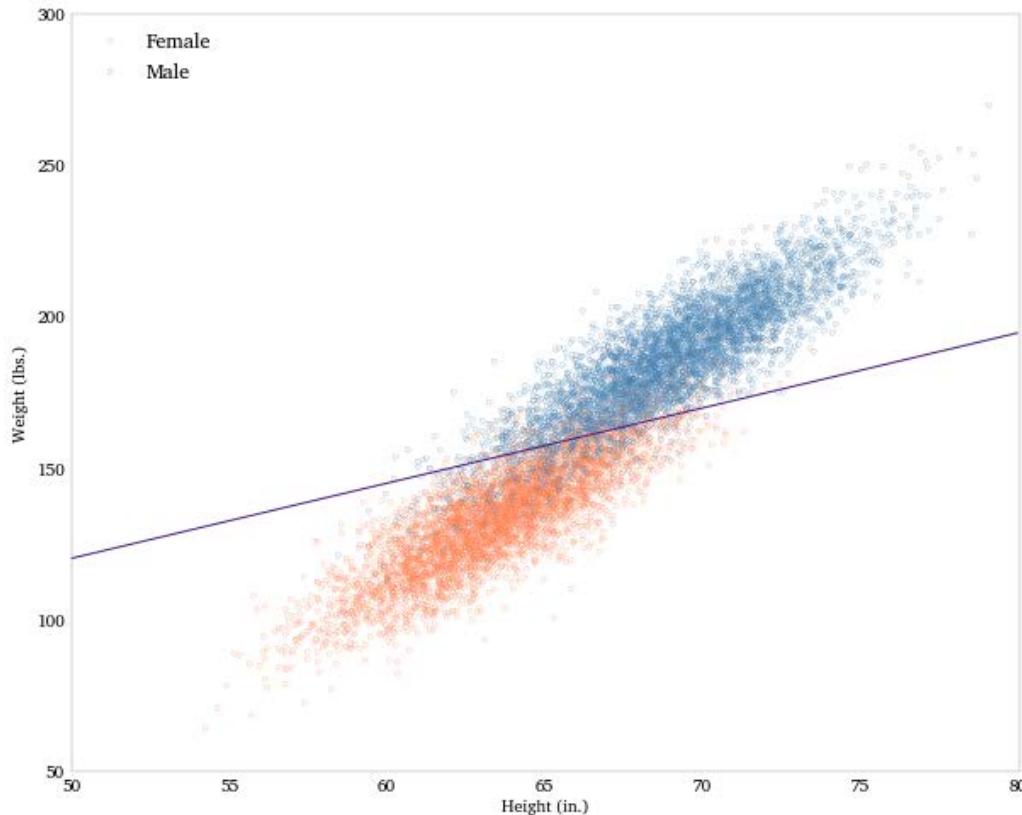


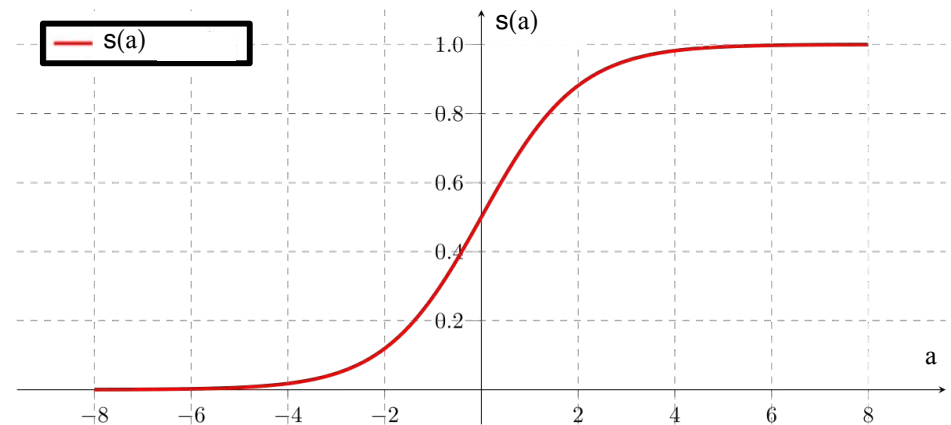
AdaBoost

Pascal Fua
IC-CVLab

Reminder: Logistic Regression



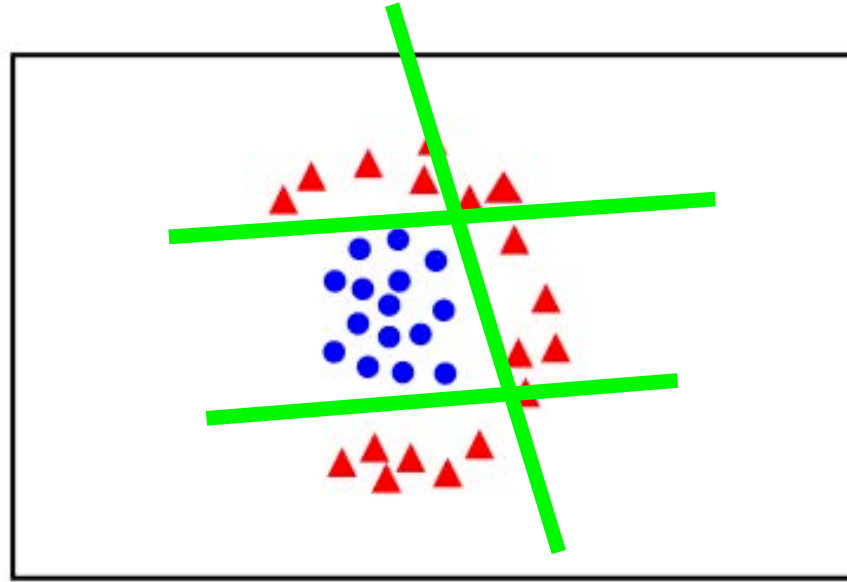
$$y(\mathbf{x}; \mathbf{w}, w_0) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \\ \approx p(t = 1, \mathbf{x})$$



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

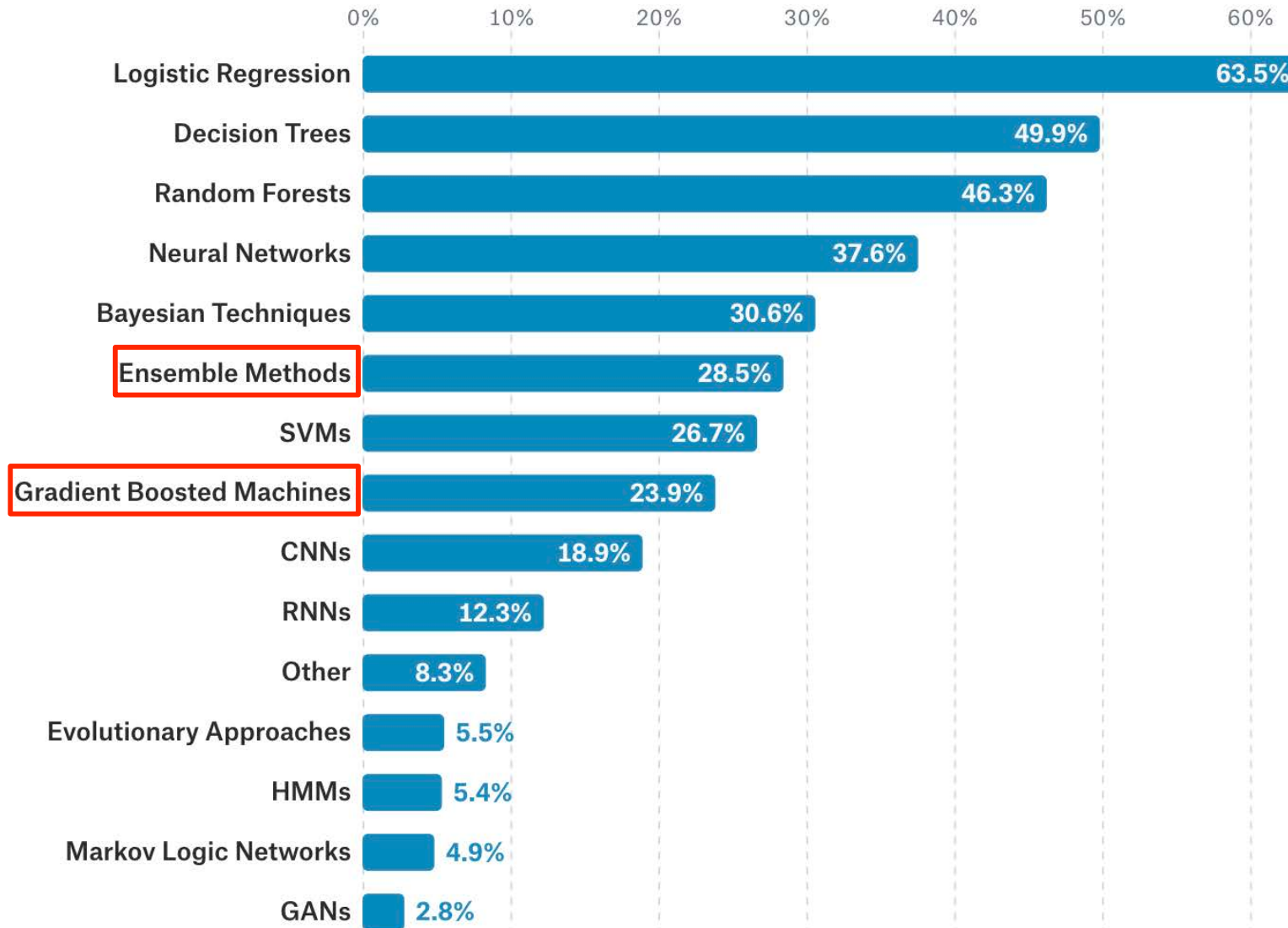
$$E(\mathbf{w}, w_0) = - \sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \approx - \ln(p(\mathbf{t} | \mathbf{w}, w_0)) .$$

Non Linearly Separable Data

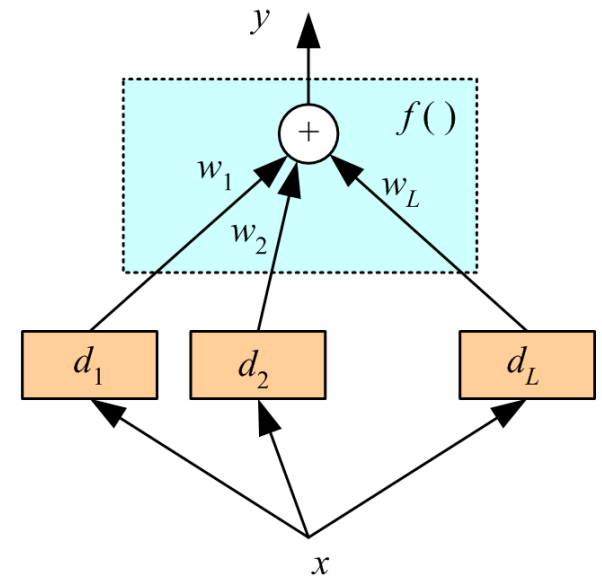


- One way is to combine multiple linear classifiers.
- We will see others in the next classes.

Boosting Methods



Combining Linear Classifiers



- Use 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.

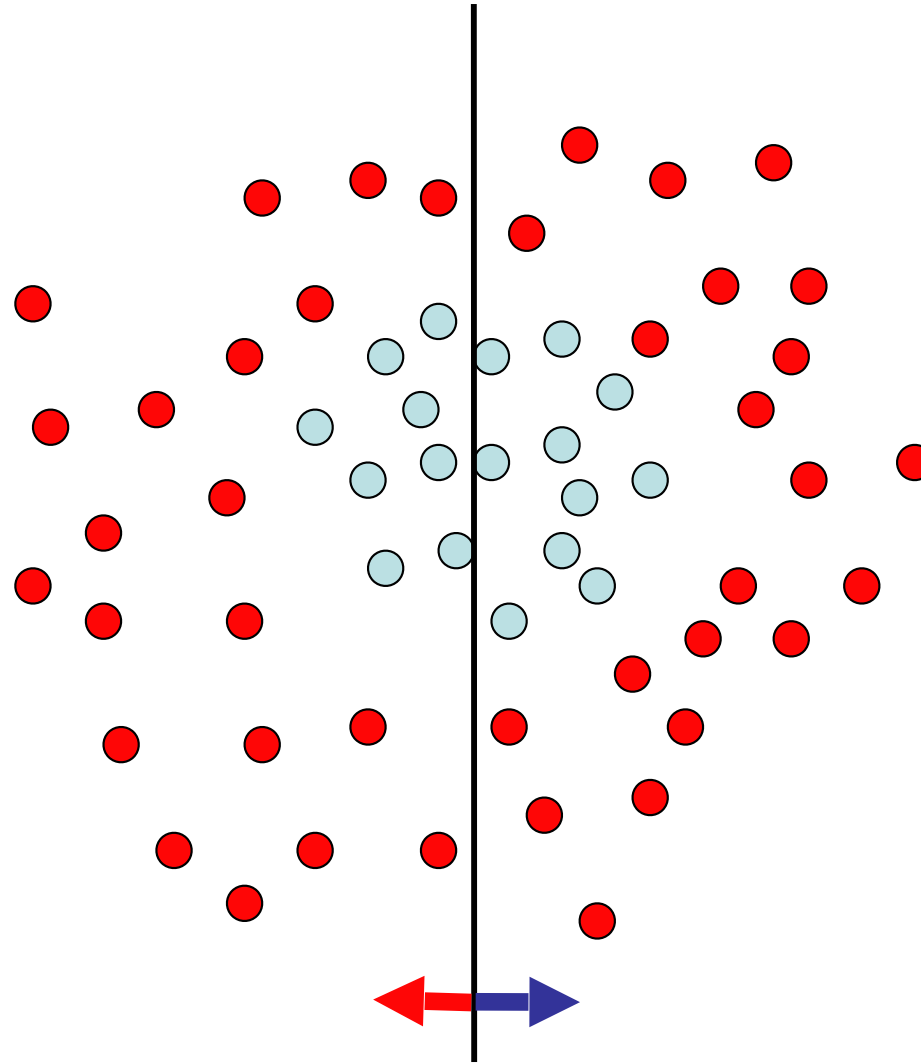
Ada Boost

Iteratively building a weighted sum of weak classifiers:

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

Strong classifier  **Weak classifier**  **Weight** 

Toy Example



Each data point has
a class label:

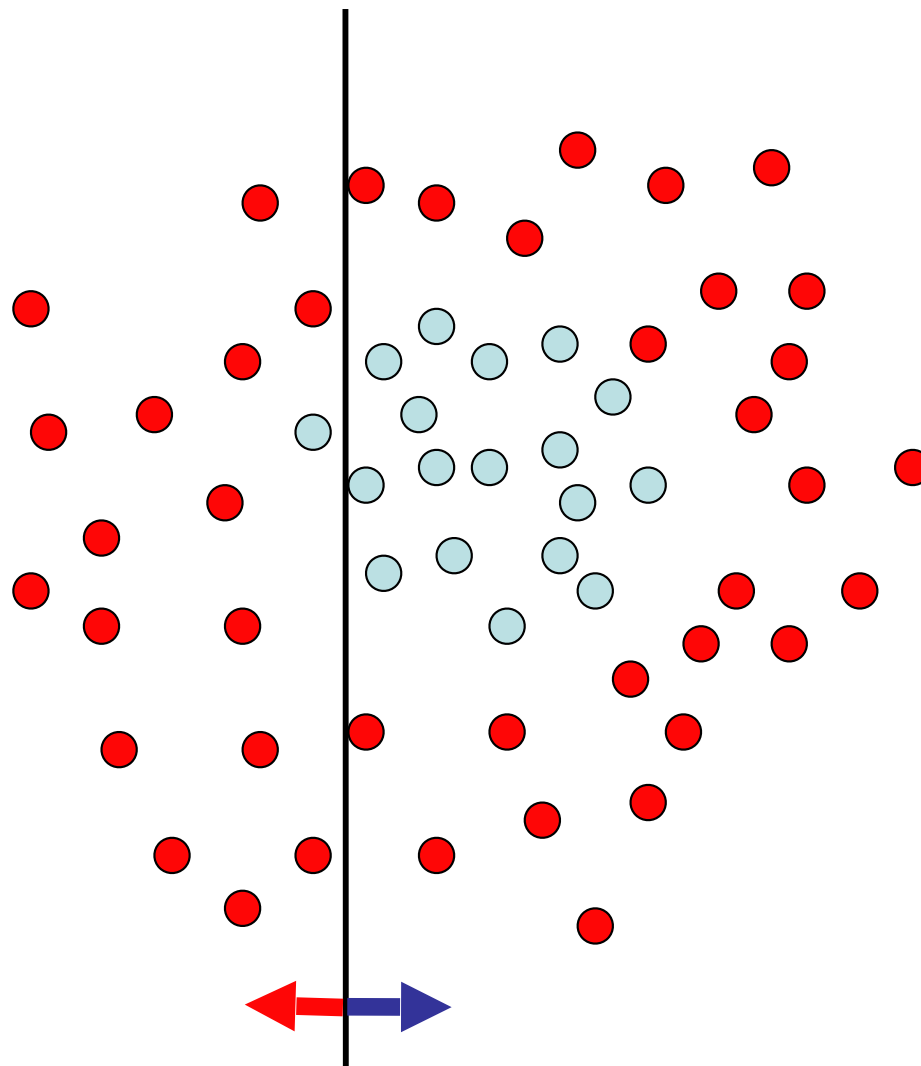
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

and a weight:

$$w_t = 1$$

Classifier is roughly at chance.

Toy Example



Each data point has
a class label:

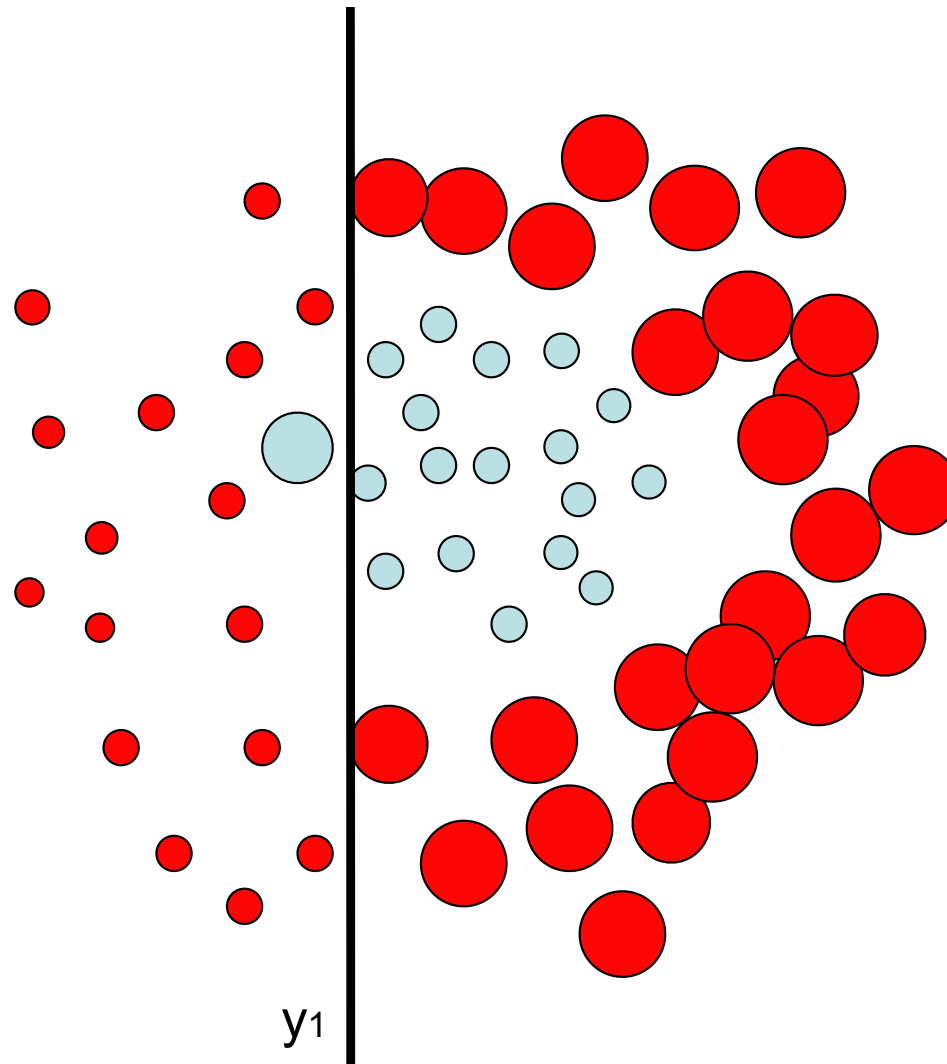
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{light blue circle}) \end{cases}$$

and a weight:

$$w_t = 1$$

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

Toy Example



Each data point is given
a new class label

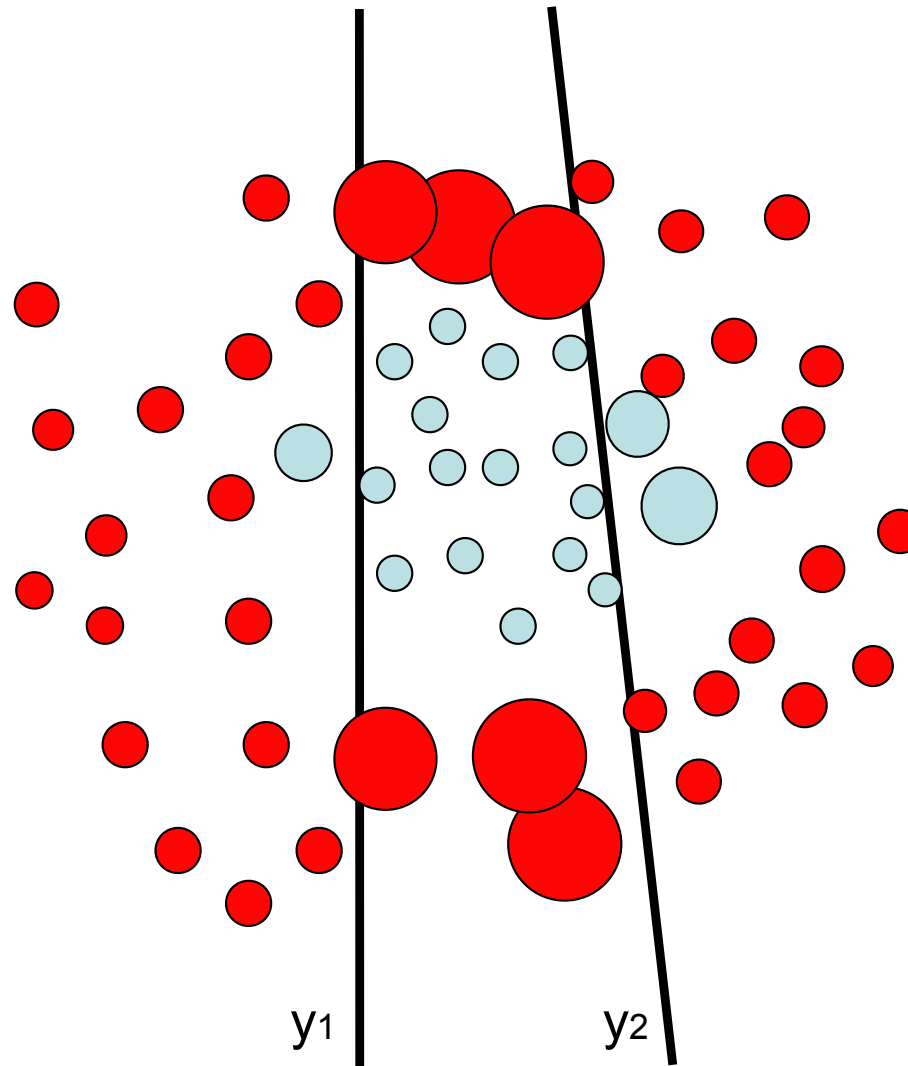
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

and a new weight

$$w_t$$

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

Toy Example



Each data point is given
a new class label

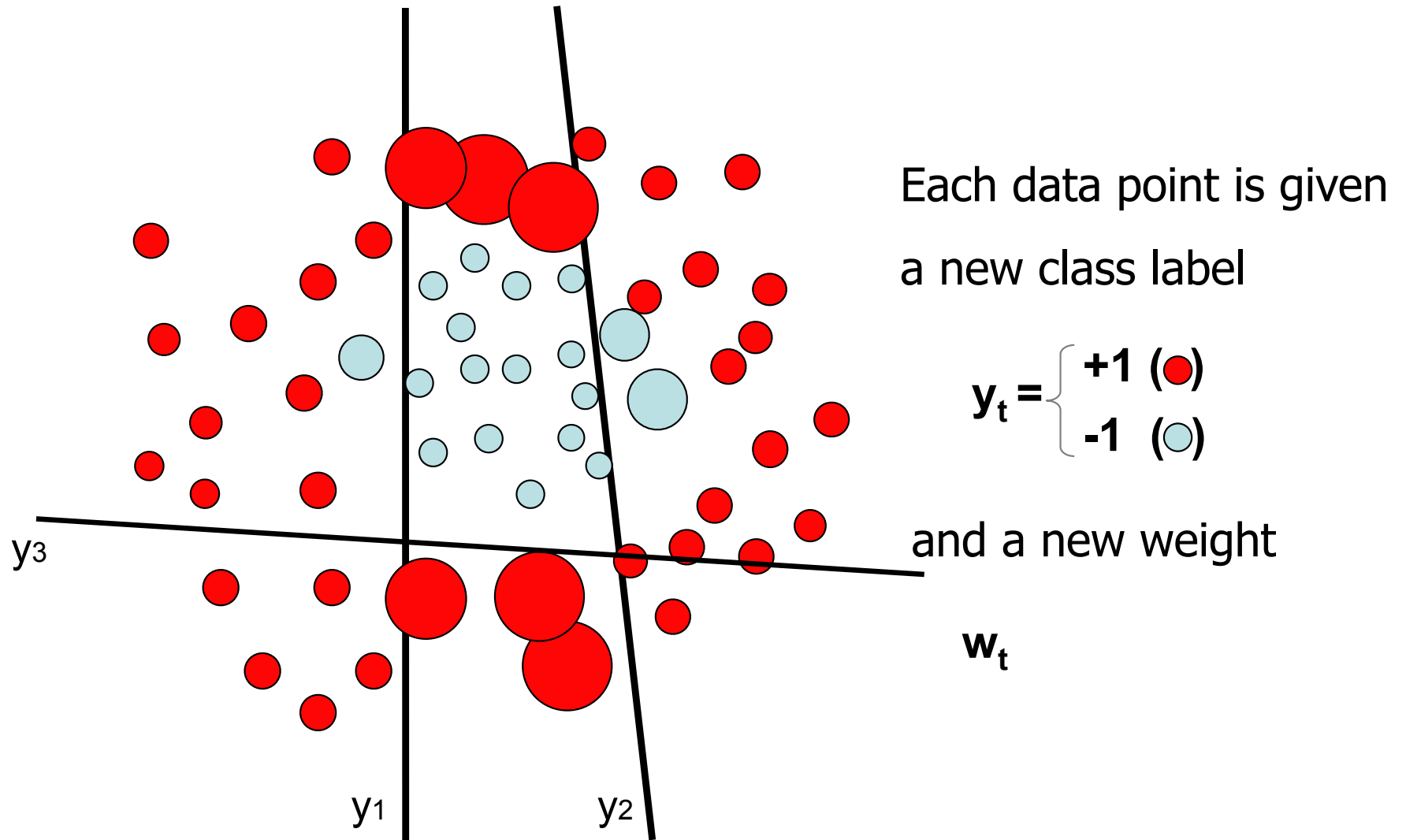
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

and a new weight

$$w_t$$

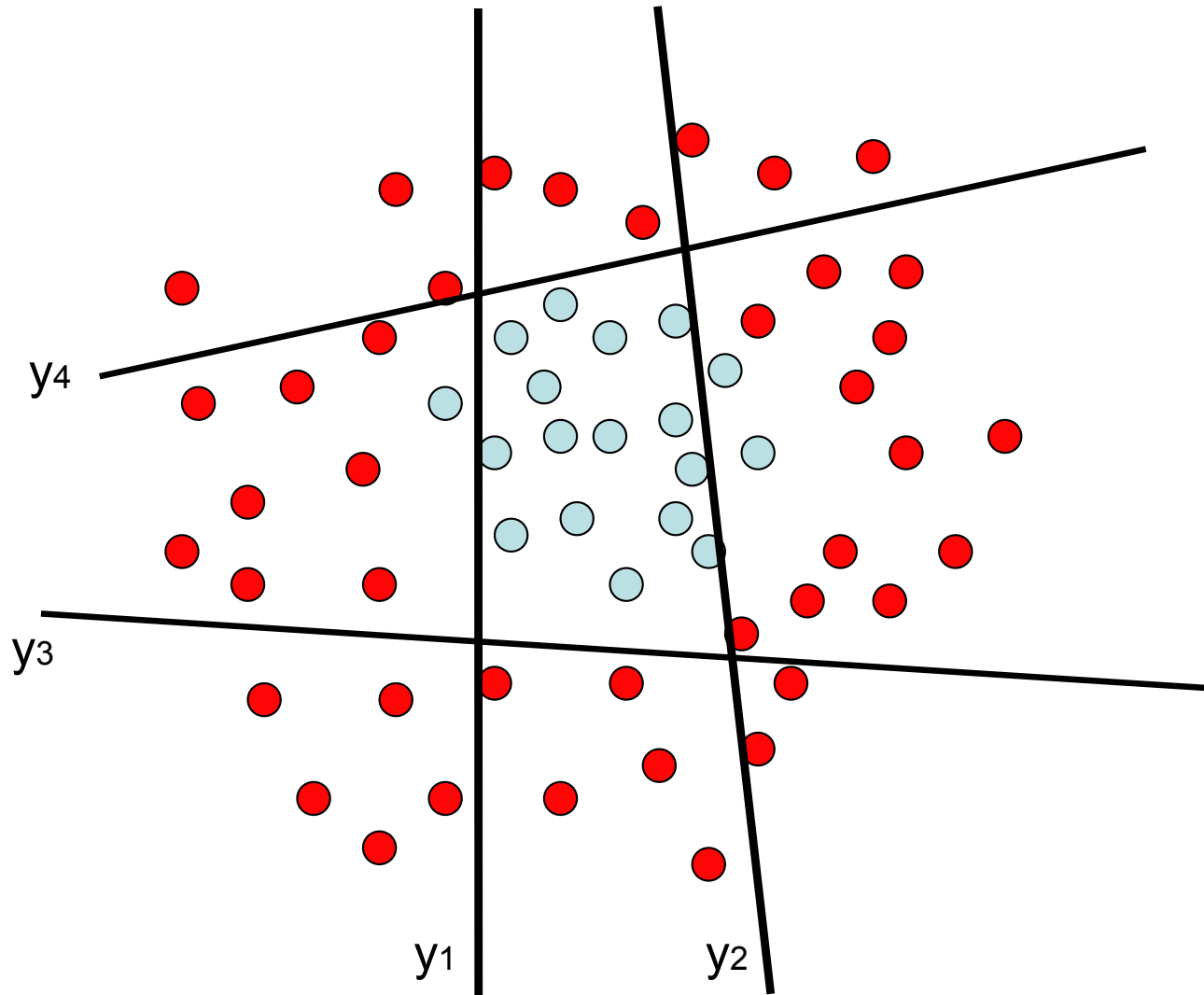
Find a new classifier y_2 and reset the weights again.

Toy Example



Find a new classifier y_3 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 $t = [1, \dots, T]$:

(a) Find classifier $y_t : \chi \rightarrow \{-1, 1\}$ 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.

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

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.

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

Optional: Proof Sketch (1)

At iteration t :

$$f_t(\mathbf{x}) = \frac{1}{2} \sum_{s=1}^t \alpha_s y_s(\mathbf{x}) ,$$
$$= f_{t-1}(\mathbf{x}) + \frac{1}{2} \alpha_t y_t(\mathbf{x}) .$$

Computed at previous iteration.

To be estimated.

To estimate the unknowns, we seek to minimize

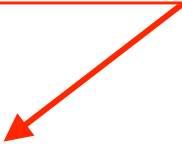
$$E_t = \sum_{n=1}^N \exp(-t_n f_t(\mathbf{x}_n)) ,$$

Exponential loss.

with respect to α_t and y_t .

Optional: Proof Sketch (2)

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

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


with respect to y_t and α_t .

Optional: Proof (2)

$$\begin{aligned} E_t &= \sum_n w_n \exp\left(-\frac{\alpha_t}{2} t_n y_t(\mathbf{x}_n)\right) \\ &= \exp(-\alpha_t/2) \sum_{t_n=y_t(\mathbf{x}_n)} w_n^t + \exp(\alpha_t/2) \sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t \\ &= \underbrace{(\exp(\alpha_t/2) - \exp(-\alpha_t/2))}_{\text{Greater than 0}} \underbrace{\sum_{n=1}^N I(t_n \neq y_t(\mathbf{x}_n)) w_n^t}_{\text{Must be minimized}} + \underbrace{\exp(-\alpha_t/2) \sum_{n=1}^N w_n^t}_{\text{Independent of } y_t} \end{aligned}$$

Therefore, for E_t to be minimized, y_t must minimize $\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$.

Optional: Proof (3)

$$E_t = \sum_n w_n \exp\left(-\frac{\alpha_t}{2} t_n y_t(\mathbf{x}_n)\right)$$

$$\Rightarrow 2 \frac{\delta E_t}{\delta \alpha_t} = -\exp(-\alpha_t/2) \sum_{t_n=y_t(\mathbf{x}_n)} w_n^t + \exp(\alpha_t/2) \sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$$

Therefore:

$$\frac{\delta E_t}{\delta \alpha_t} = 0 \Rightarrow \alpha_t = \log \left[\frac{\sum_{t_n=y_t(\mathbf{x}_n)} w_n^t}{\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t} \right]$$

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

Optional: Proof (4)

At the following iteration:

$$\begin{aligned}w_n^{t+1} &= \exp(-t_n f_t(\mathbf{x}_n)) \\&= \exp(-t_n f_{t-1}(\mathbf{x}_n) - \frac{1}{2} \alpha_t t_n y_t(\mathbf{x}_n)) \\&= w_n \exp(-\frac{1}{2} \alpha_t t_n y_t(\mathbf{x}_n))\end{aligned}$$

$$t_n y_t(\mathbf{x}_n) = 1 - 2I(y_t(\mathbf{x}_n) \neq t_n)$$

$$\begin{aligned}\Rightarrow w_n^{t+1} &= w_n^t \exp(-\alpha_t/2) \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n))) \\&\propto w_n^t \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n)))\end{aligned}$$

Optional: Proof Sketch (5)

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:

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

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

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

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

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 $t = [1, \dots, T]$:

(a) Find classifier $y_t : \chi \rightarrow \{-1, 1\}$ that minimizes weighted error $\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$.

(b) Evaluate

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

(c) Update weights

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

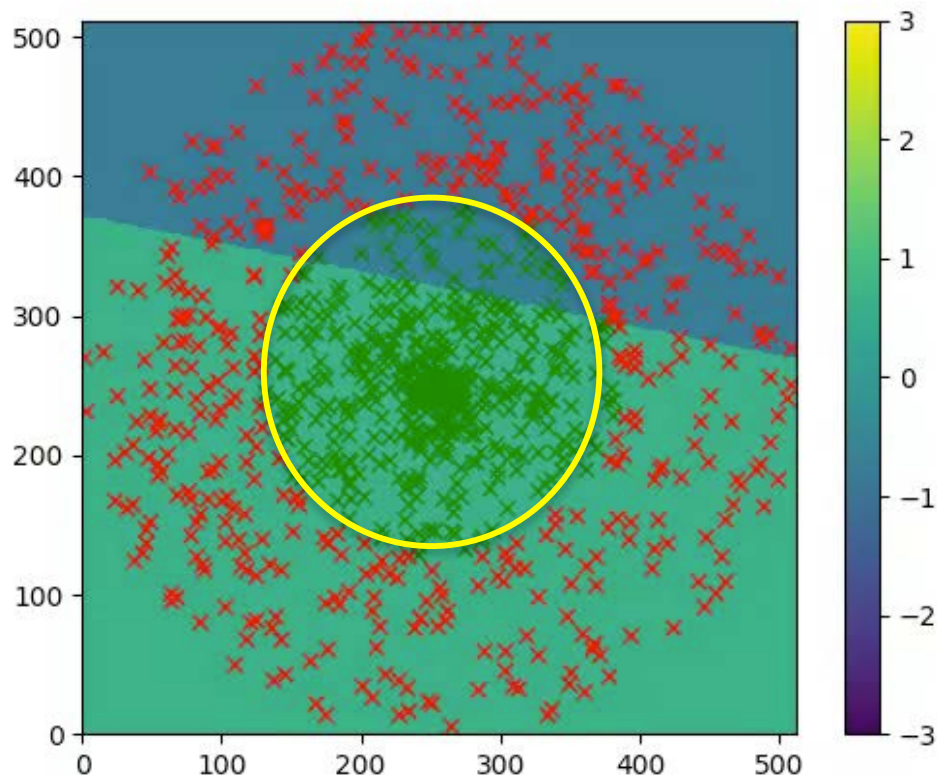
→ **Final classifier:** $Y(\mathbf{x}) = \text{sign}(\sum_{t=1}^T \alpha_t y_t(\mathbf{x}))$

Adaboost in Python

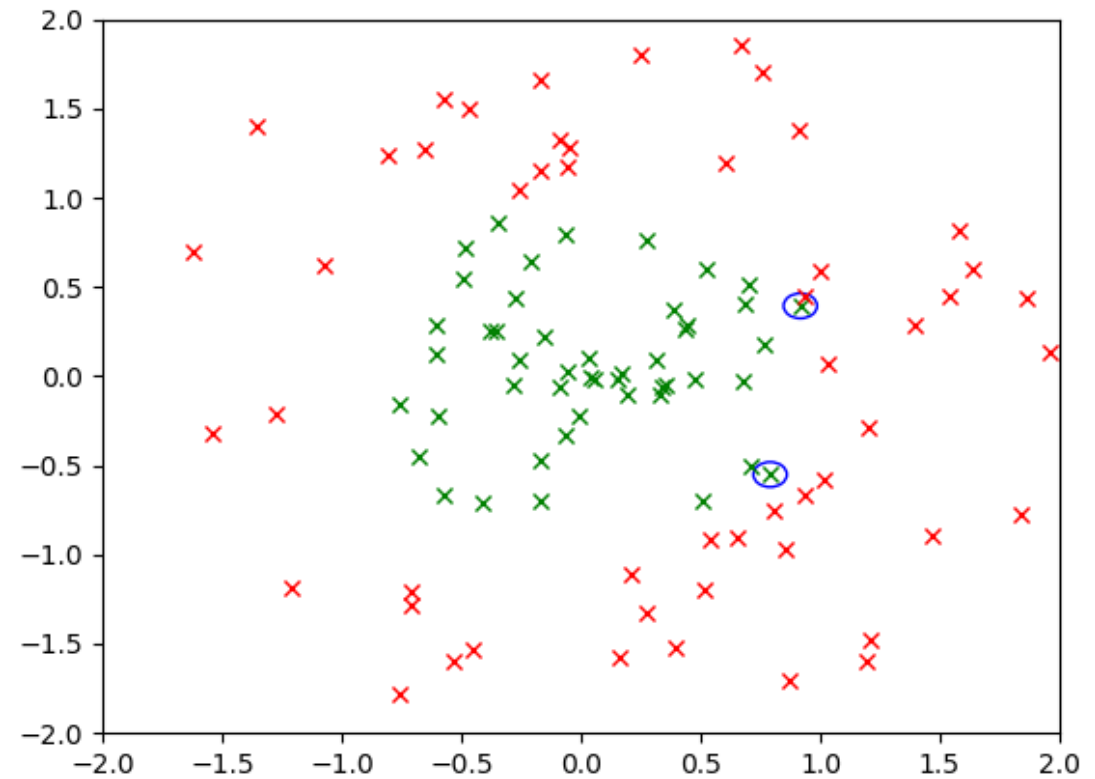
```
def fit(self,nit=10):  
    # Initialize weights and list of classifiers  
    self.weakCls = []  
    bestAcc = 0.0  
    self.datCoeffs = np.ones(self.ns,dtype=np.float)/self.ns  
    # Find nit weak classifiers and update weights each time.  
    for m in range(nit):  
        weakC=self.getWeakC()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

Circular Distribution

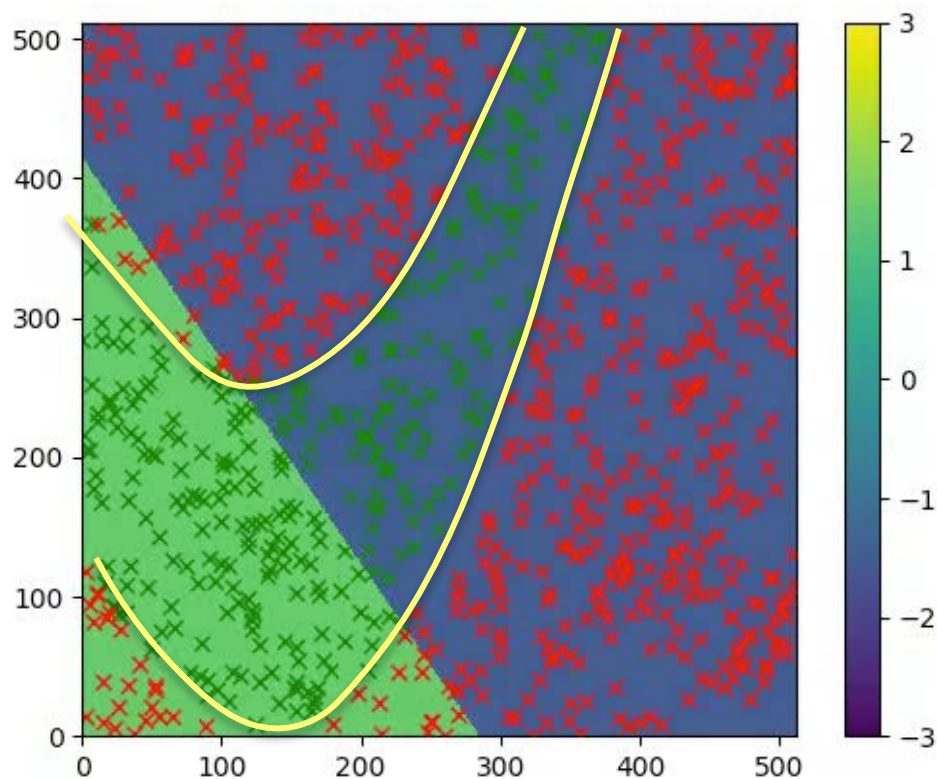


Training (100 iterations)

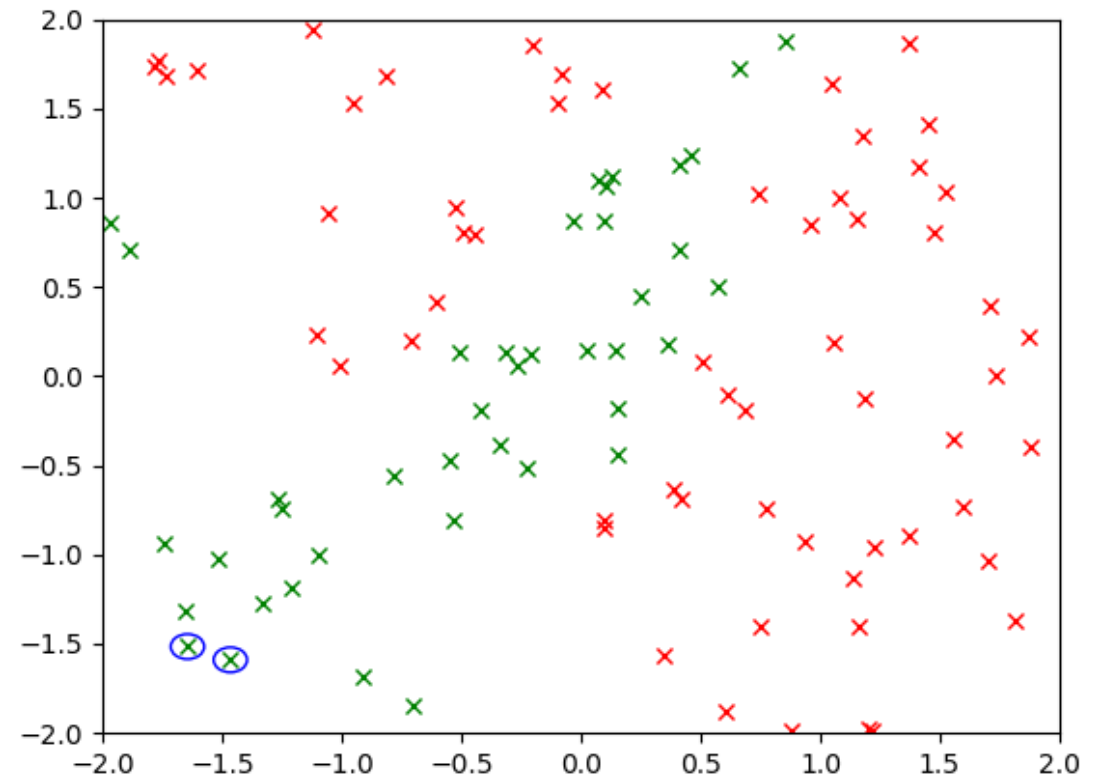


Validation (98% accuracy)

Rosenbrock Function



Training (100 iterations)

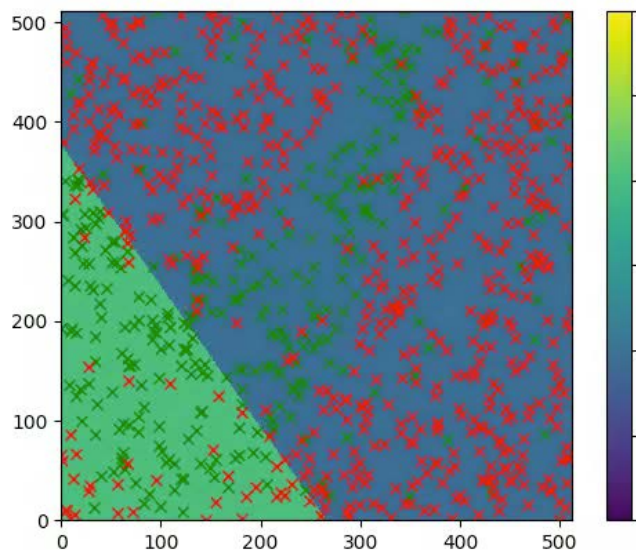


Validation (98% accuracy)

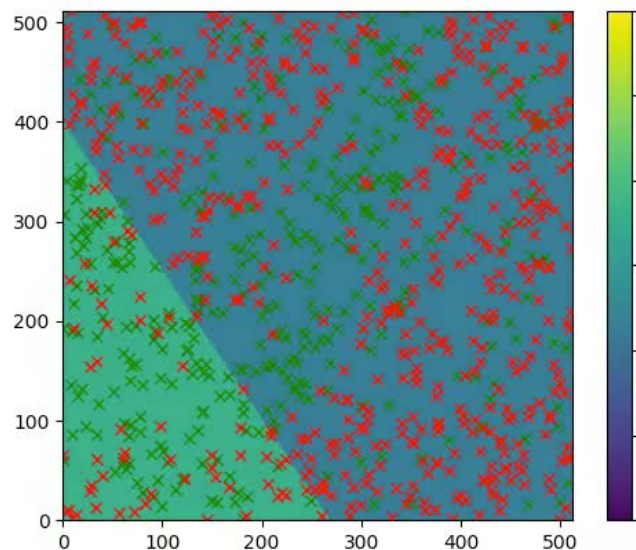
$$r(x, y) = 100 * (y - x^2)^2 + (1 - x)^2$$

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

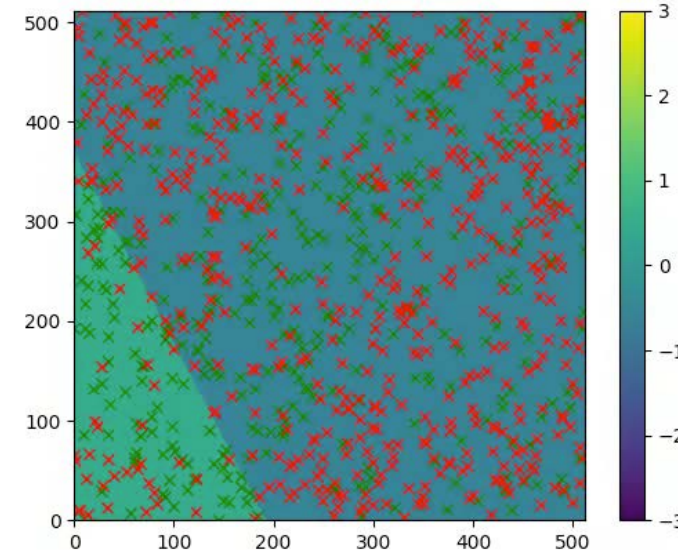
Noisy Labels



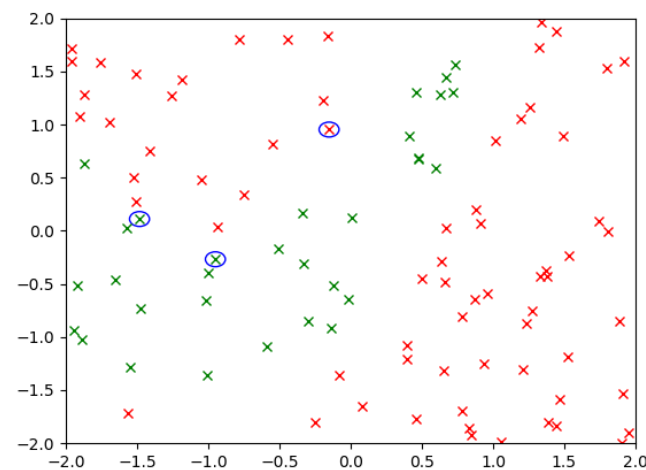
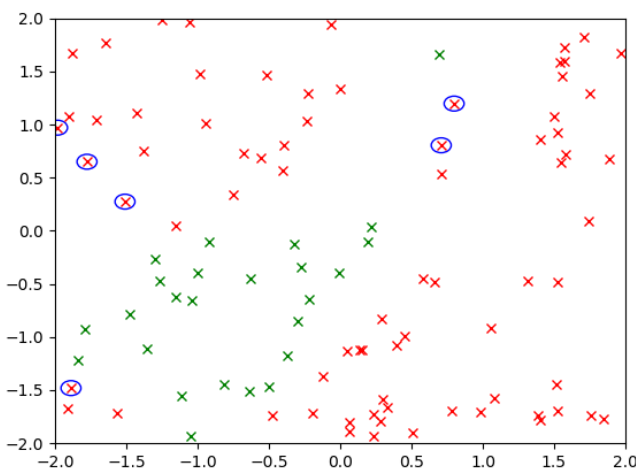
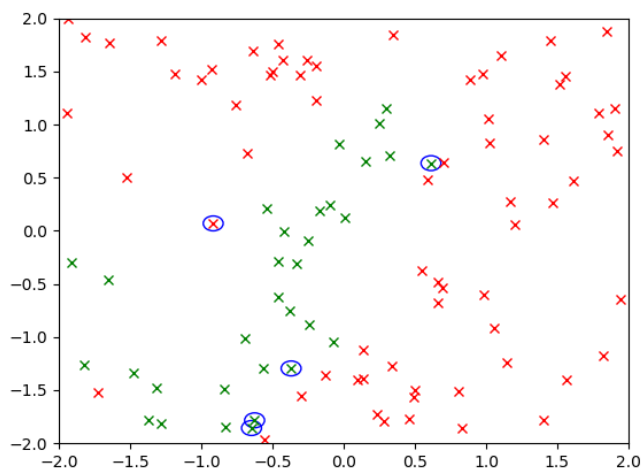
10% mislabeled



20% mislabeled



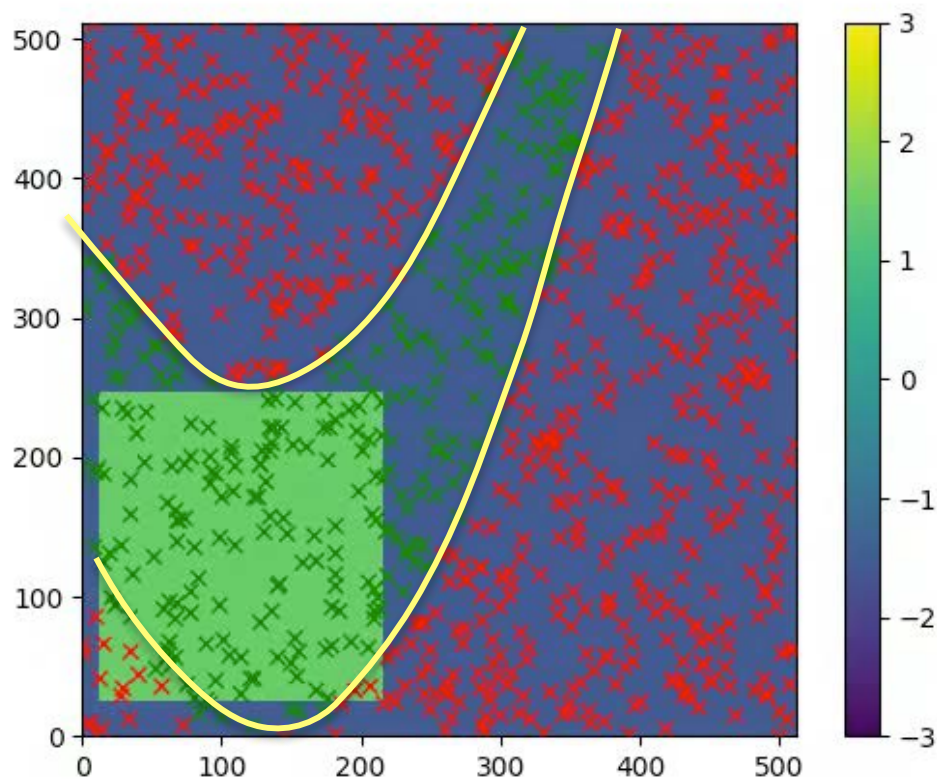
30% mislabeled



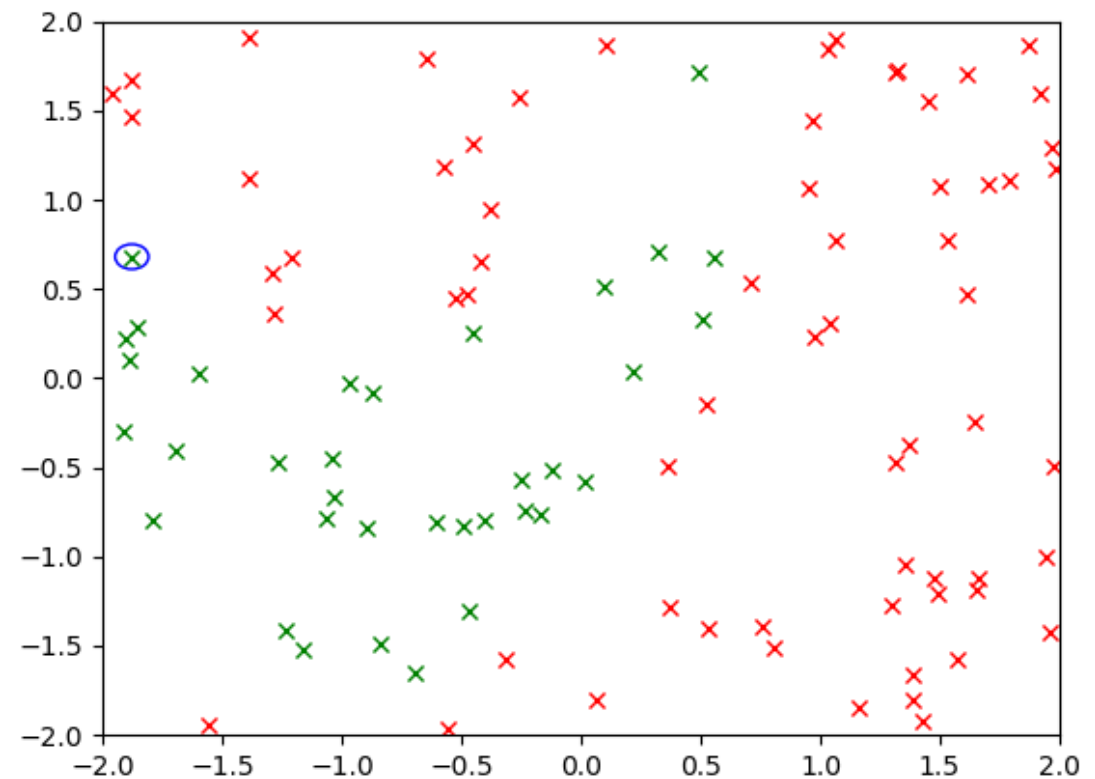
The incorrect labels have relatively little impact because they are randomly distributed, but they could.

Changing the Weak Learners

Using boxes instead of lines.



Training (100 iterations)



Validation (99% accuracy)

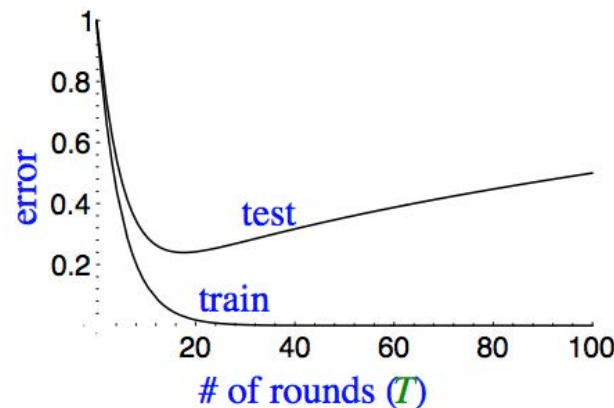
$$y(\mathbf{x}; \mathbf{w}) = \begin{cases} 1 & \text{if } x_0 < \mathbf{x}[1] < x_1 \text{ and } y_0 < \mathbf{x}[2] < y_1, \\ -1 & \text{otherwise.} \end{cases}$$
$$\mathbf{w} = (x_0, y_0, x_1, y_1)$$

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.

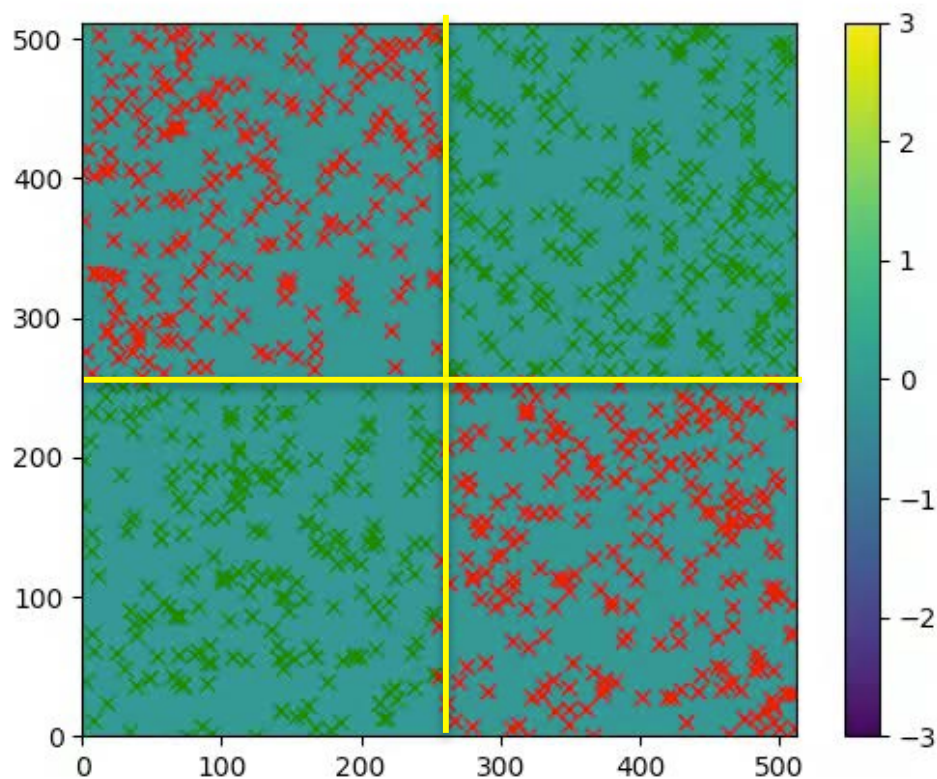
$$\frac{1}{N} \sum_n [t_n \neq h(\mathbf{x}_n)] < \prod_{t=1}^T \sqrt{\epsilon_t(1 - \epsilon_t)}$$

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

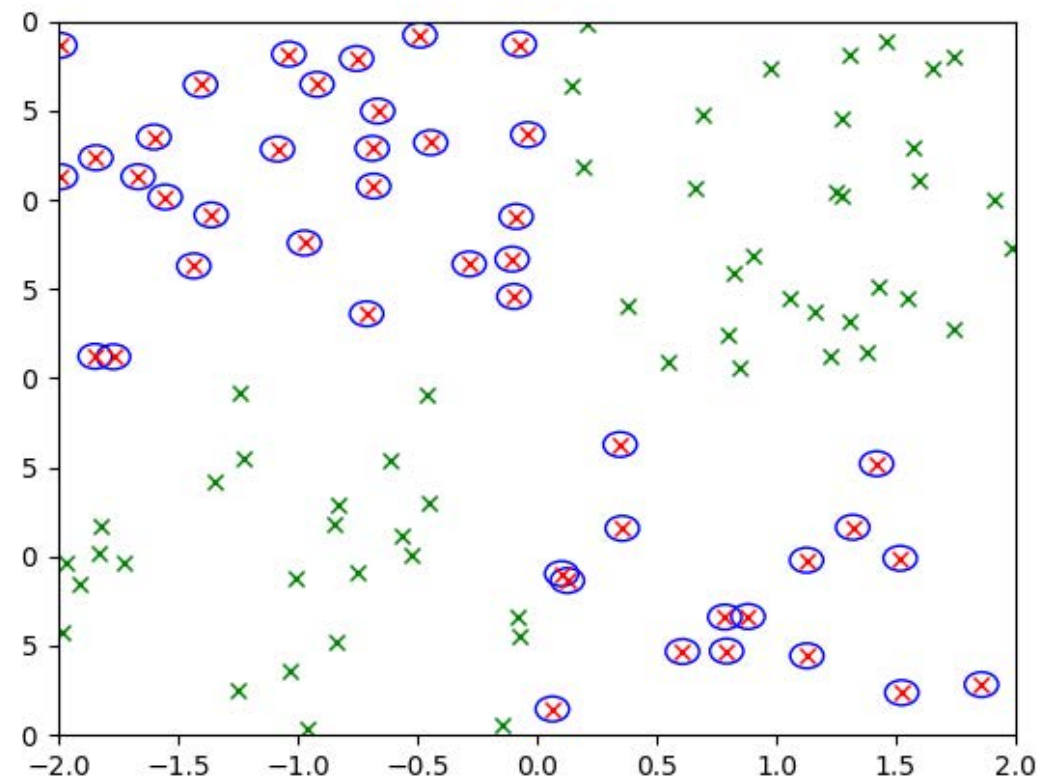


—> Use a validation set.

Failure Mode



Training (100 iterations)



Validation (56% accuracy)

- Individual linear classifiers cannot do better than chance!
- Box classifiers would work though.

Adaboost in Python

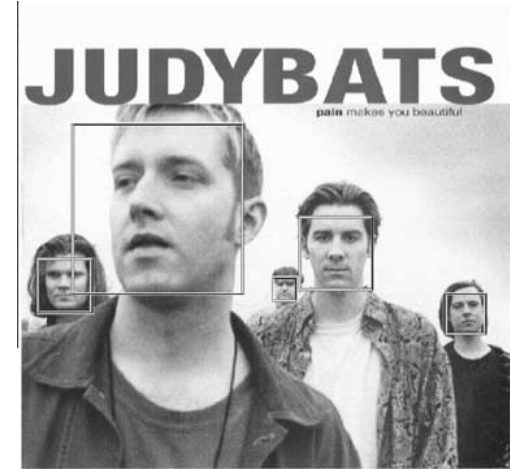
```
def fit(self,nit=10):  
    # Initialize weights and list of classifiers  
    self.weakCls = []  
    bestAcc = 0.0  
    self.datCoeffs = np.ones(self.ns,dtype=np.float)/self.ns  
    # Find nit weak classifiers and update weights each  
    time.  
    for m in range(nit):  
        weakC=self.getWeakC()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

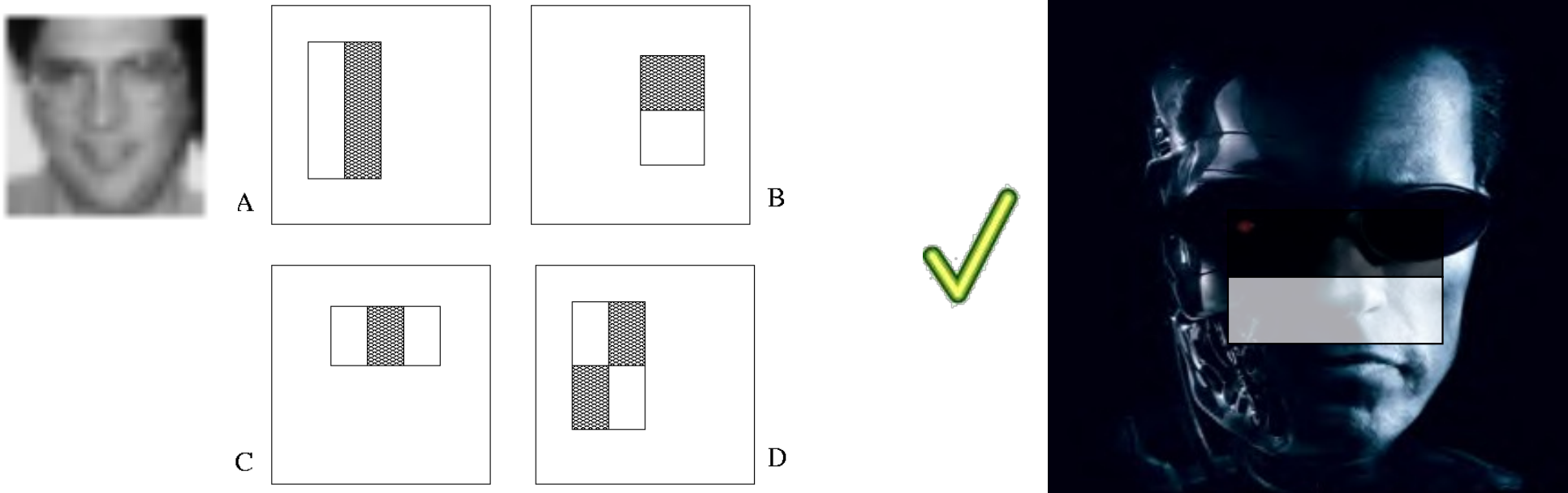


Viola & Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR 2001:

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



Weak Learners for Images



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

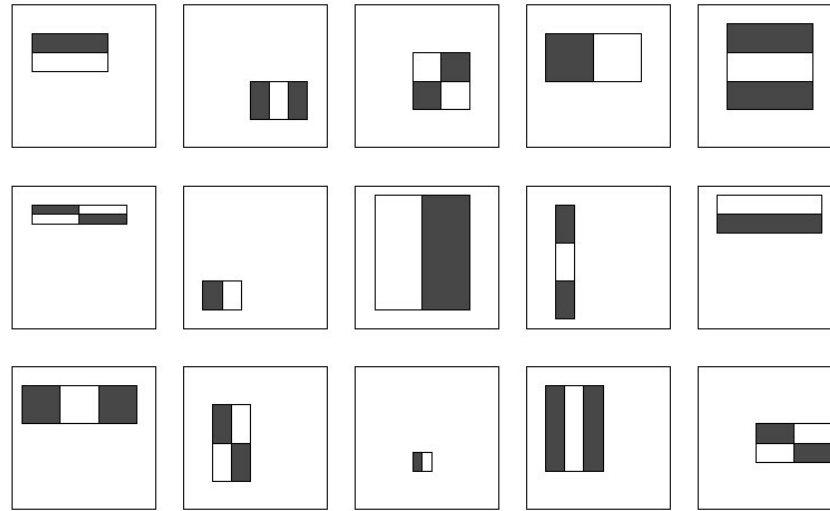
Haar Wavelets:

- Allow target function over an interval to be represented in terms of an **orthonormal basis**.
- Fast to compute (4 operations per rectangle).
- 180'000 possibilities for a 24x24 window.

—> Use AdaBoost to choose a good subset.

Feature Selection

Among:



1st WL

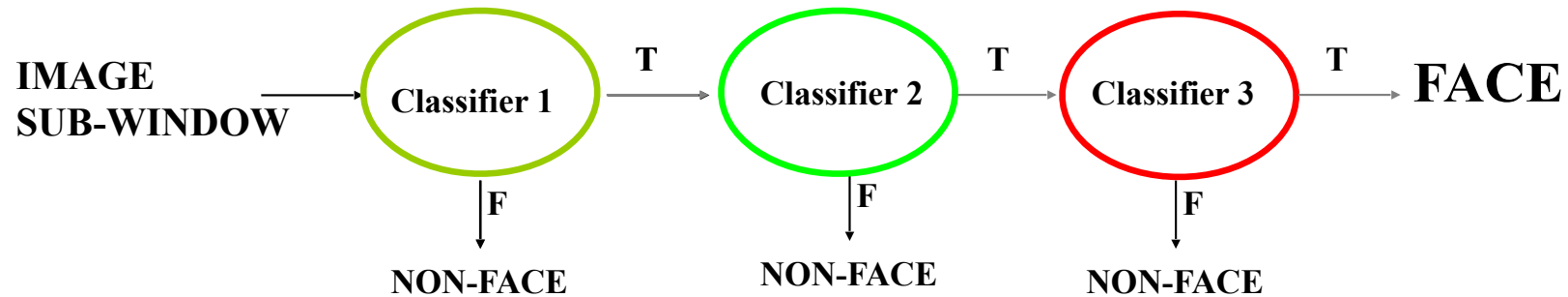
2nd WL



Pick:



Cascade



Reject large portions of the images using only the response of the first few weak classifiers.

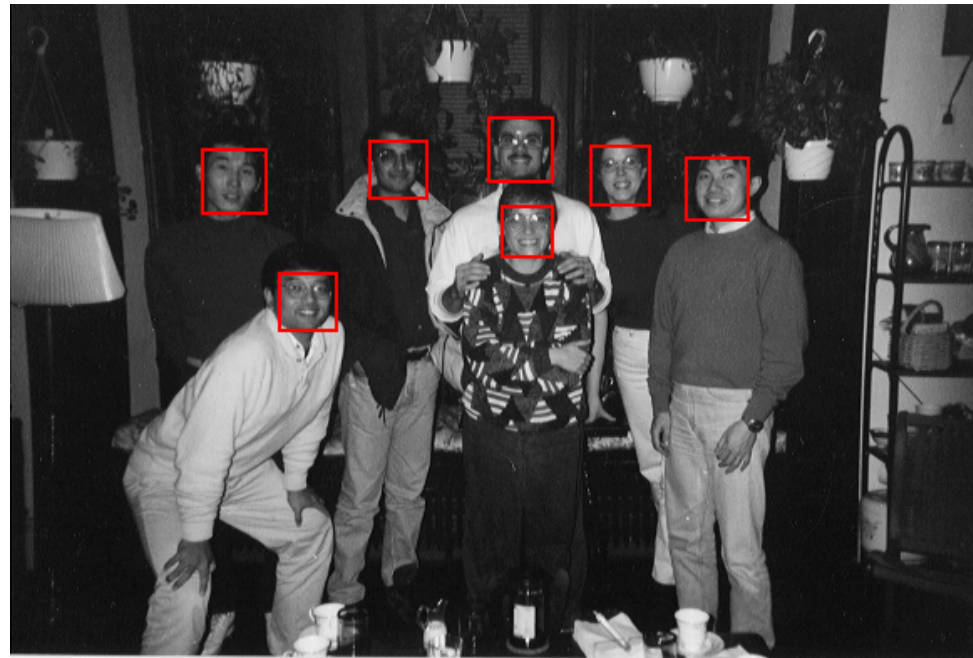
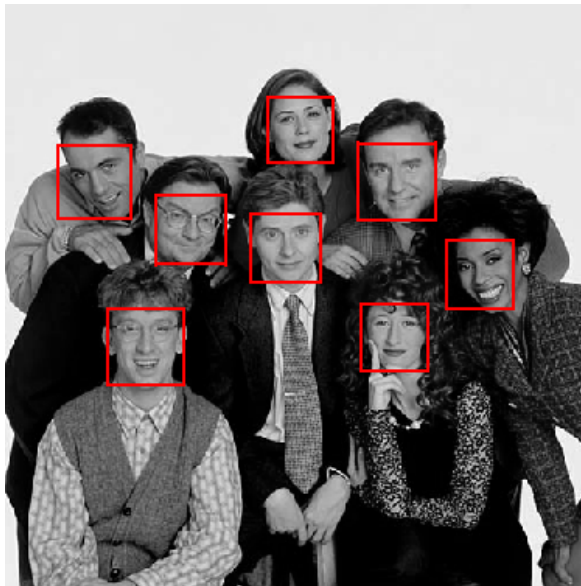
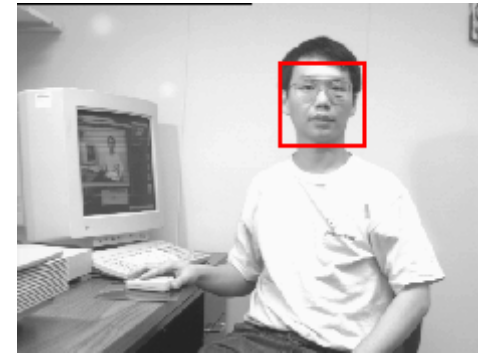
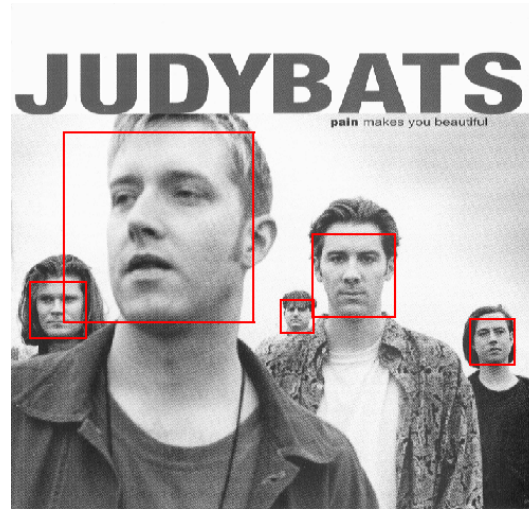
—> Large potential speed-up at run-time.

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



Detection Results (2001)

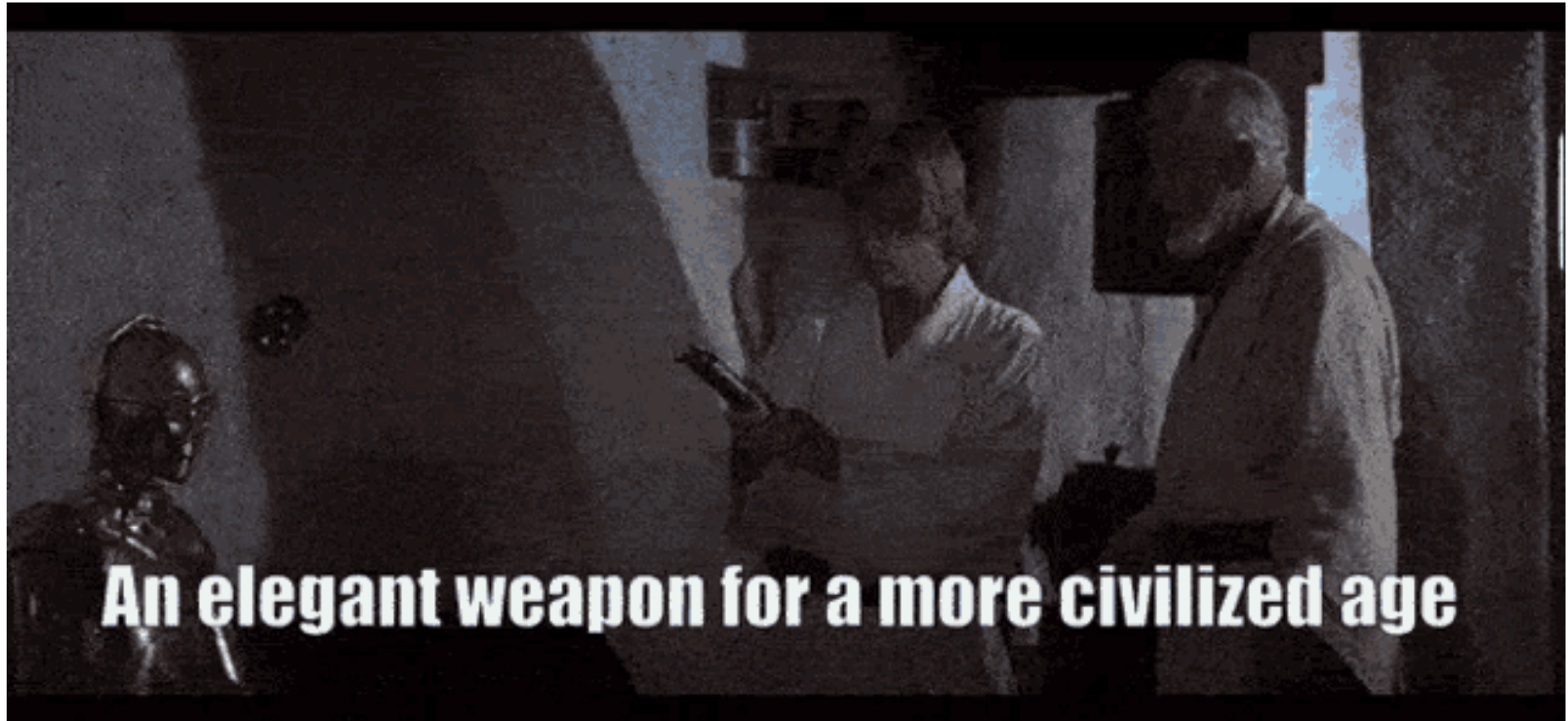


Detection Results (2017)



Head turned in the back: Still detected!

AdaBoost Today



- Useful one when computational resources are limited.
- A good hint about what more complex techniques do.