Neural Networks and Learning Systems
TBMI 26

Lecture 4
Ensemble learning & boosting

Ola Friman
ola.friman@liu.se

History
(roughly)

- 1960’s (and before): Linear methods, perceptron, LDA
- 1980’s: Nonlinear breakthroughs, neural networks
- 1990’s-now:
  – Kernel methods, SVM
  – Ensemble learning, boosting, bagging

Applications

Face detection

Pedestrian detection

J. Shotton et al., *"Real-Time Human Pose Recognition in Parts from Single Depth Images"*

Pose estimation

Organ detection

Barbu et al. *"Marginal Space Learning for Fast Object Detection in Medical Imaging"*

Combining simple rules

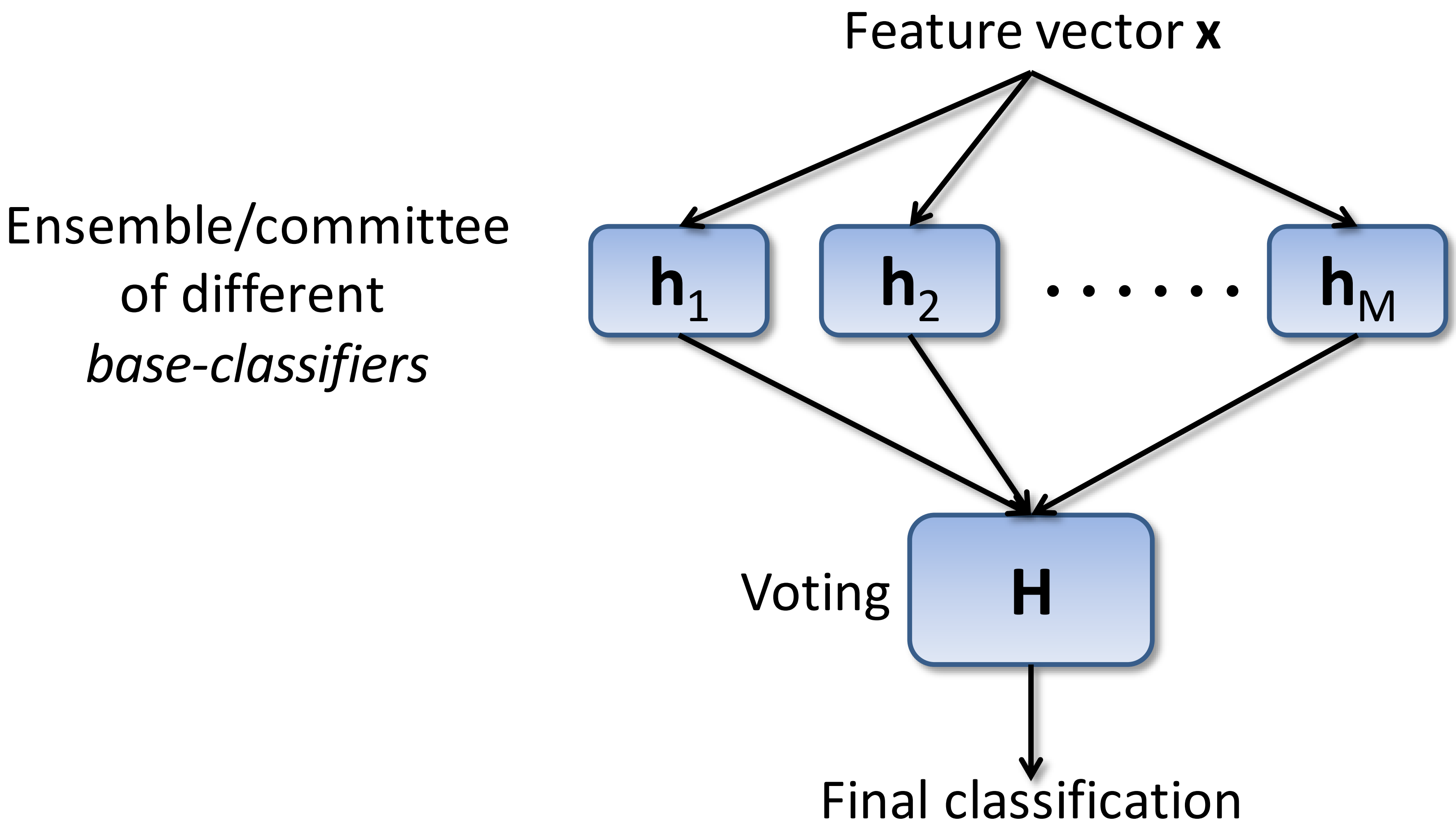
Example taken from *"A Short Introduction to Boosting"*
by Y. Freund and R. Schapire

- "A horse-racing gambler, hoping to maximize his winnings, decides to create a computer program that will accurately predict the winner of a horse race based on the usual information (number of races recently won by each horse, betting odds for each horse, etc.)."
- "To create such a program, he asks a highly successful expert gambler to explain his betting strategy. Not surprisingly, the expert is unable to articulate a grand set of rules for selecting a horse. On the other hand, when presented with the data for a specific set of races, the expert has no trouble coming up with a "rule of thumb" for that set of races (such as, "Bet on the horse that has recently won the most races" or "Bet on the horse with the most favored odds"). Although such a rule of thumb, by itself, is obviously very rough and inaccurate, it is not unreasonable to expect it to provide predictions that are at least a little bit better than random guessing."
- "Furthermore, by repeatedly asking the expert's opinion on different collections of races, the gambler is able to extract many rules of thumb."

"**how can they be combined into a single, highly accurate prediction rule**?"

- "Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above."

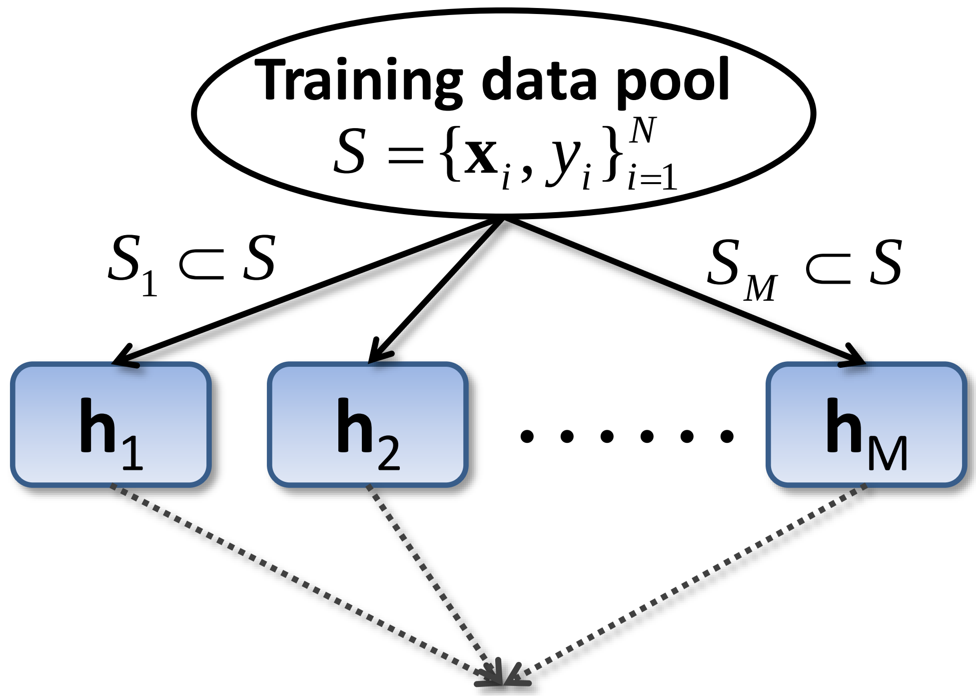
Classifier ensemble



Bootstrap Aggregating (Bagging)

Breiman,1994

Train each base-classifier using a subset of the training data



Bagging

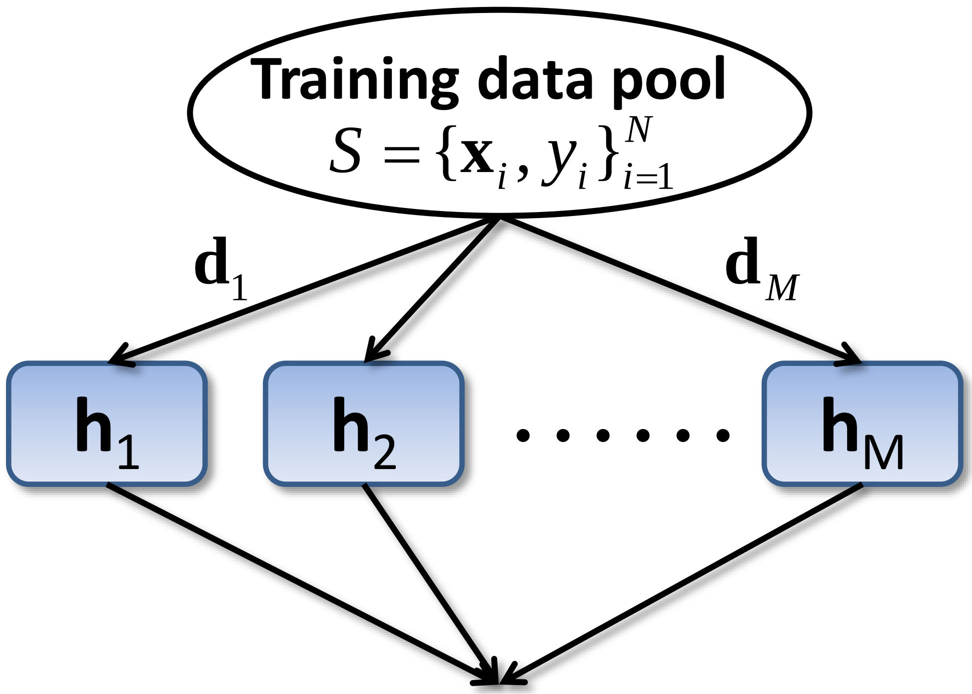
- Reduces overfitting
- Is best used with base-classifiers that can give very different outputs if the input is changed slightly, for example neural networks (different local minima found).
- Does not work with linear classifiers, e.g., LDA.

Boosting

Schapire and others, (1989-1990)

Train each base-classifier using all training data but with weights indicating how important each training sample is.

$$\mathbf{d} = \{d_i\}_{i=1}^{N}$$



$$H(\mathbf{x}) = \operatorname{sign}\left(\sum_{k=1}^{M} \alpha_k h_k(\mathbf{x})\right)$$

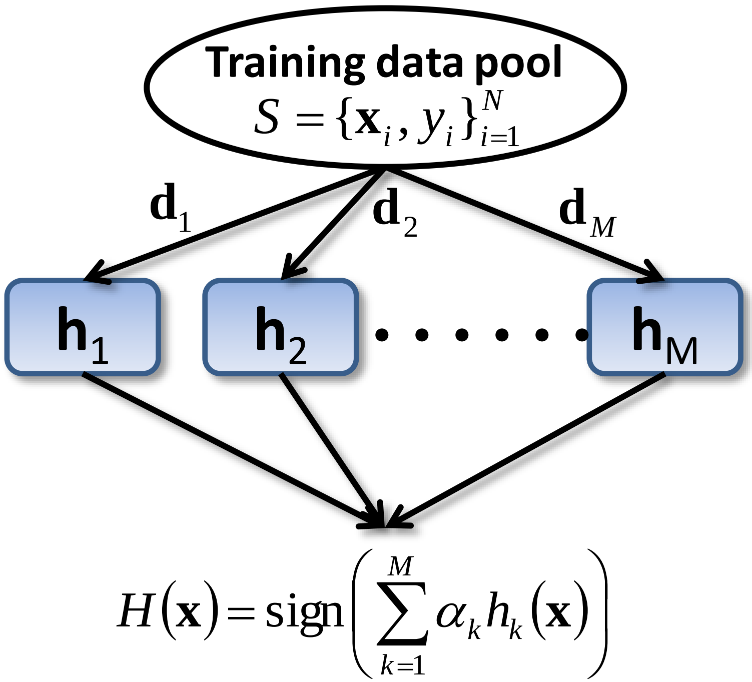
Boosting

- We have seen before that all training samples are not equally important, e.g., SVM.
- Both SVM and Bagging can also be considered to use weights for each sample d_i ={0,1}.
- While the base-classifiers in principle can be any classifier (SVM, neural network, etc.), the driving question has been:

Can we combine a number of simple classifiers to create a single strong classifier?

General boosting algorithm

Train weak classifiers sequentially!



1. Set weights d_1=1/N
2. Train weak classifier h_1(**x**) using weights $\mathbf{d}_1$
3. Increase and decrease weight for wrongly and correctly classified training examples respectively -> $\mathbf{d}_2$
4. Train weak classifier h_2(**x**) using weights $\mathbf{d}_2$
5. Repeat until h_M(**x**)

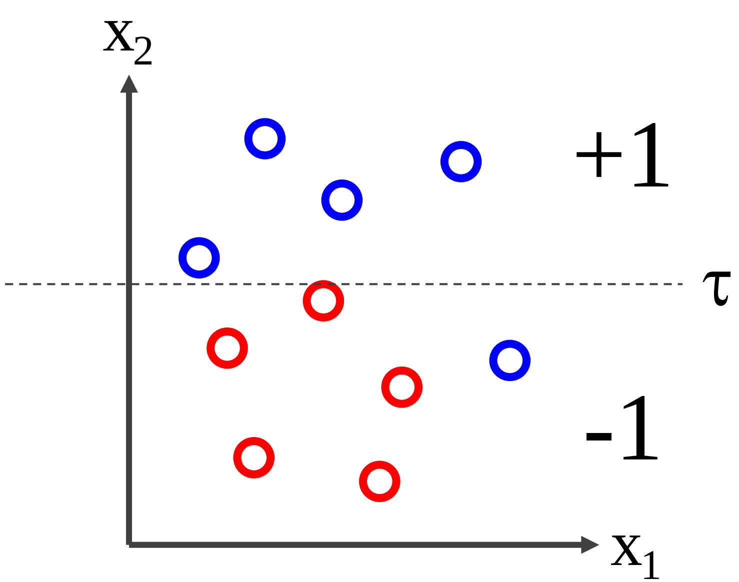
Simple/Weak classifiers

cf. ”a rule of thumb”

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}$$

Example: Threshold ***one*** feature

$$h(x_2) = \begin{cases} +1 & x_2 \geq \tau \\ -1 & x_2 < \tau \end{cases}$$



Weak classifiers – Threshold polarity

$$h(x) = \begin{cases} +1 & p\,x \geq p\,\tau \\ -1 & p\,x < p\,\tau \end{cases}$$

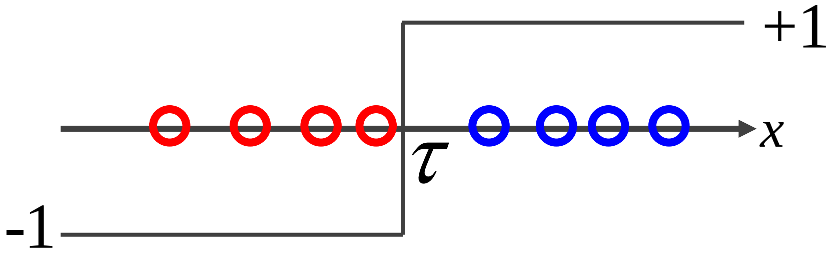
Polarity p ={-1,1}

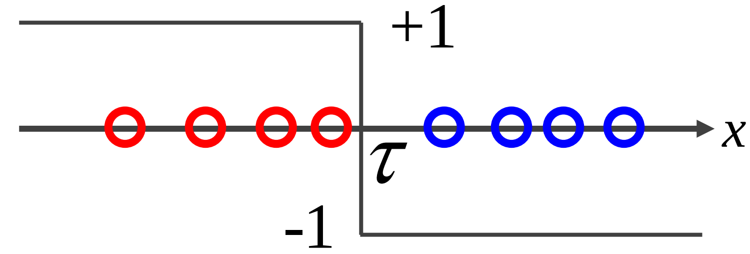
$p = 1$

$$h(x) = \begin{cases} +1 & x \geq \tau \\ -1 & x < \tau \end{cases}$$

p = -1

$$h(x) = \begin{cases} +1 & x \leq \tau \\ -1 & x > \tau \end{cases}$$





Decision stump

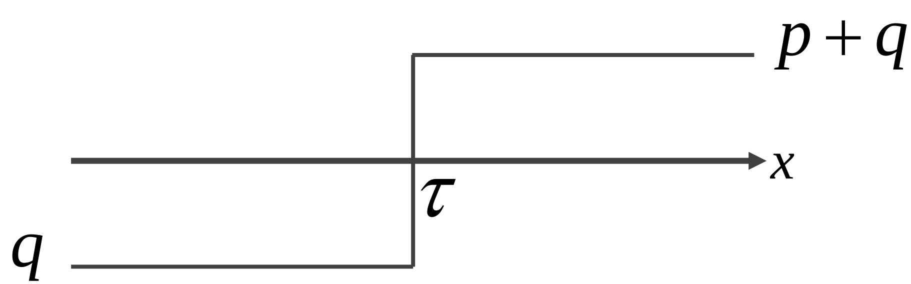
More generally, 4 parameters

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}$$

$$h(x; p, q, \tau, k) = p \cdot \left(x_k \geq \tau\right) + q$$

Feature nbr

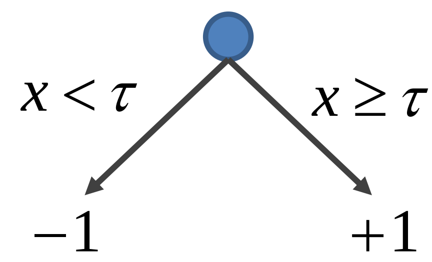
Threshold



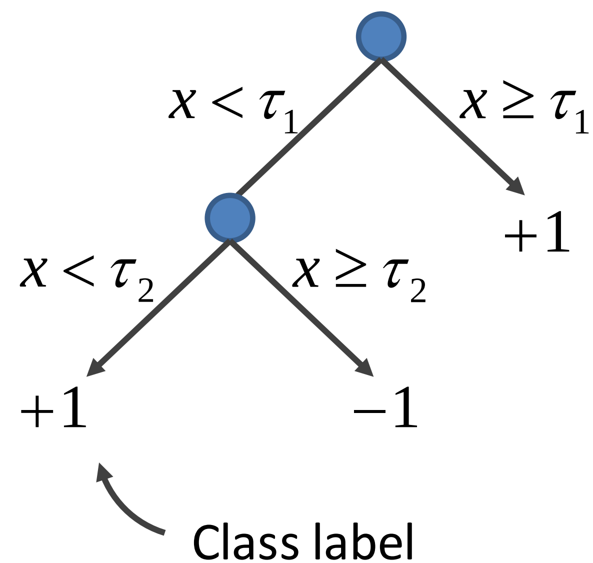
Classification and Regression Trees (CART)

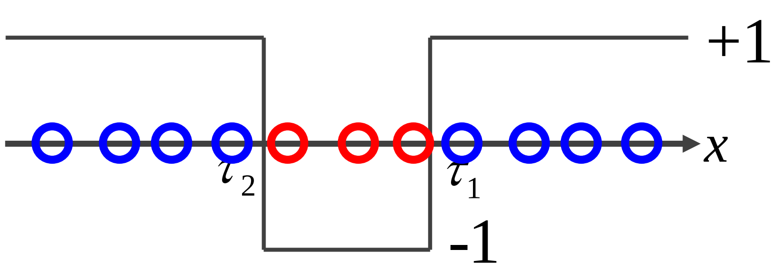
$$h(x) = \begin{cases} +1 & x \geq \tau \\ -1 & x < \tau \end{cases}$$

Decision stump

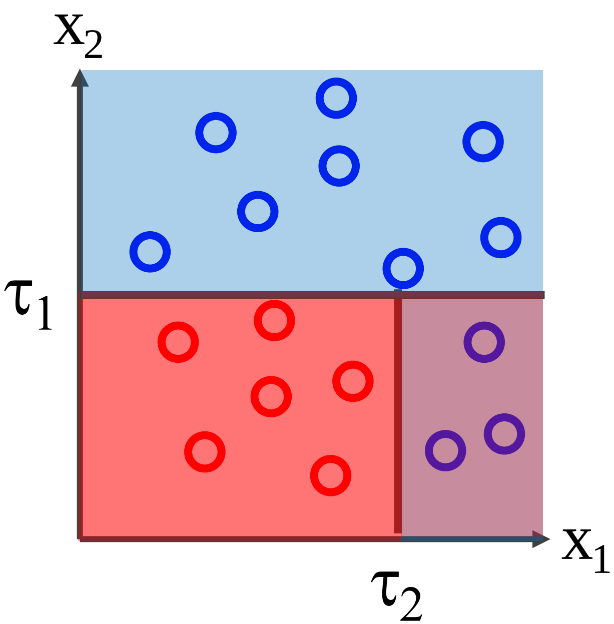


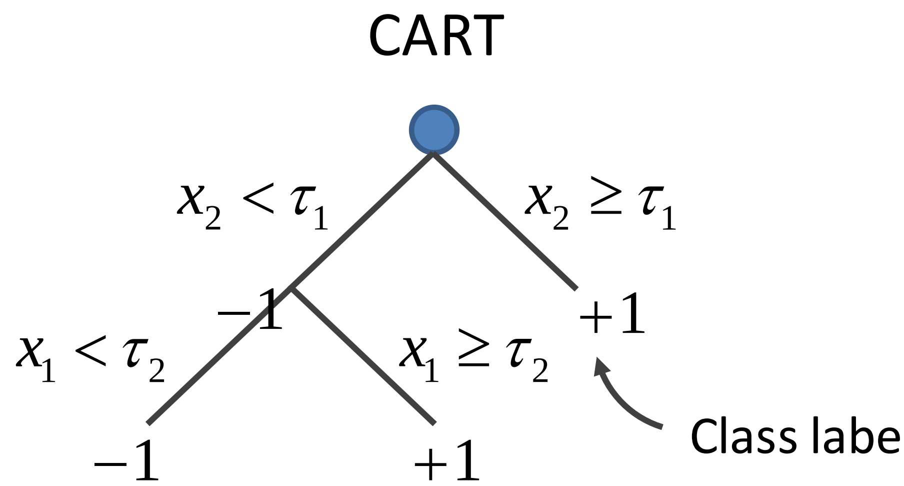
$$h(x) = \begin{cases} +1 & x \geq \tau_1 \\ -1 & x < \tau_1 \text{ and } x \geq \tau_2 \\ +1 & x < \tau_2 \end{cases}$$





CART – 2D example





Piecewise flat classification function

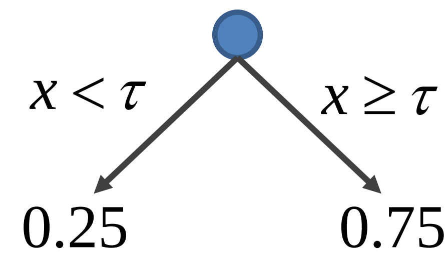
$$f(\mathbf{x};\underbrace{w_1,\ldots,w_k}_{\text{Feature index and thresholds}})\rightarrow \Omega$$

Regression Tree

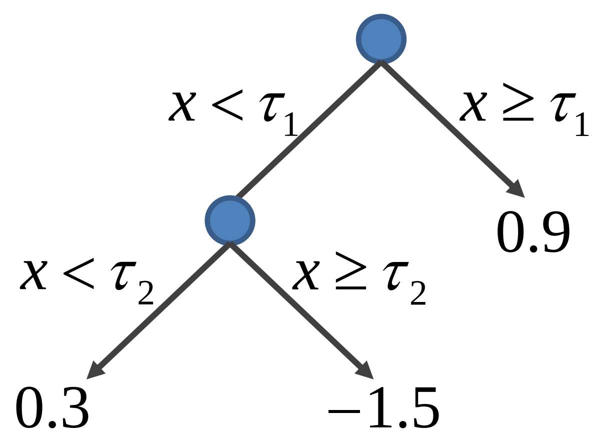
Same, but with real-valued output!

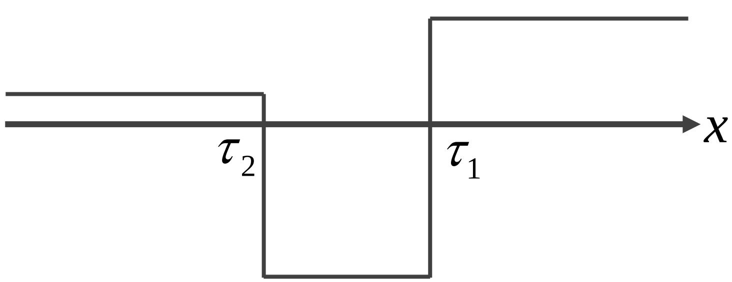
$$h(x) = \begin{cases} 0.75 & x \geq \tau \\ 0.25 & x < \tau \end{cases}$$

Decision stump



$$h(x) = \begin{cases} 0.9 & x \geq \tau_1 \\ -1.5 & x < \tau_1 \text{ and } x \geq \tau_2 \\ 0.3 & x < \tau_2 \end{cases}$$





Training a decision stump

Find best split threshold τ!

Class label {-1,+1}

Training input: $\{x_i, y_i, d_i\}_{i=1}^{M}$

Consider only one feature

Weight

Normalized weights: $\sum_{i=1}^{M} d_i = 1$

Threshold function: $h(x;\tau,p) = \begin{cases} +1 & p\,x \geq p\,\tau \\ -1 & p\,x < p\,\tau \end{cases}$

Polarity {-1,1}

Cost function: $\min_{\tau,p} \varepsilon(\tau,p) = \sum_{i=1}^{M} d_i\, I\big(y_i \neq h(x_i;\tau,p)\big)$

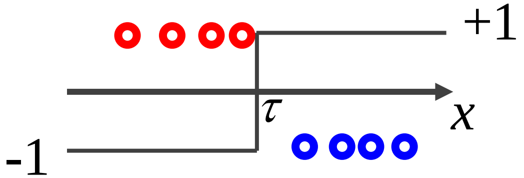
1 for false classifications

Training a decision stump, cont.

$$\min_{\tau,p} \varepsilon(\tau,p) = \sum_{i=1}^{M} d_i\, I\big(y_i \neq h(x_i;\tau,p)\big) \text{ is always} \leq 0.5!$$

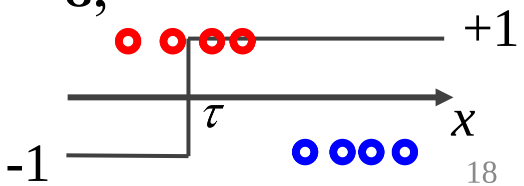
Why? If we classify <u>all</u> training samples wrong we get:

$$\varepsilon(\tau,p) = \sum_{i=1}^{M} d_i = 1$$



But we can then just change polarity/sign and get all samples correct, i.e., ε = 0!

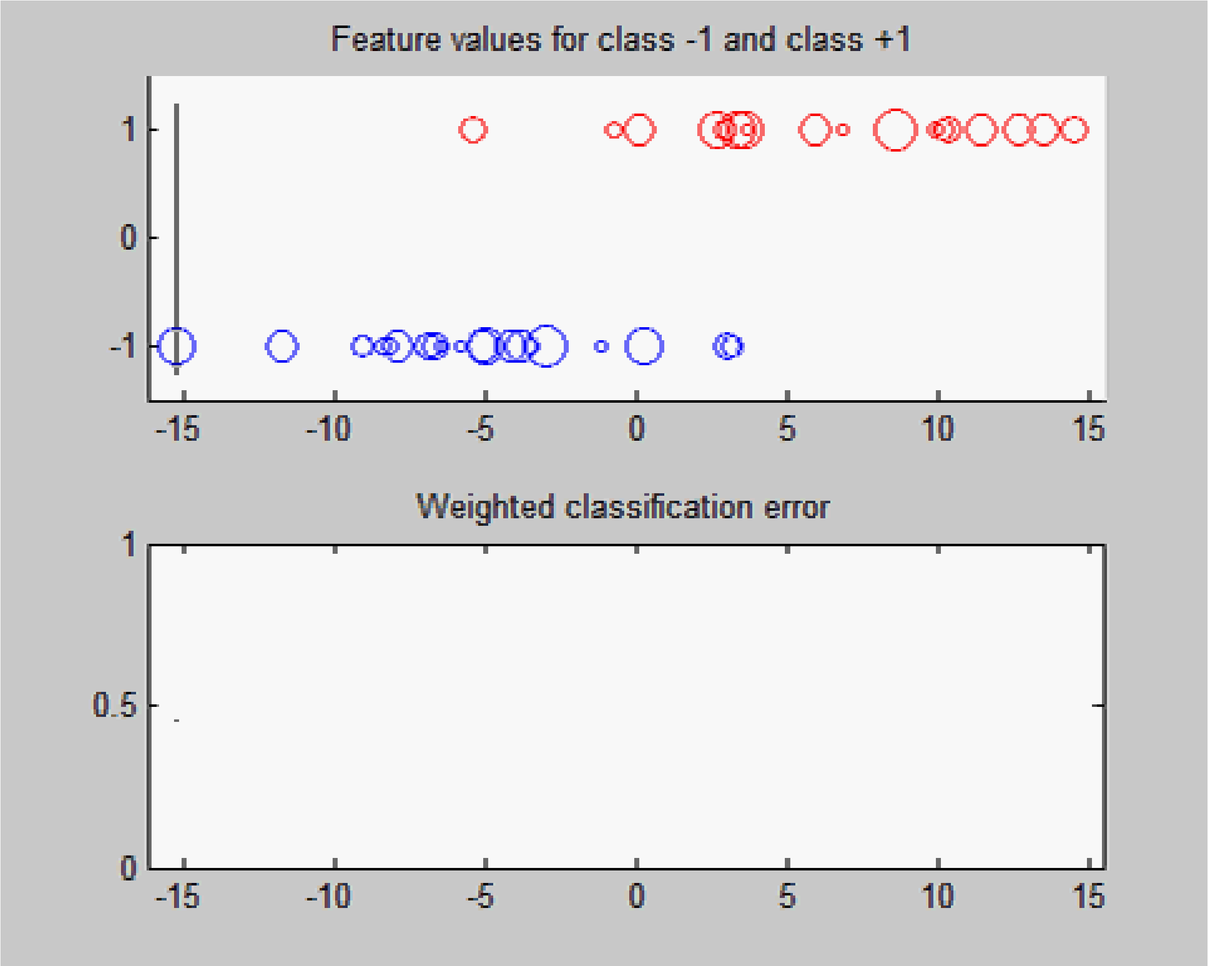
In general, if we obtain an error ε between 0.5 and 1.0, we can switch polarity and get the error 1.0 – ε, which is smaller than 0.5.



Brute force optimization

$$\min_{\tau,p} \varepsilon(\tau, p) = \sum_{i=1}^{N} d_i \, I\left(y_i \neq h(x_i; \tau, p)\right)$$

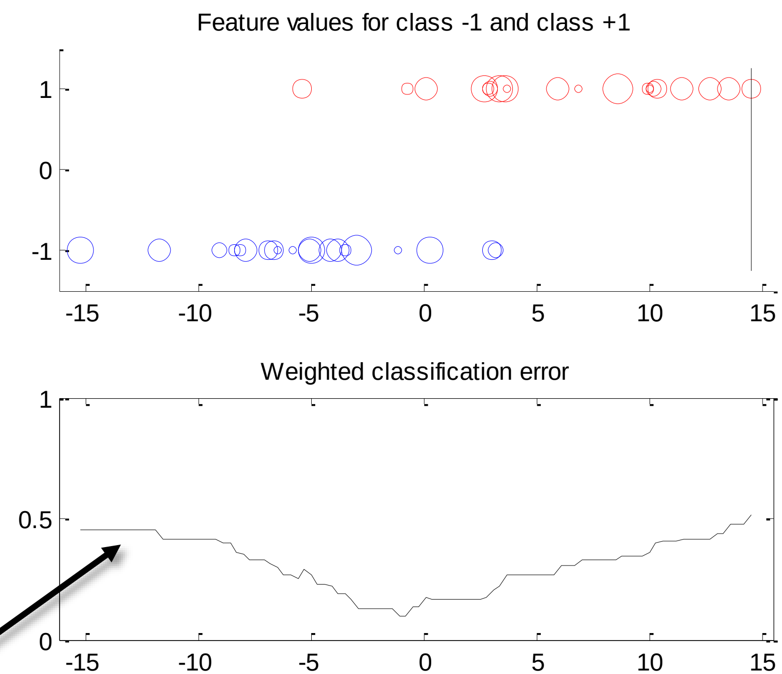
Movie!



Brute force optimization

$$\min_{\tau, p} \varepsilon(\tau, p) = \sum_{i=1}^{M} d_i \, I\left(y_i \neq h(x_i; \tau, p)\right)$$

Cost-function jumps at the x_i:s. Enough to test all thresolds in the set $\tau \in \{x_i\}_{i=1}^{N}$ and see which one gives the smallest error.



Cost function is piece-wise constant!

Brute force optimization

$$\text{Training samples } \mathbf{x}_i = \begin{pmatrix} x_{1,i} \\ \vdots \\ x_{N,i} \end{pmatrix}, i = 1 \ldots M$$

training samples

feature components

Pseudo code:

ε_{min} = inf;
for all feature components k = 1:N
  for all thresholds $\tau \in \{x_{k,i}\}_{i=1}^{M}$

$$\varepsilon(\tau, p=1) = \sum_{i=1}^{M} d_i \, I(y_i \neq h(x_i; \tau, p=1))$$

    if ε>0.5
      p = -1;
      ε = 1 - ε;
    end
   if ε< ε_{min} ...... **end**
  end
end

Discrete AdaBoost

Freund & Schapire, 1995

Training data: $\{\mathbf{x}_i, y_i\}_{i=1}^{M}$, $y_i \in \{-1,+1\}$

Initialization: $d_1(i) = \frac{1}{M}, T = \#\,\text{base classifiers}$

for t = 1 to T

Find weak classifier $h_t(\mathbf{x})$ = {-1,+1} that minimizes the weighted classification error:

$$\varepsilon_t = \sum_{i=1}^{M} d_t(i) I(y_i \neq h_t(\mathbf{x}_i))$$

Update weights:

$$d_{t+1}(i) = d_t(i) e^{-\alpha_t y_i h_t(x_i)}, \text{ where } \alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$$

and renormalize so that $\sum_{i=1}^{M} d_{t+1}(i) = 1$

end

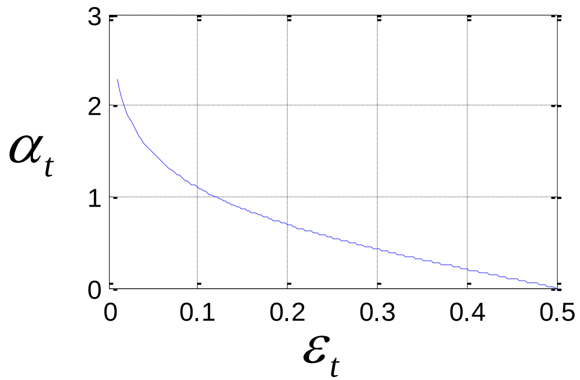
Final strong classifier: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \mathrm{h}_t(\mathbf{x})\right)$

Discrete AdaBoost

- <u>Discrete</u> output from the weak classifier
  $h_t(\mathbf{x}) = \{-1,+1\}$
- Weight update

$$d_{t+1}(i) = d_t(i)e^{-\alpha_t y_i h_t(\mathbf{x}_i)} = \begin{cases} d_t(i)e^{-\alpha_t} & \text{If } \mathbf{x}_i \text{ correctly classified} \\ d_t(i)e^{\alpha_t} & \text{If } \mathbf{x}_i \text{ wrongly classified} \end{cases}$$

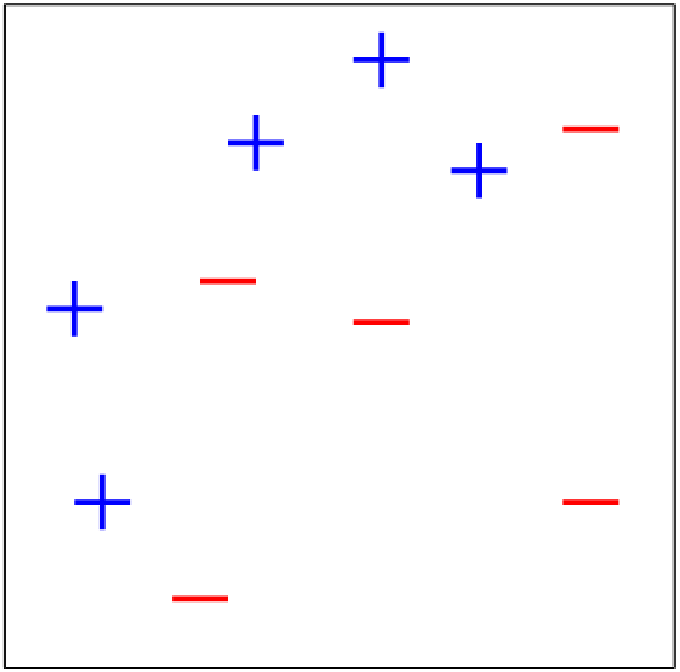
- Performance of weak classifier: $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$



Final strong classifier: $H(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T}\alpha_t \mathrm{h}_t(\mathbf{x})\right)$

Discrete AdaBoost – Toy example

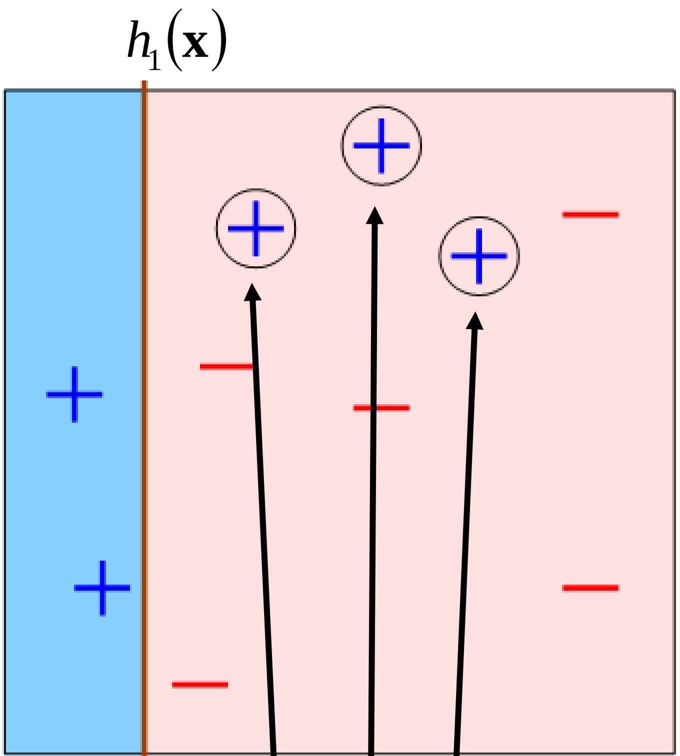
$$\text{Initial weights } d_1(i) = \frac{1}{10}, \mathrm{T} = 3$$

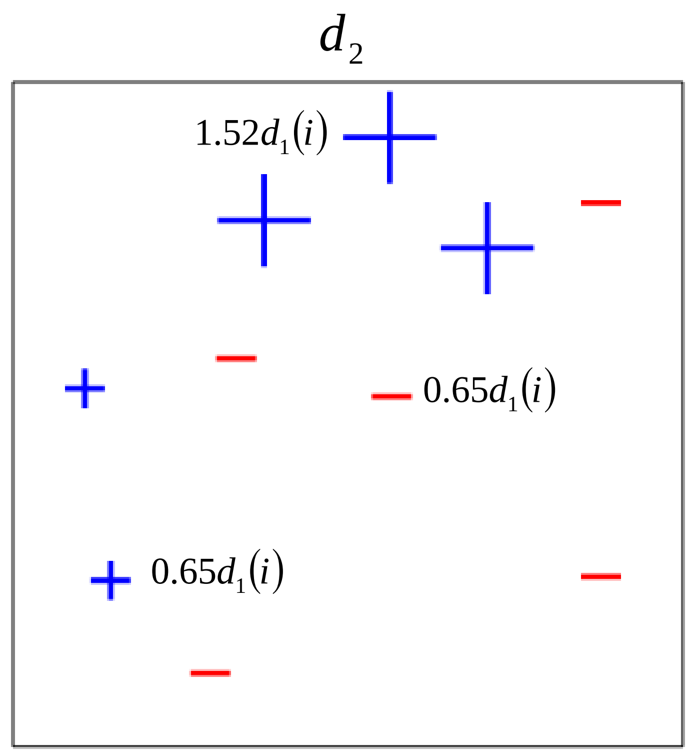


$$10 \text{ training samples } \mathbf{x}_{\mathrm{i}}, \mathrm{i} = 1\ldots10$$

Images borrowed from R. Schapire, ”A Boosting Tutorial”

Discrete AdaBoost – Toy example





$$d_1(i) = \frac{1}{10}$$

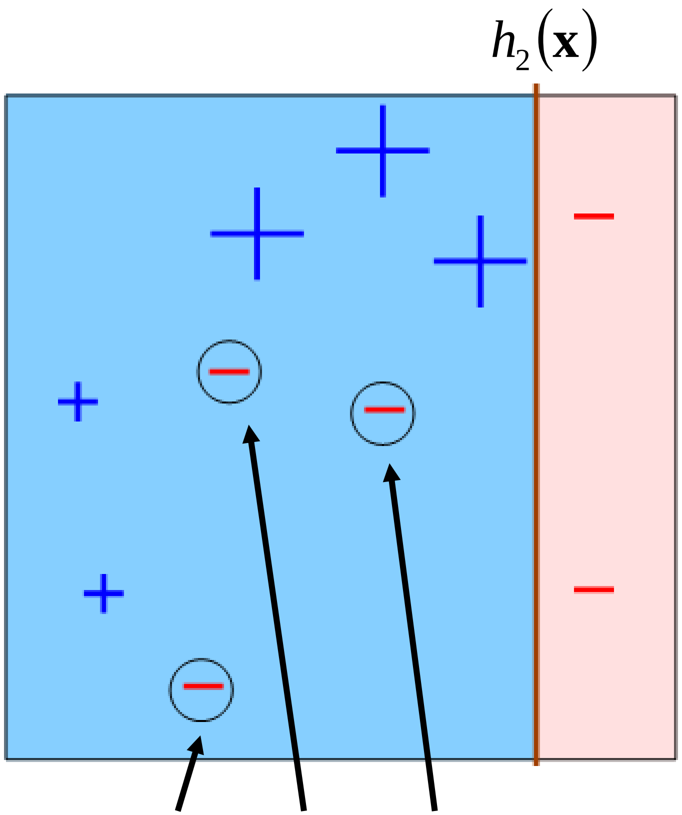
$$\varepsilon_1 = \left(\frac{1}{10} + \frac{1}{10} + \frac{1}{10}\right) = 0.3$$

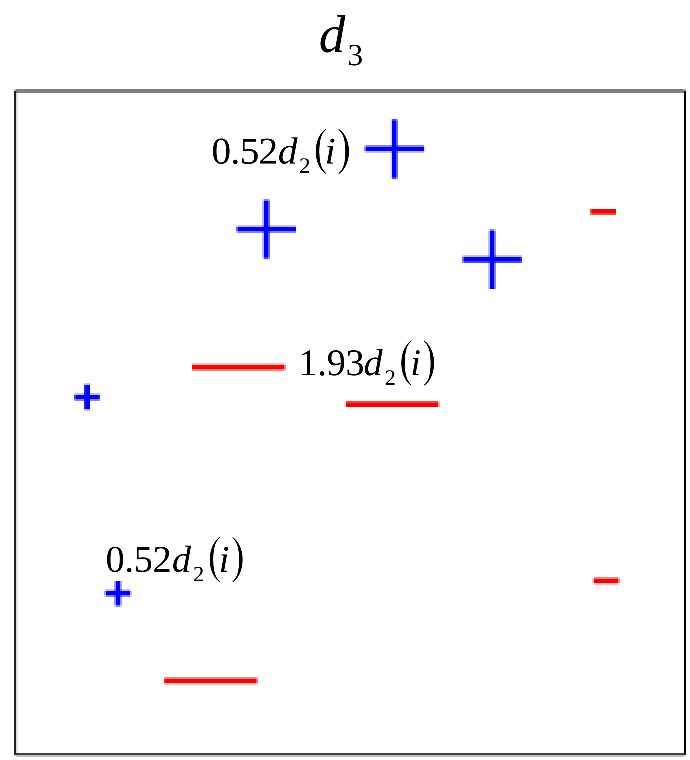
$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-0.3}{0.3}\right) = 0.42$$

$$d_2(i) = \begin{cases} d_1(i)e^{-0.42} = 0.65d_1(i) & \text{If } \mathbf{x}_i \text{ correctly classified} \\ d_1(i)e^{0.42} = 1.52d_1(i) & \text{If } \mathbf{x}_i \text{ wrongly classified} \end{cases}$$

Normalize d_2!

Discrete AdaBoost – Toy example





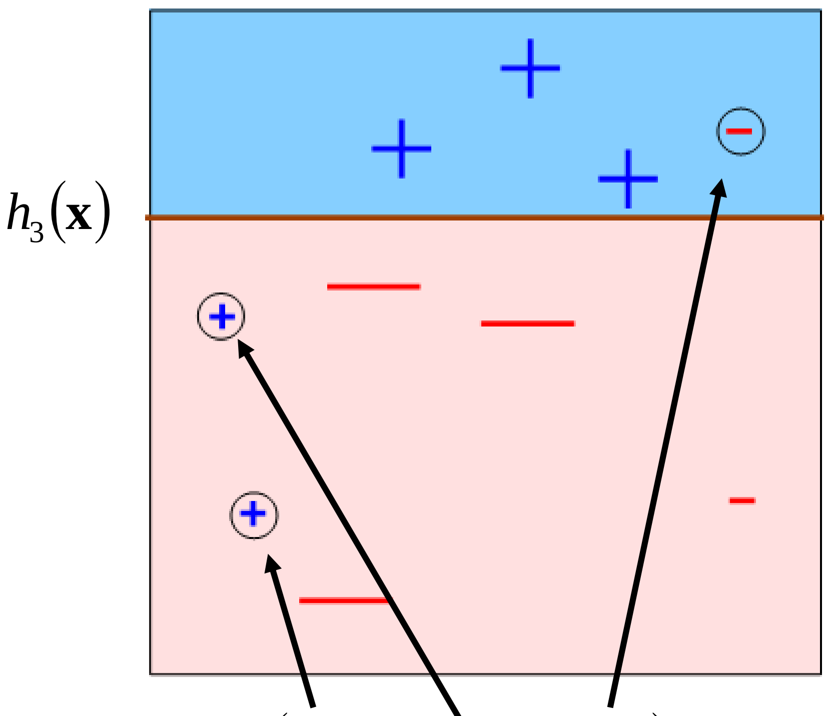
$$\varepsilon_2 = \left(d_2(j) + d_2(k) + d_2(l)\right) = 0.21$$

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.21}{0.21}\right) = 0.66$$

$$d_3(i) = \begin{cases} d_2(i)e^{-0.66} = 0.52d_2(i) & \text{If } \mathbf{x}_i \text{ correctly classified} \\ d_2(i)e^{0.66} = 1.93d_2(i) & \text{If } \mathbf{x}_i \text{ wrongly classified} \end{cases}$$

Normalize d_3!

Discrete AdaBoost – Toy example

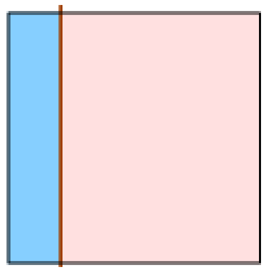
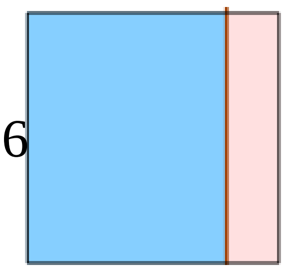
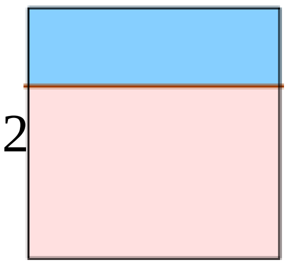


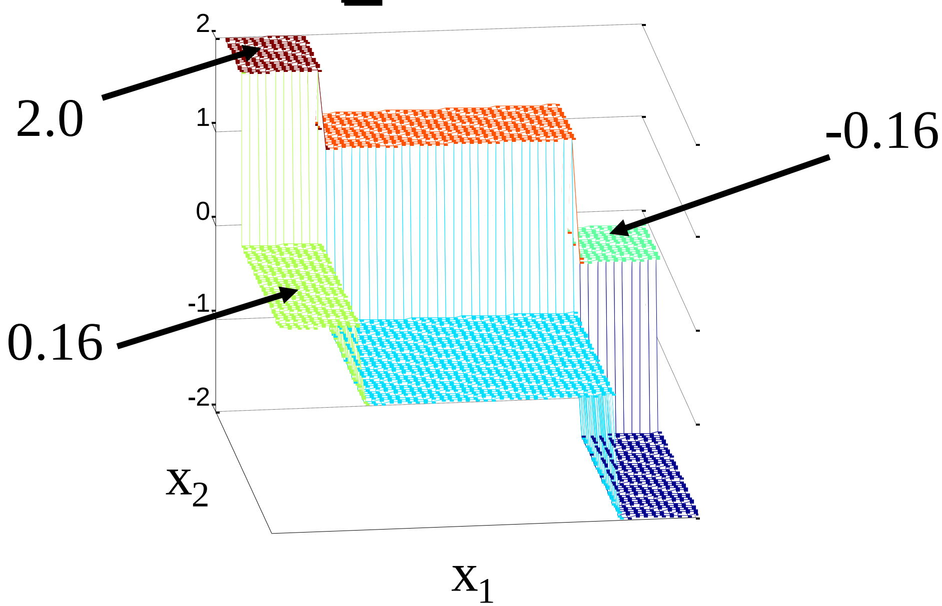
$$\varepsilon_3 = \left(d_3(p) + d_3(q) + d(r)\right) = 0.14$$

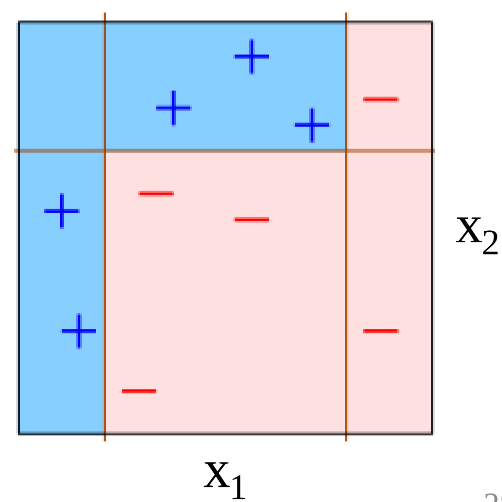
$$\alpha_3 = \frac{1}{2}\ln\left(\frac{1-0.14}{0.14}\right) = 0.92$$

Discrete AdaBoost – Toy example

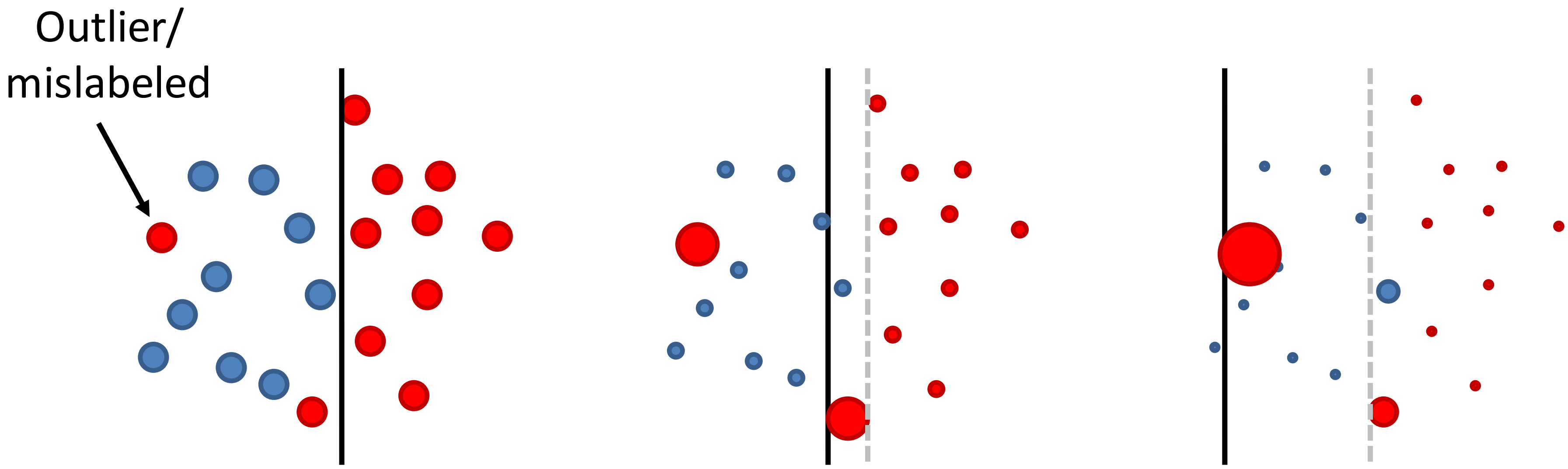
Final strong classifier: $H(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t \mathrm{h}_t(\mathbf{x})\right)$

$\mathrm{H}(\mathbf{x}) = \mathrm{sign}\Big[$ 0.42  + 0.66  + 0.92  $\Big]$



$\mathrm{H}(\mathbf{x}) =$ 

Problem - Outliers



- Outliers gain weight exponentially
- Will eventually result in bad weak classifiers
- May ruin the strong classifier

Outlier strategies

- Keep an eye on the weights (plot them!)
- Weight trimming
  - Don't allow weights larger than a certain threshold
  - Disregard training samples with too large weights
- Use alternative weight update schemes with less agressive increases for misclassified training data
  - LogitBoost
  - GentleBoost

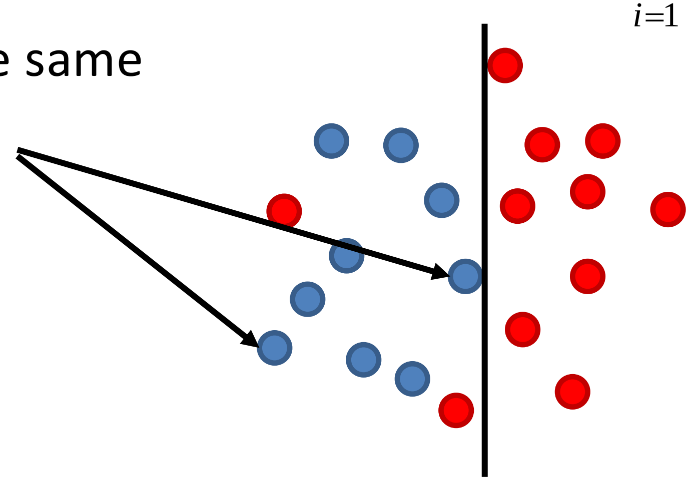
A possible improvement

Discrete AdaBoost

Find weak classifier $h_t(\mathbf{x}) = \{-1,+1\}$ that minimizes the weighted classification error:

$$\varepsilon_t = \sum_{i=1}^{M} d_t(i) I\left(y_i \neq h_t(\mathbf{x}_i)\right)$$

Will receive the same weight update



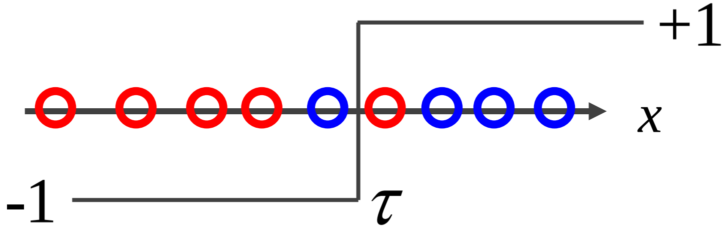
- In discrete AdaBoost, the weight update for each sample ignores how correct/wrong that sample is. Instead, all correct/wrong samples are updated using an average over all samples!
- The discrete weak classifier $h_t(\mathbf{x}) = \{-1,+1\}$ throws away important information!

A possible improvement, cont.

Idea: Let the weak classifier $h_t(\mathbf{x})$ output a real number, where the sign indicates the class label and the magnitude a confidence.

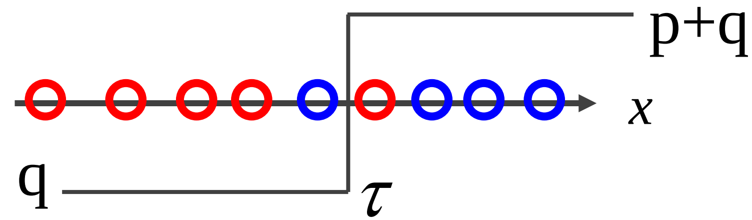
Discrete

$$h(x) \in \{-1, +1\}$$

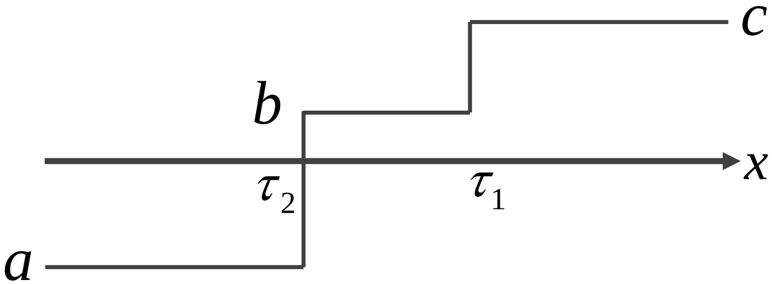


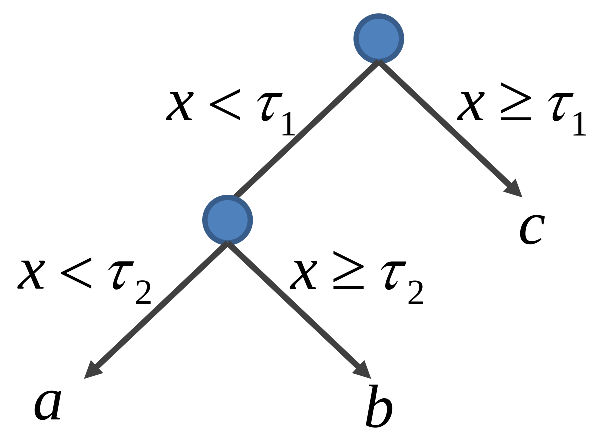
Real

$$h(x; p, q, \tau) = p \cdot (x \geq \tau) + q$$



Regression trees





AdaBoost modifications

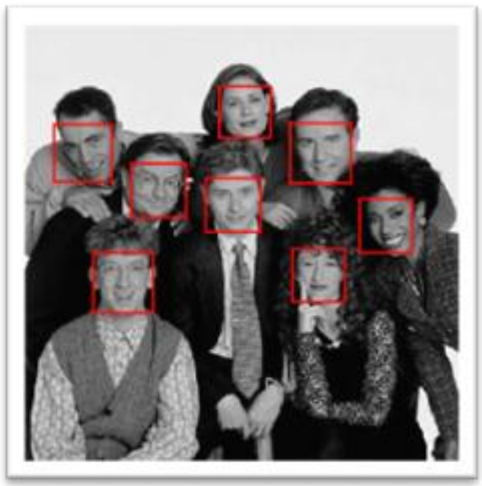
- Use real-valued outputs from weak classifiers $h_t(\mathbf{x})$
  - Real AdaBoost (as opposed to Discrete AdaBoost)
- More robustness against outliers/noise:
  - LogitBoost
  - GentleBoost

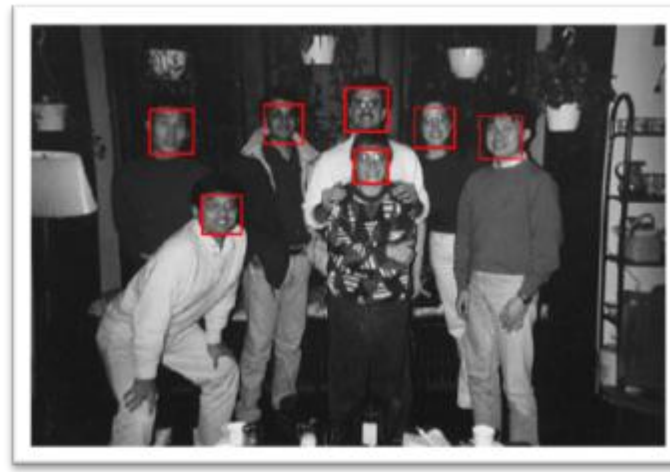
Summary AdaBoost

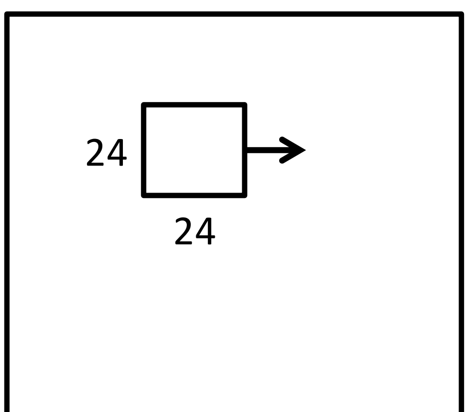
- Nonlinear classifier that is easy to implement
- Easy to use - just one parameter (T)
- Can obtain performance similar to SVM
- Inherent feature selection
- Slow to train – fast to classify (real-time)
- Look out for outlier problems
- Try applet:
  - http://cseweb.ucsd.edu/~yfreund/adaboost/

Real-time object detection

P. Viola and M. Jones “*Rapid Object Detection using a Boosted Cascade of Simple Features*”, 2001



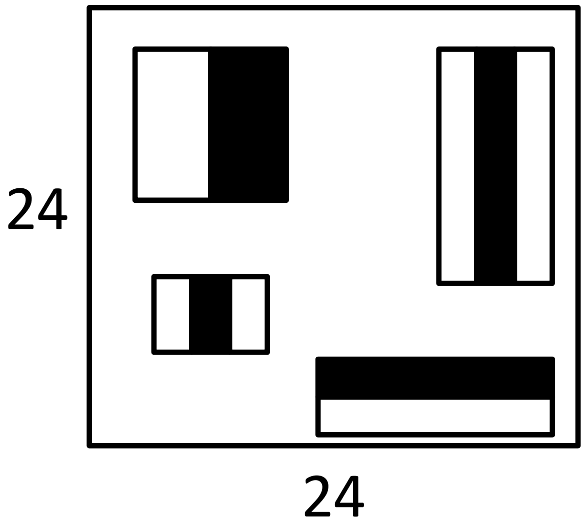




Sweep a sub-window over the image. for each position, determine if the sub-window contains a face or not.

Haar-features

Rectangle filters



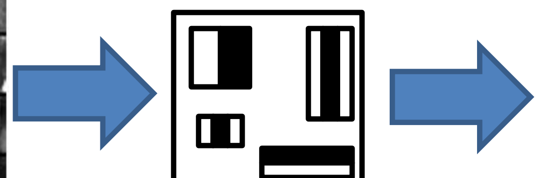
From the sub-window, calculate contrast features (Haar features) at different locations and scales, and in different orientations. LOTS of different combinations, can be several 100,000 features!

Train detector with AdaBoost

As described previously!

24 x 24 pixels face images

Apply Haar filters
to each image

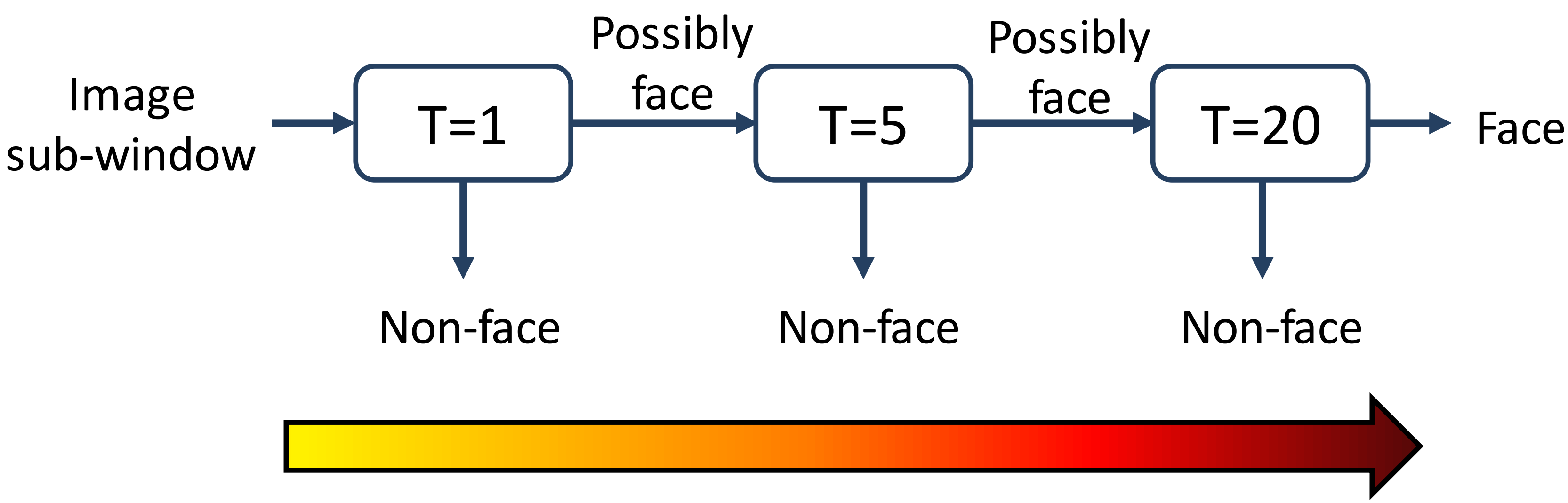


$$\mathbf{x}_i = \begin{pmatrix} x_{1,i} \\ \vdots \\ x_{N,i} \end{pmatrix}, i = 1 \ldots M$$

Similar with non-face images to obtain negative examples.

Detector cascade for real-time

Non-face windows MUCH more frequent than face windows.
Focus on quickly rejecting non-face windows!

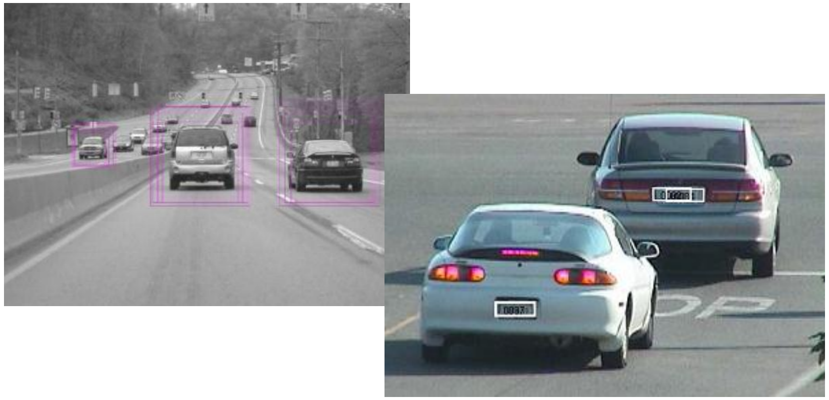


Increasing classifier complexity to separate more and more difficult non-face samples from face samples.

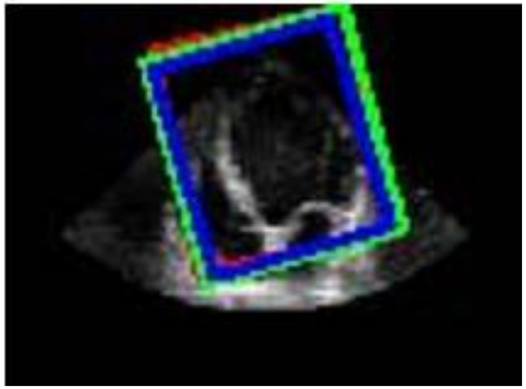
Real-time applications

Face detection

Pedestrian detection



Car & license plate detection



Heart detection

D. Lee, "*Boosted Classifier for Car Detection*"
L. Dlagnekov, "*License Plate Detection Using AdaBoost*"

S. Zhou et al., "*A boosting regression approach to medical anatomy detection*"