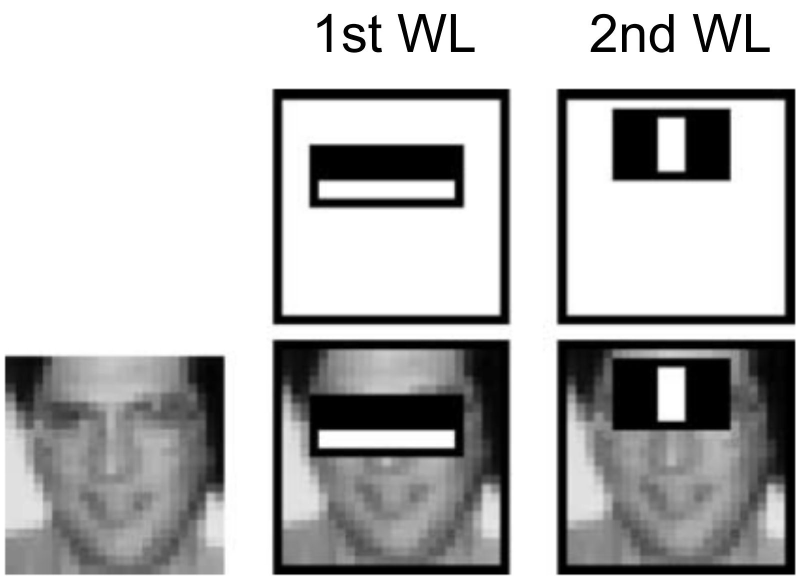
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Feature Selection

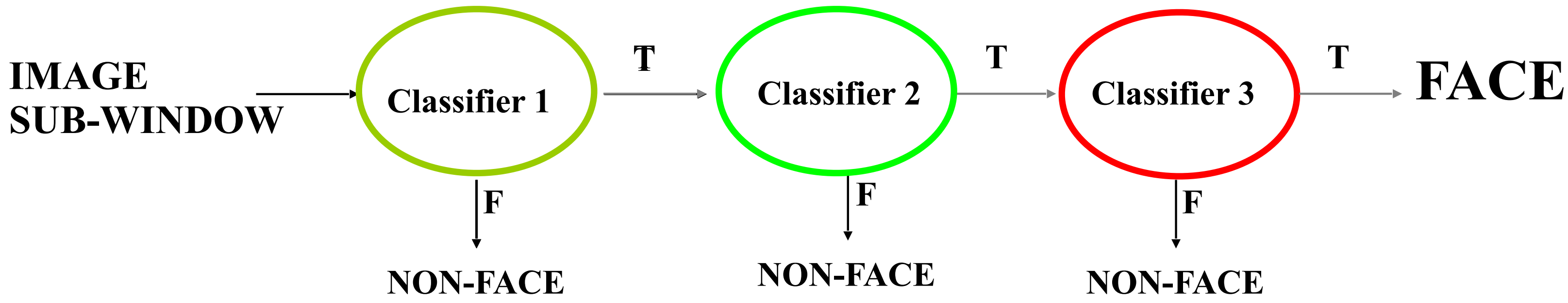
Among:



Pick:



Cascade



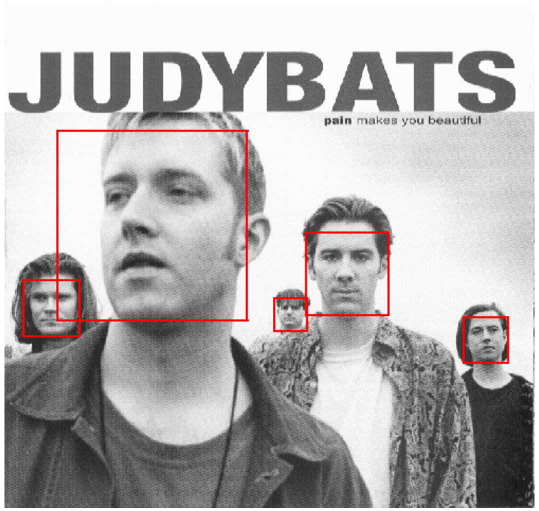
Reject large portions of the images using only the response of the first few weak classifiers —> Large potential speed-up at run-time.

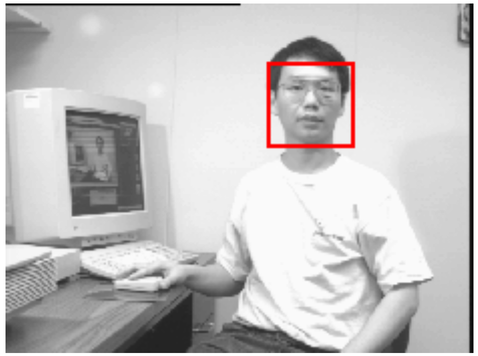
Training Set

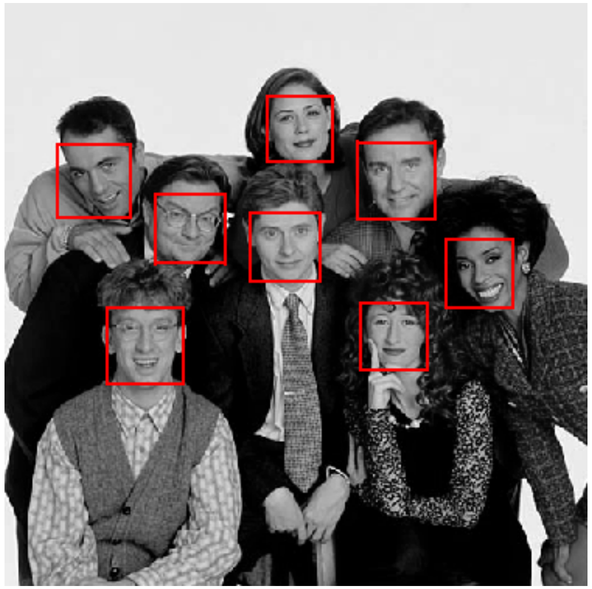
- Training Set
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 300 million non-faces
    - 9500 non-face images
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose

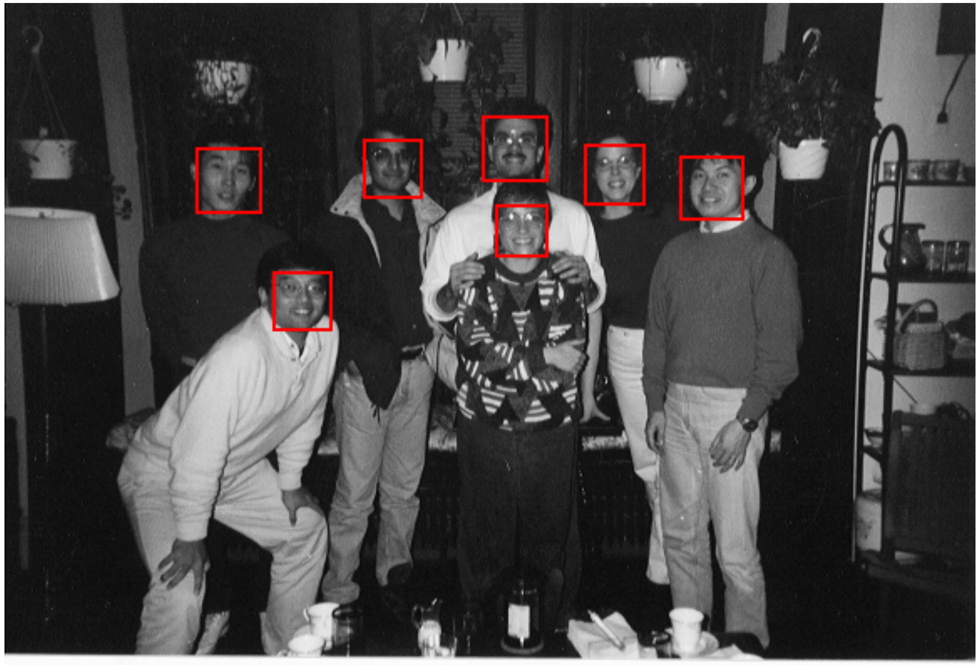
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Detection Results (2001)









ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Detection Results (2017)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE