Data Mining

Lecture 10
Ensemble Learning

http://www.informatik.uni-hamburg.de/WTM/

Ensemble Learning

- So far – learning methods learn a single hypothesis (model), chosen form a hypothesis space to make predictions.

- ***"There ain't no such thing as a free lunch"***
  - No single algorithm wins all the time!

- Ensemble learning
  - select a collection (*ensemble*) of hypotheses (models) and combine their predictions.

- **Example:** Generate 100 different decision trees from the same or different training set and have them vote on the best classification for a new example.

Value of Ensembles

- Key motivation: ***reduce*** the ***error rate***!
  Hope: it is less likely that an *ensemble* misclassifies an example

- **Examples**: Human ensembles are demonstrably better:
  - How many jelly beans in the jar?:
    Individual estimates vs. group average
  - Who Wants to be a Millionaire: Audience vote
  - Diagnosis based on multiple doctors' majority vote

- **Theory behind**: We combine multiple
  independent and ***diverse*** decisions
  - each is at least more accurate than random guessing
  - random errors cancel each other out
  - correct decisions are reinforced

Achieving Diversity (1)

- Using different learning ***algorithms***
- Using different ***hyper-parameters*** in the same algorithm (e.g. different number of hidden nodes in ANNs)
- Using different ***input representations***, e.g. different subsets of input features

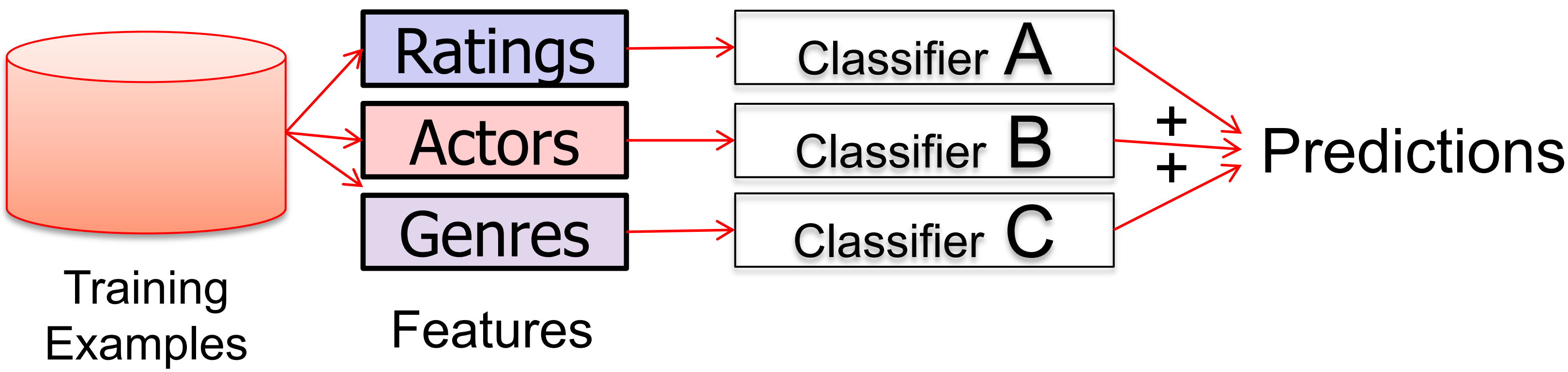
  ← *diversity largely hand-designed*

- Using different training ***subsets*** of input data, e.g. known procedures of bagging, boosting, and cascading.

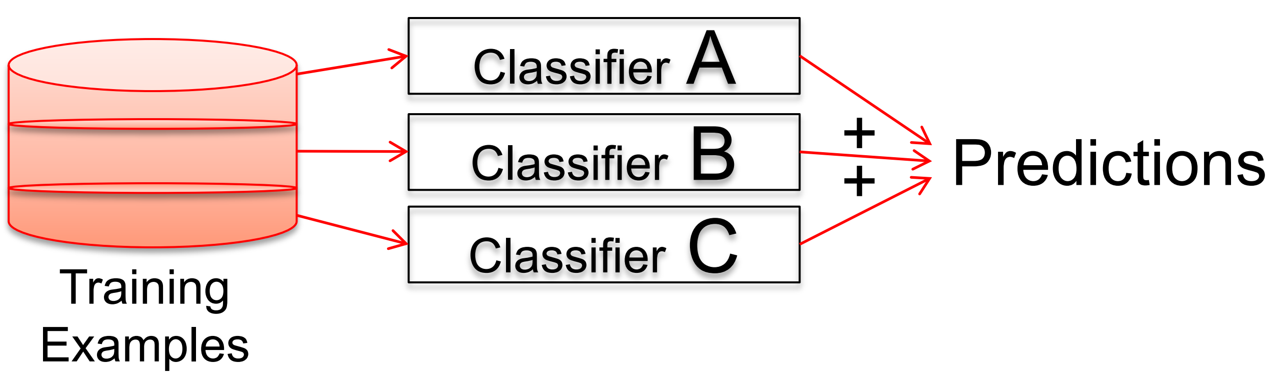
  ← *diversity easily achieved automatically*

Achieving Diversity (2)

- Diversity from *differences in input features*:

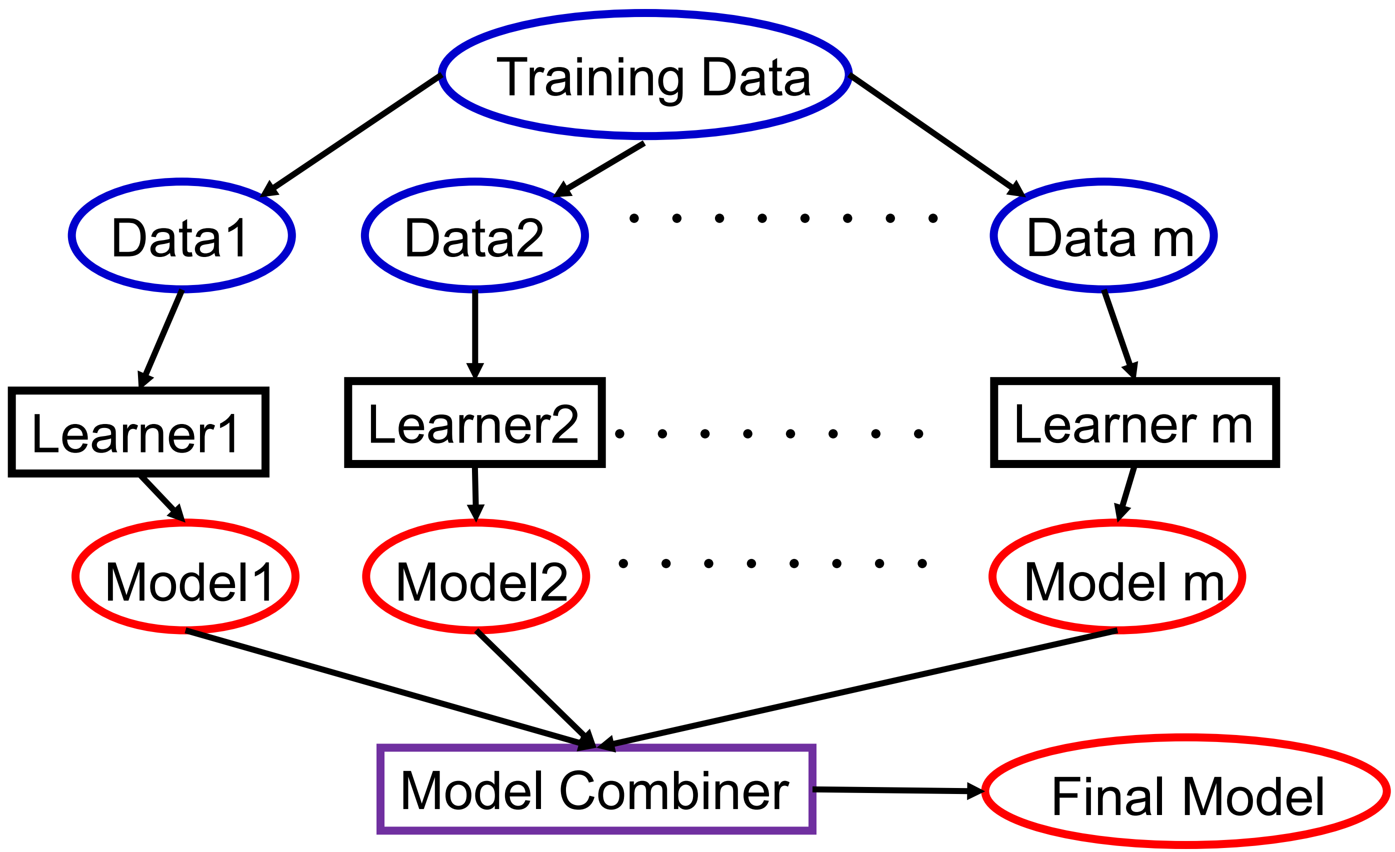


- *Divide up training data* among models :



Learning Ensembles

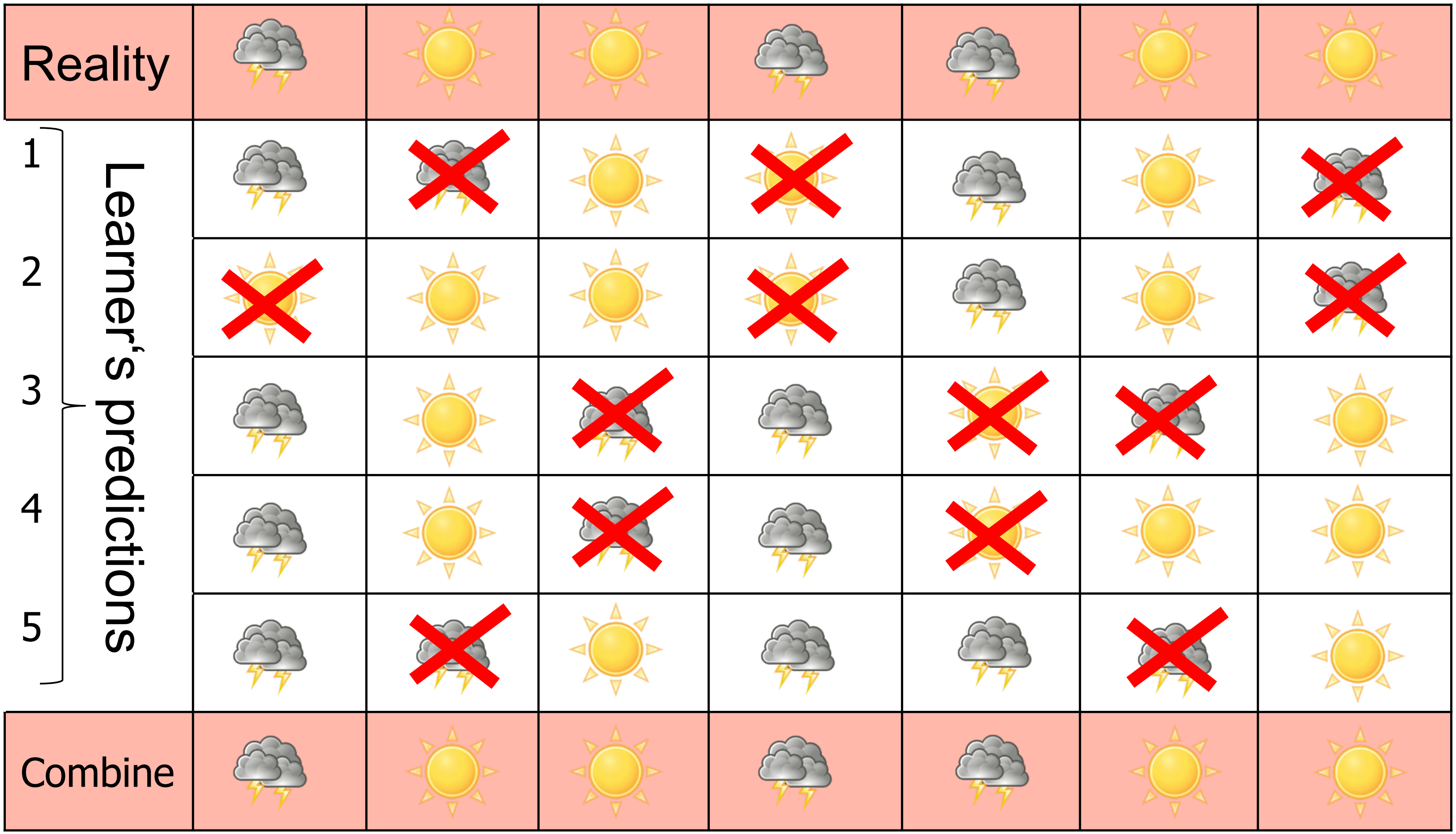
- **Example**: learn multiple alternative definitions of a concept using ***different training data***:



How to Combine the Outputs of Base Learners?

- ***Global approach*** is through fusion – the outputs of all learners are combined by *voting, averaging, or stacking*
- ***Local approach*** is based on learner selection – it looks for the input and chooses for learners (one or more) responsible for generating the output
- ***Multistage combination*** use a serial approach where the next learner is trained with or tested on instances only where previous learners failed, or were inaccurate

Example: Weather Forecast

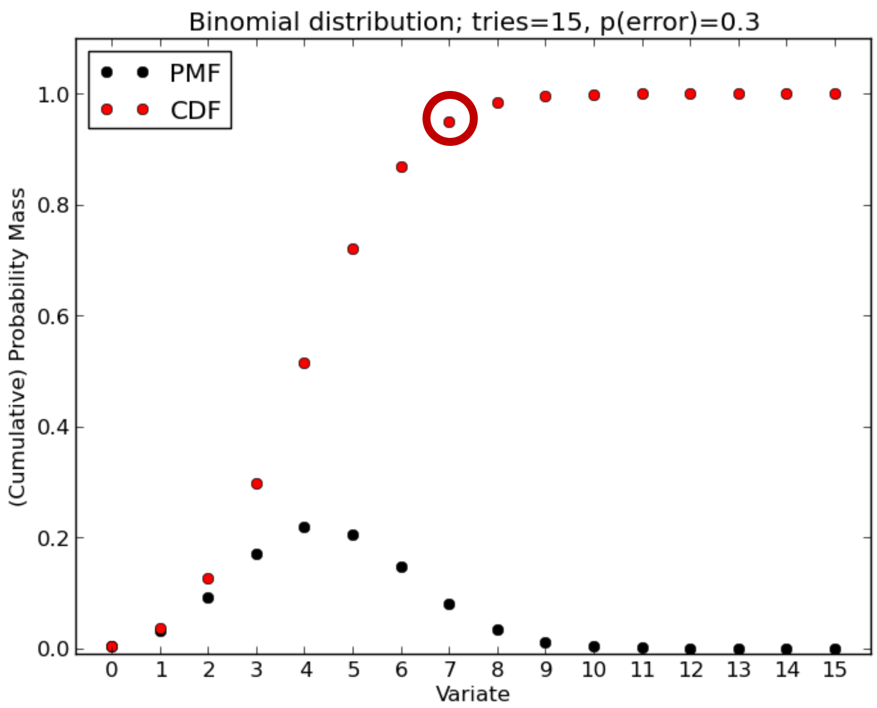


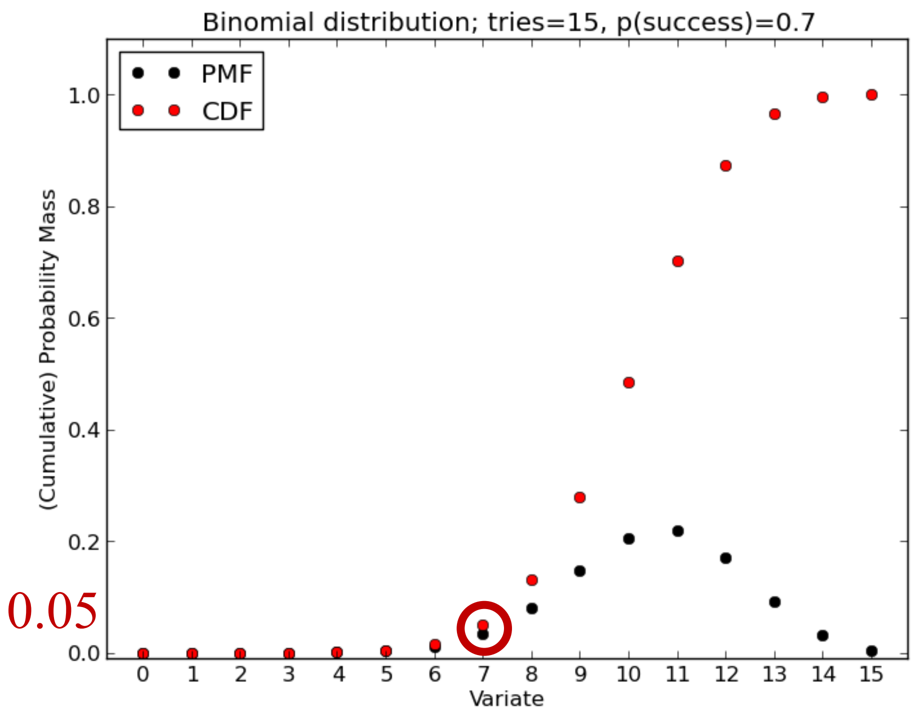
- Combine decisions of multiple models using *voting* procedure!

Ensembles Give Better Results

- Majority vote of *n*=15 classifiers, error rate each ε=0.3:

$$\varepsilon_{ensemble} = \sum_{i=8}^{15} \binom{15}{i} \cdot \varepsilon^{i} (1-\varepsilon)^{15-i} = 0.05$$

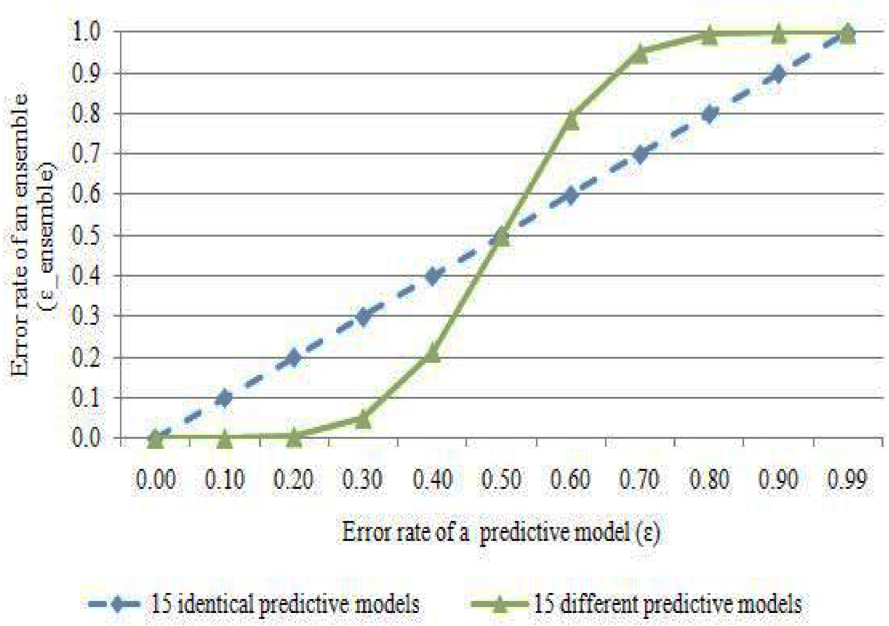




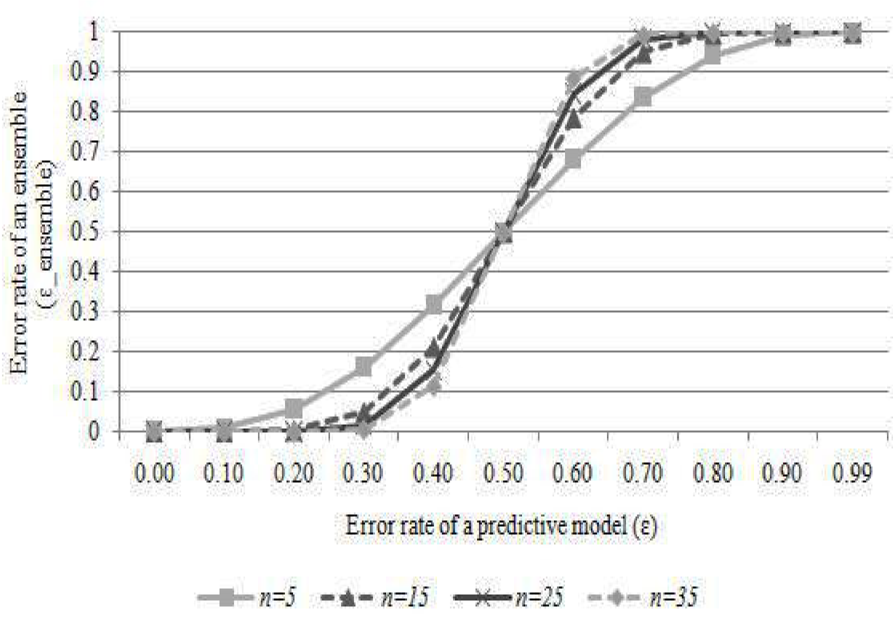
Ensembles Give Better Results

- Majority vote of *n*=15 classifiers, error rate each ε=0.3:

$$\varepsilon_{ensemble} = \sum_{i=8}^{15} \binom{15}{i} \cdot \varepsilon^i (1-\varepsilon)^{15-i} = 0.05$$



(a) Identical predictive models vs. different predictive models in an ensemble



(b) The different number of predictive models in an ensemble

Global Approach: Voting is not Only Majority Voting?

- Voting is the simplest way of combining classifiers, it is a linear combination of outputs d_{ij} for j learners:

$$y_i = \sum w_j \cdot d_{ij} \quad \text{where} \quad w_j \geq 0 \quad \text{and} \quad \sum w_j = 1$$

- ***Alternatives*** for combination are:
  - Simple sum (equal weights)
  - Weighted sum
  - Median
  - Minimum or minimum
  - Geometric mean: $\sqrt[k]{d_1 \cdot d_2 \cdot \ldots \cdot d_k}$

Global Approach:
Rank-Level Fusion Method

- Four-class problem (a,b,c,d)?

Rank / score	Classifier 1	Classifier 2	Classifier 3
4	c	a	d
3	b	b	b
2	d	d	c
1	a	c	a

$$r_a = r_a(1) + r_a(2) + r_a(3) = 1 + 4 + 1 = 6$$
$$r_b = r_b(1) + r_b(2) + r_b(3) = 3 + 3 + 3 = 9$$
$$r_c = r_c(1) + r_c(2) + r_c(3) = 4 + 1 + 2 = 7$$
$$r_d = r_d(1) + r_d(2) + r_d(3) = 2 + 2 + 4 = 8$$

- The winner-class is ***b*** because it has the maximum overall rank

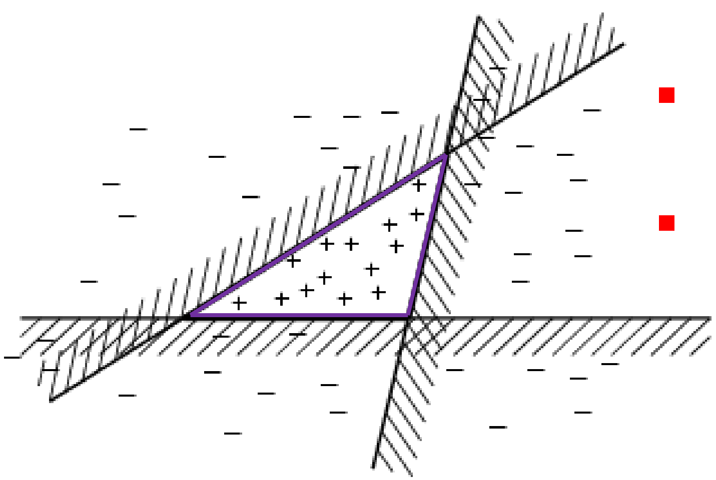
Local Approach:
Dynamic Classifier Selection

- ***Algorithm:***
  - Find the k nearest training points to the test input
  - Look at the accuracies of the base classifiers on these points,

    and
  - Choose the one that performs best on them (or vote over a few “competent” ones).

Ensemble Learning

- Another way of thinking about ensemble learning:
  - way of ***enlarging the hypothesis space,*** i.e., the ensemble itself is a hypothesis (the new hypothesis space is the set of all possible ensembles constructible form hypotheses of the original space)
- Increased power of ensemble learning:



- Three linear threshold hypotheses (positive examples on the non-shaded sides)
- Ensemble classifies as positive any example classified positively by all three
- The resulting triangular region hypothesis is not expressible by any of the base hypotheses

Homogenous Ensembles

- Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.
  - Data1 ≠ Data2 ≠ … ≠ Data m
  - Learner1 = Learner2 = … = Learner m
- Different methods for changing training data:
  - Bagging: Resample training data
  - Boosting: Reweight training data

Bagging: Bootstrap Aggregation (1)

- Training
  - Given a set D of *d* tuples
  - At each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (bootstrap) *
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns **X** to the class with the most votes
  - Prediction of continuous values: by taking the average value of each prediction for a given test sample

* *each set D_i is expected to have ~2/3 unique tuples and ~1/3 duplicates ≈ random (re)weighting of data*

Bagging: Bootstrap Aggregation (2)

- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noisy data: not considerably worse, more robust
  - Proven improved accuracy in prediction
  - Decreases error by ***decreasing the variance*** in the results due to ***unstable learners:*** algorithms (like decision trees and neural networks) whose output can change dramatically when the training data is slightly changed
  - Increases classifier stability, reduces variance!

(Breiman, 1996)

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses – weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - ***Weights*** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to ***pay more attention to the training tuples that were misclassified*** by M_i
  - The final M* combines the votes of each individual classifier, where the weight of each ***classifier's vote is a function of its accuracy***
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to achieve greater accuracy, but it also risks overfitting the model to misclassified data.

Boosting: Strong And Weak Learners (1)

- ***Strong Learner***
  - Take labeled data for training
  - Produce a classifier which can be ***arbitrarily accurate***
  - Strong learners are an objective of machine learning

- ***Weak Learner***
  - Take labeled data for training
  - Produce a classifier which is ***more accurate than random guessing***
  - Weak learners can be base classifiers for ensemble methods

Boosting: Strong And Weak Learners (2)

- ***Weak Learner:*** only needs to generate a hypothesis with a training accuracy greater than 0.5, i.e., < 50% error over any distribution
  - Strong learners are very difficult to construct
  - Constructing weaker learners is relatively easy

- Can a set of weak learners create a single strong learner?
  - Yes! Boost weak classifiers to a strong learner! (Shapire,1990)

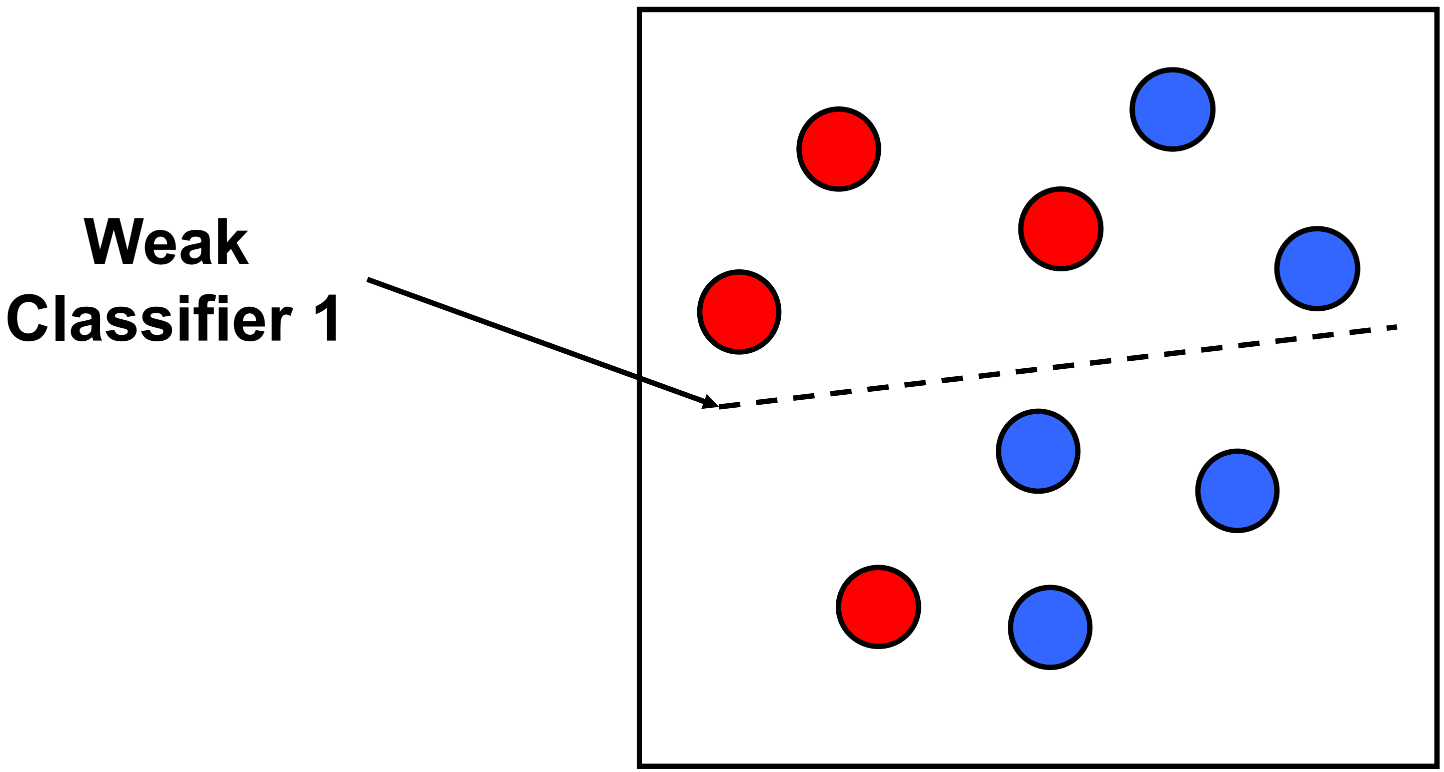
Boosting: Construct Weak Classifiers

Idea: Focus on difficult samples which are not correctly classified in the previous steps

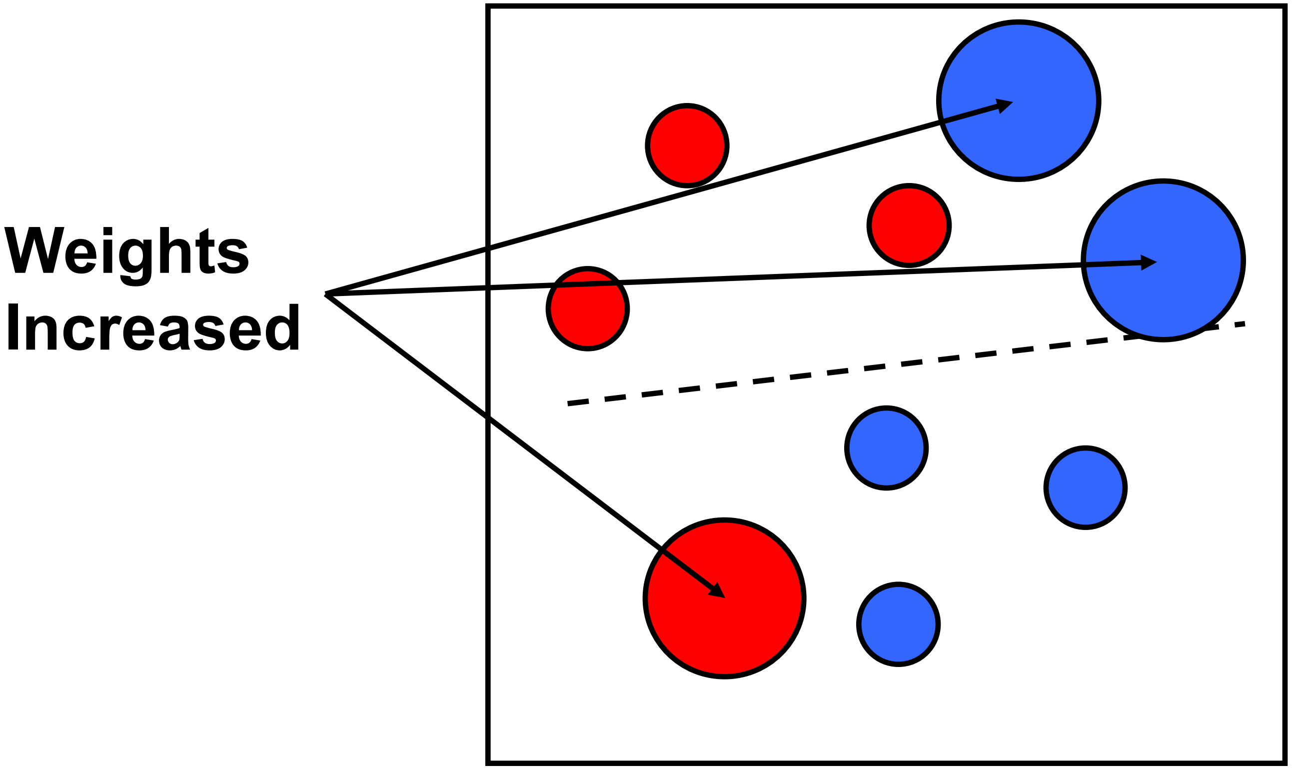
Use different data distribution:

- Start with ***uniform weighting*** of samples
- During each step of learning
  - Increase weights of the samples which are not correctly learned by the weak learner
  - Decrease weights of the samples which are correctly learned by the weak learner

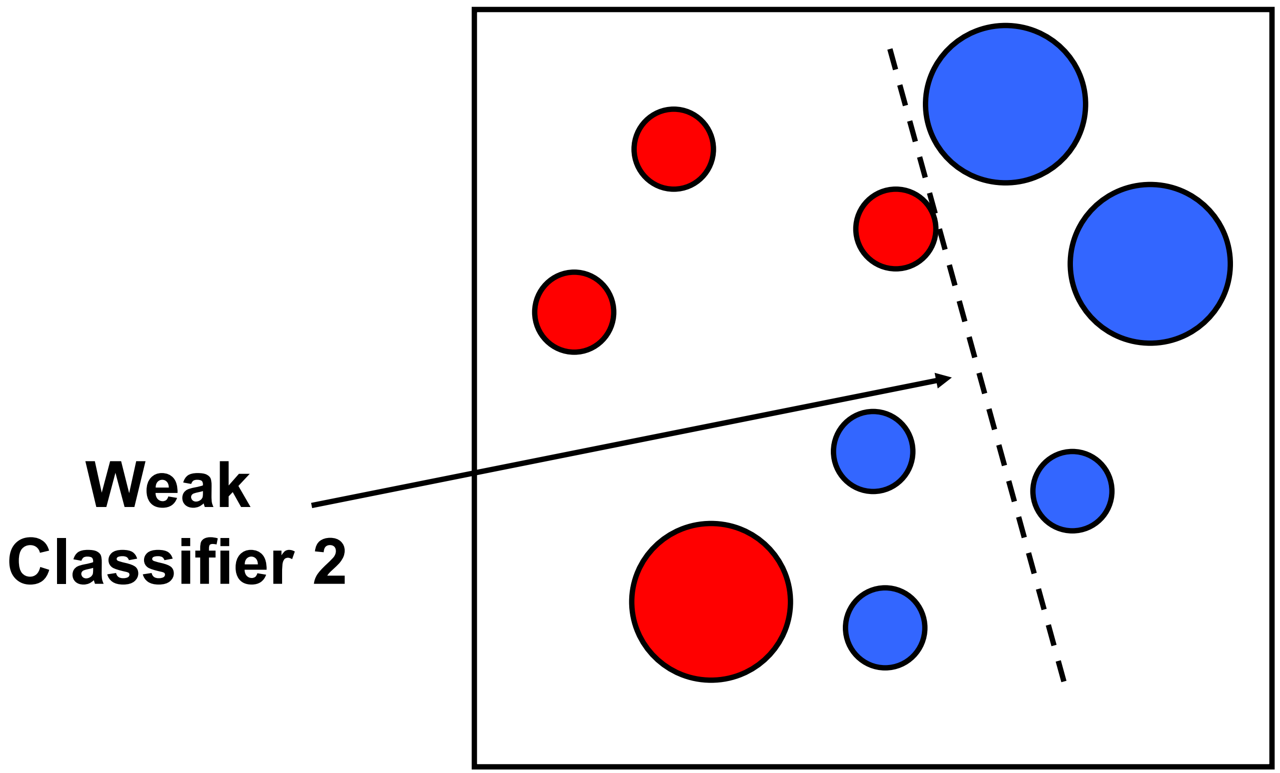
Boosting Idea



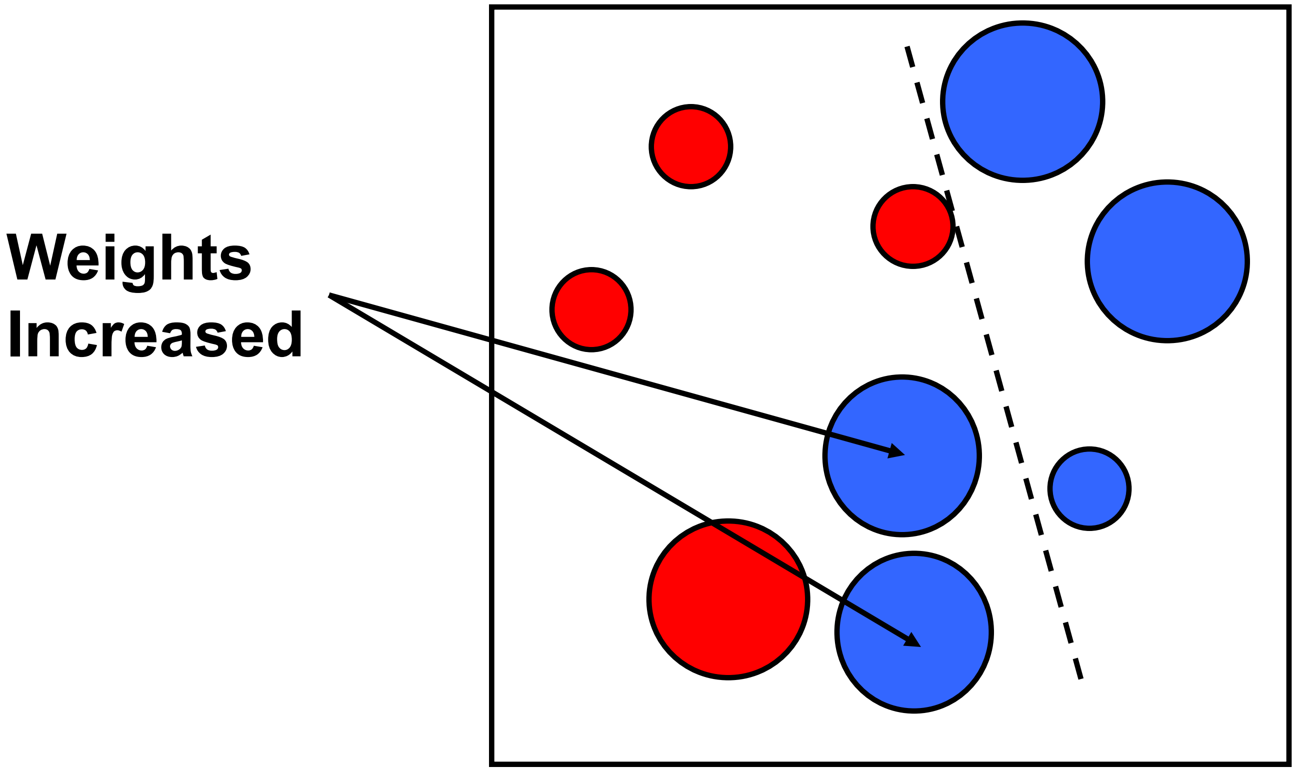
Boosting Idea



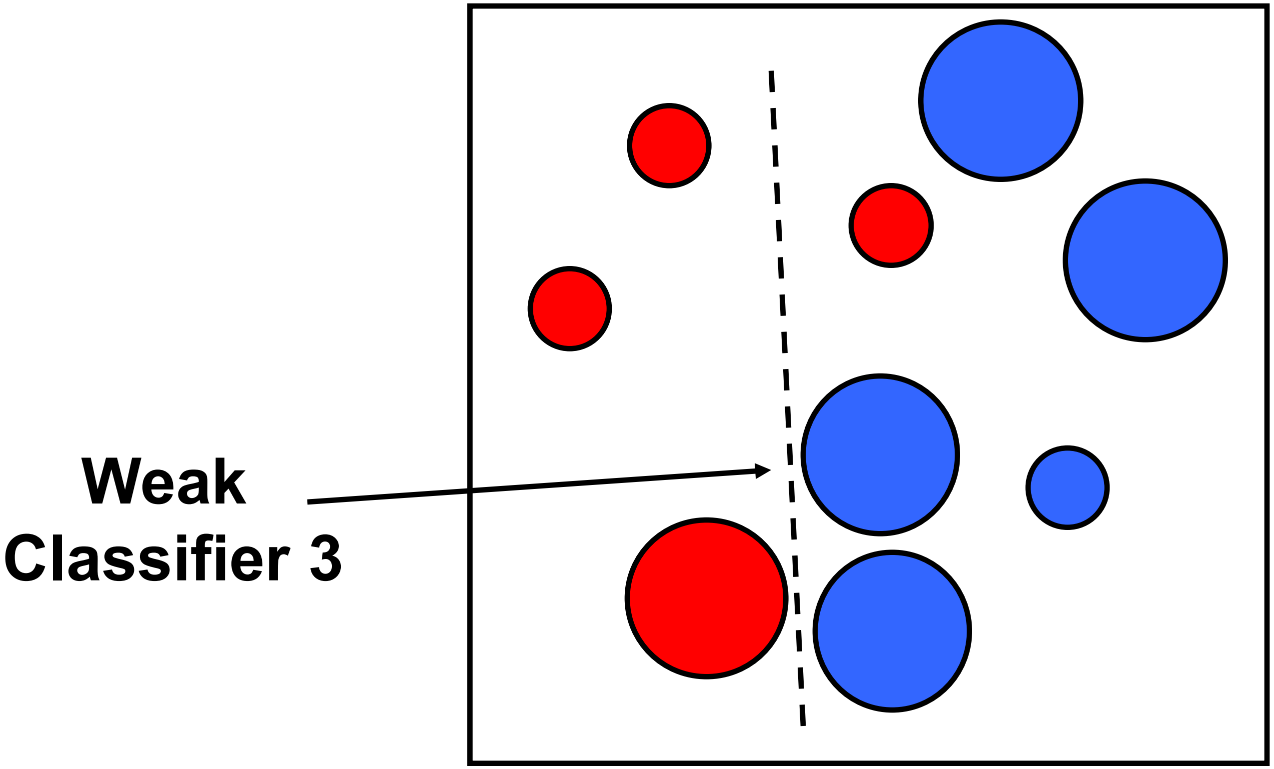
Boosting Idea



Boosting Idea

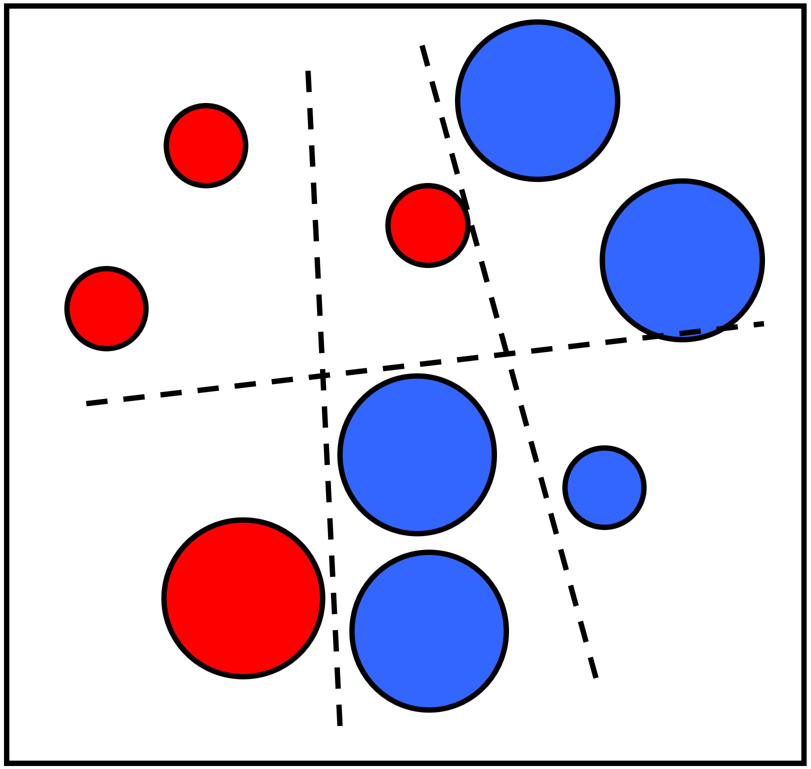


Boosting Idea



Boosting Idea

Final classifier is a combination of weak classifiers



Boosting: Combine Weak Classifiers

- Idea: Better weak classifier gets a larger weight!

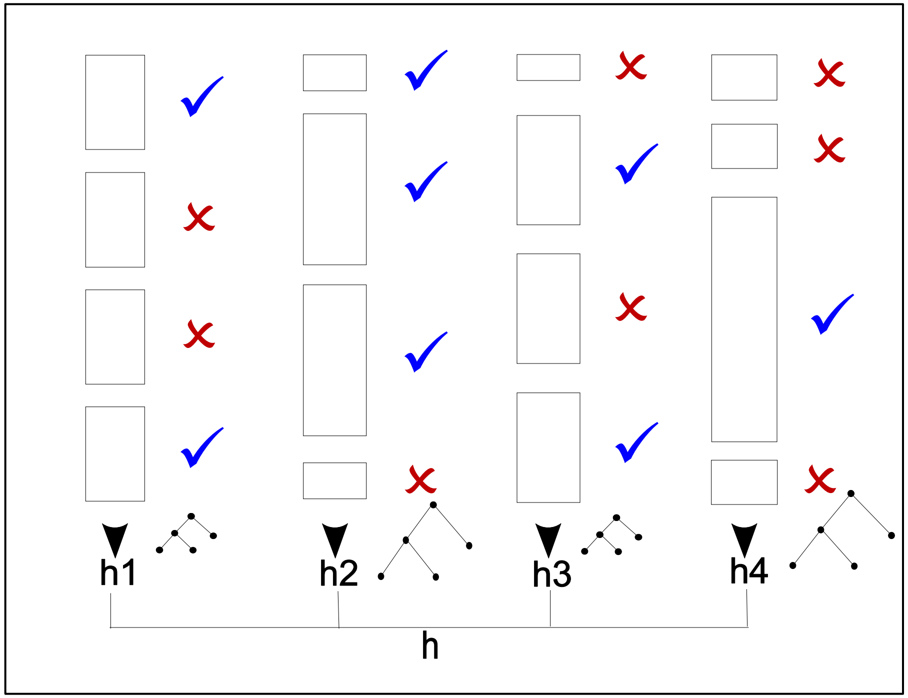
- Weighted Voting
  - Construct ***strong classifier*** by ***weighted voting*** of the weak classifiers
  - Weight of each learner is directly proportional to its accuracy

AdaBoost: Adaptive Boosting

- Does not need to know the number of weak classifiers in advance
- Does not need to know error bounds on the weak classifiers, unlike previous boosting algorithms

AdaBoost: Adaptive Boosting

- Each rectangle corresponds to an example, with weight proportional to its height.
- Crosses correspond to misclassified examples.
- Size of decision tree indicates the weight of that hypothesis in the final ensemble.



Initialization

Given : $(x_1, y_1), \ldots (x_n, y_n)$, where $x_i \in X, y_i \in Y = \{-1,+1\}$

Initialze distribution (weight) $D_{t=1}(i) = 1/n$; such that $n = M + L$

M = number of positive $(+1)$ examples; L = number of negative (-1) examples

Main Loop

For $t = 1, \ldots T$

{ Step1a : Find the classifier $h_t : X \to \{-1,+1\}$ that minimizes the

error with respect to D_t, that means : $h_t = \arg\left[\min_q(\varepsilon_q)\right]$

Step1b : error $\varepsilon_t = \sum_{i=1}^{n} D_t(i) * I_{[h_t(x_i) \neq y_i]}$, where $I_{[h_t(x_i) \neq y_i]} = \begin{cases} 1 \text{ if } [h_t(x_i) \neq y_i] \text{(classified incorrectly)} \\ 0 \quad \textit{otherwise} \end{cases}$

checking step : prerequisite : $\varepsilon_t < 0.5$: (error smaller than 0.5 is ok) otherwise stop.

Step2 : $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$, α_t = weight (or confidence value).

*Step*3 : $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, see next slide for explanation

Step4 : Current total cascaded classifier error $CE_t = \sum_{j=1}^{j=t} E_j\left(t, \alpha_\tau, h_\tau(x_i)\right)$

while the current classifier error $E_\tau = \frac{1}{n}\sum_{\tau=1}^{n} I\left(t, \alpha_\tau, h_\tau(x_i)\right)$,

and $I(\,)$ is defined as follows :

If x_i is correctly classified by the current cascaded classifier , i.e.

$y_i = sign\left(\sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)\right)$, hence error $I\left(t, \alpha_\tau, h_\tau(x_i)\right) = 0$

If x_i is incorrectly classified by the current cascaded classifier i.e.

$y_i \neq sign\left(\sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)\right)$, hence error $I\left(t, \alpha_\tau, h_\tau(x_i)\right) = 1$

If $CE_t = 0$ then T = t, break;

}

The output $o_t(x_i) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)$, and $S\left(t, \alpha_\tau, h_\tau(x_i)\right) = \begin{cases} 1 \text{ if } y_i = sign(o_t(i)) \\ 0 \quad \textit{otherwise} \end{cases}$

where Z_t = *normalization* factor, so D_t is a *probability distrubution*

$$Z_t = \sum_{i=1}^{n\_correctly\_classified} correct\_weight + \sum_{i=1}^{n\_incorrectly\_classified} incorrrect\_weight$$

$$= \sum_{i=1}^{n\_correctly\_classified} D_t(i) e^{-\alpha_t} y_i h_t(x_i) + \sum_{i=1}^{n\_incorrectly\_classified} D_t(i) e^{\alpha_t} y_i h_t(x_i)$$

Strong Classifier

The final strong classifier $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

enlarged versions on the following slides

Initialization

Given $(x_{1,} y_1),..(x_n, y_n)$, where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialze weights of samples $D_{t=1}(i) = 1/n$;

such that $\mathrm{n} = \mathrm{M} + \mathrm{L}$

M = number of positive $(+1)$ examples;

L = number of negative (-1) examples

Adapted from:
Kin Hong Wong: Adaboost for building robust classifiers. http://appsrv.cse.cuhk.edu.hk/~khwong/

Main Loop (Steps 1, 2, 3)

For $t = 1,\ldots T$

$\{$

Step1a : Find the classifier $h_t : X \to \{-1,+1\}$ that minimizes the

error with respect to D_t : $h_t = \arg\left[\min_q(\varepsilon_q)\right]$

Step1b : error $\varepsilon_t = \sum_{i=1}^{n} D_t(i) * I_{[h_t(x_i)\neq y_i]}$,

$$\text{where } I_{[h_t(x_i)\neq y_i]} = \begin{cases} 1 \text{ if } \left[h_t(x_i) \neq y_i\right] \text{ (classified incorrectly)} \\ 0 \qquad\qquad\qquad \textit{otherwise} \end{cases}$$

Check whether $\varepsilon_t < 0.5$, otherwise stop.

Step2 : $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$, α_t = weight of classifier (confidence).

*Step*3 : $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, see later slide for explanation

Main Loop (Step 4)

Step4 : Current total cascaded classifier error $CE_t = \sum_{j=1}^{j=t} E_j\left(t, \alpha_\tau, h_\tau(x_i)\right)$

where the current classifier error $E_\tau = \frac{1}{n}\sum_{\tau=1}^{n} I\left(t, \alpha_\tau, h_\tau(x_i)\right)$,

and $I(\,)$ denotes correctness for x_i of the current cascaded classifier :

$$y_i = sign\left(\sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)\right) \quad \rightarrow I\left(t, \alpha_\tau, h_\tau(x_i)\right) = 0$$

$$y_i \neq sign\left(\sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)\right) \quad \rightarrow I\left(t, \alpha_\tau, h_\tau(x_i)\right) = 1$$

If $CE_t = 0$ then T = t, break;

}

$$\boxed{\text{The final strong classifier } H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x) - 0\right)}$$

Note: Normalization Factor Z_t in Step3

AdaBoost chooses this weight update function deliberately

$$D_{t+1}(i) \propto D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

because:

- sample correctly classified *(sign(h)=sign(y))* → weight decreases
- sample incorrectly classified *(sign(h)≠sign(y))* → weight increases

$\mathrm{Re}\,call:$

$$Step3: \quad D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where $Z_t = normalization$ factor

$$Z_t = \sum_{i=1}^{correctly\_classified} D_t \quad (i)e^{-\alpha_t y_i h_t(x_i)} + \sum_{i=1}^{incorrectly\_classified} D_t \quad (i)e^{\alpha_t y_i h_t(x_i)}$$

so D_t becomes a *probability distribution*

Loss Function View

- AdaBoost optimizes the exponential loss:

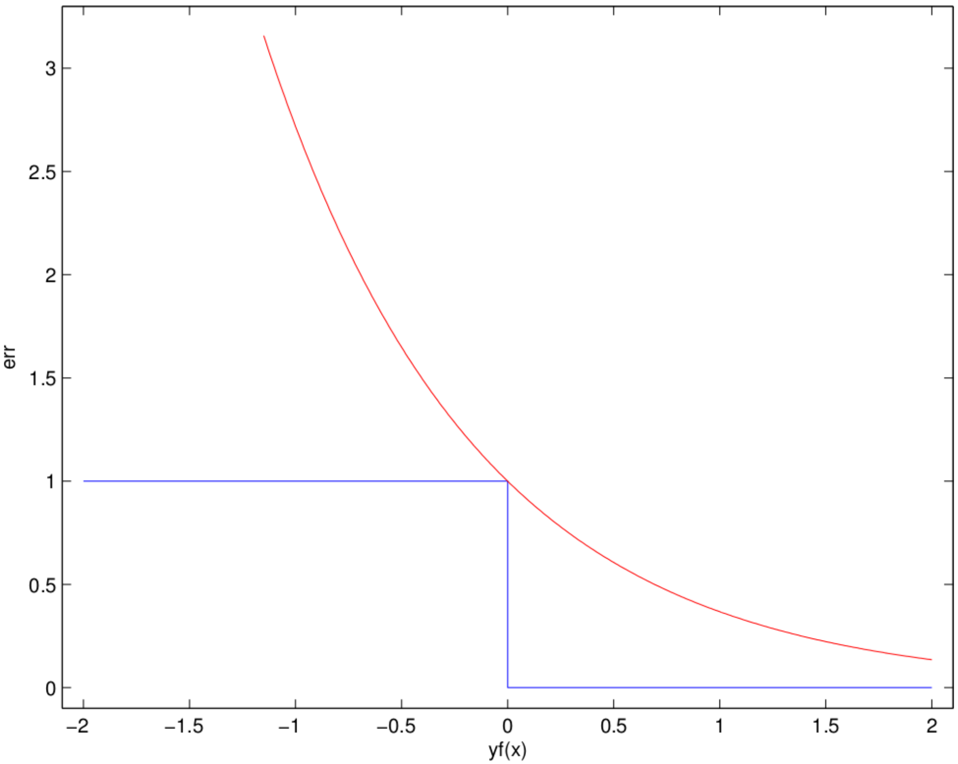
$$L_{\exp}(x, y) = e^{-y\, h(x)}$$

- Full objective function:

$$E = \sum_i e^{-1/2\, y_i \sum_t \alpha_t h_t(x_i)}$$

- Upper bound on error:

$$L_{\exp}(x, y) \geq L_{0\text{-}1}(x, y)$$



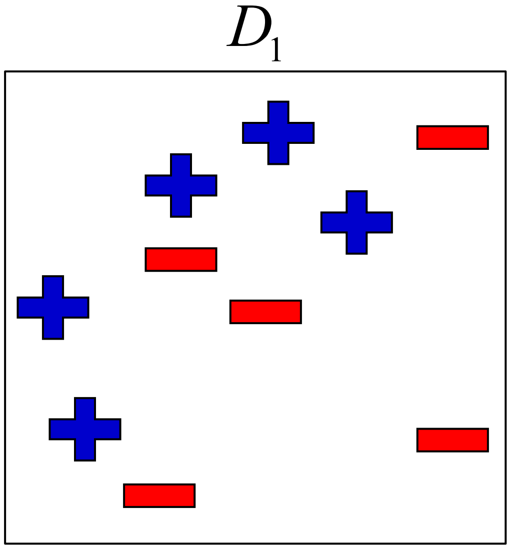
Loss Function View (2)

- Loss function discovered long after the algorithm
- Loss function explains the formula for setting the classifier weights α_t (Step2)
- Gradient descent on exponential loss function would not be recommendable

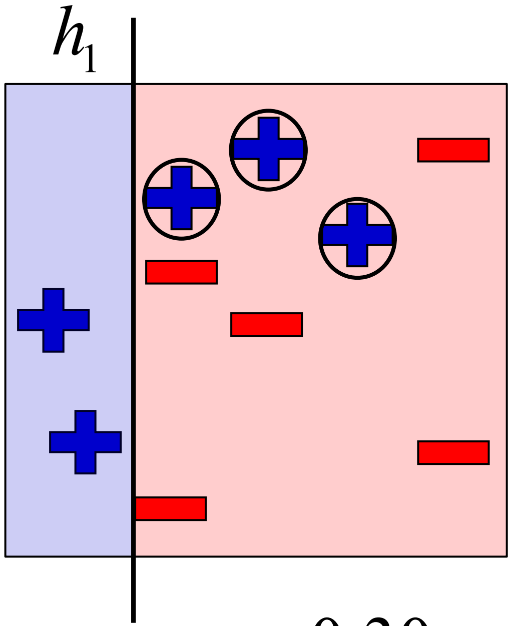
Further reading: Aaron Hertzmann and David Fleet: CSC 411 / CSC D11 Chapter 18 Adaboost

AdaBoost: Toy Example

- Weak classifiers = vertical or horizontal half-planes:

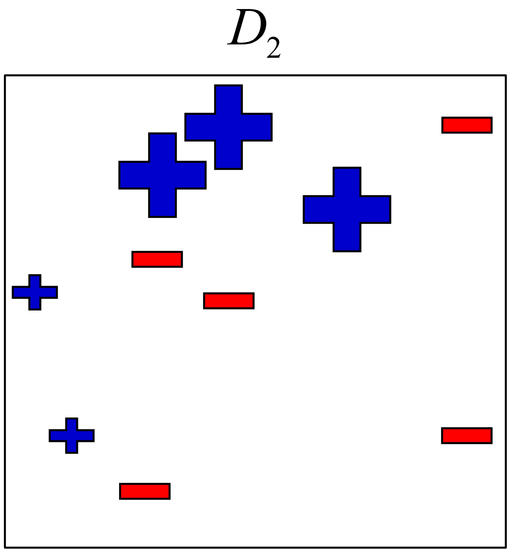


Round One:

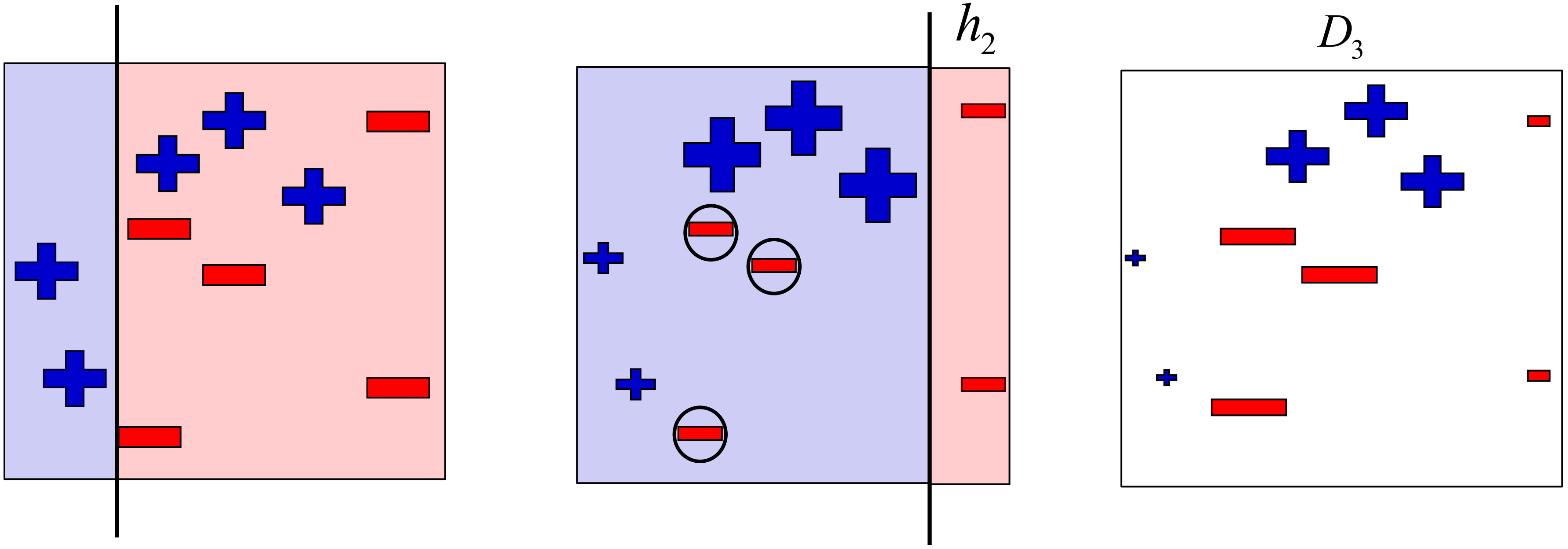


$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$



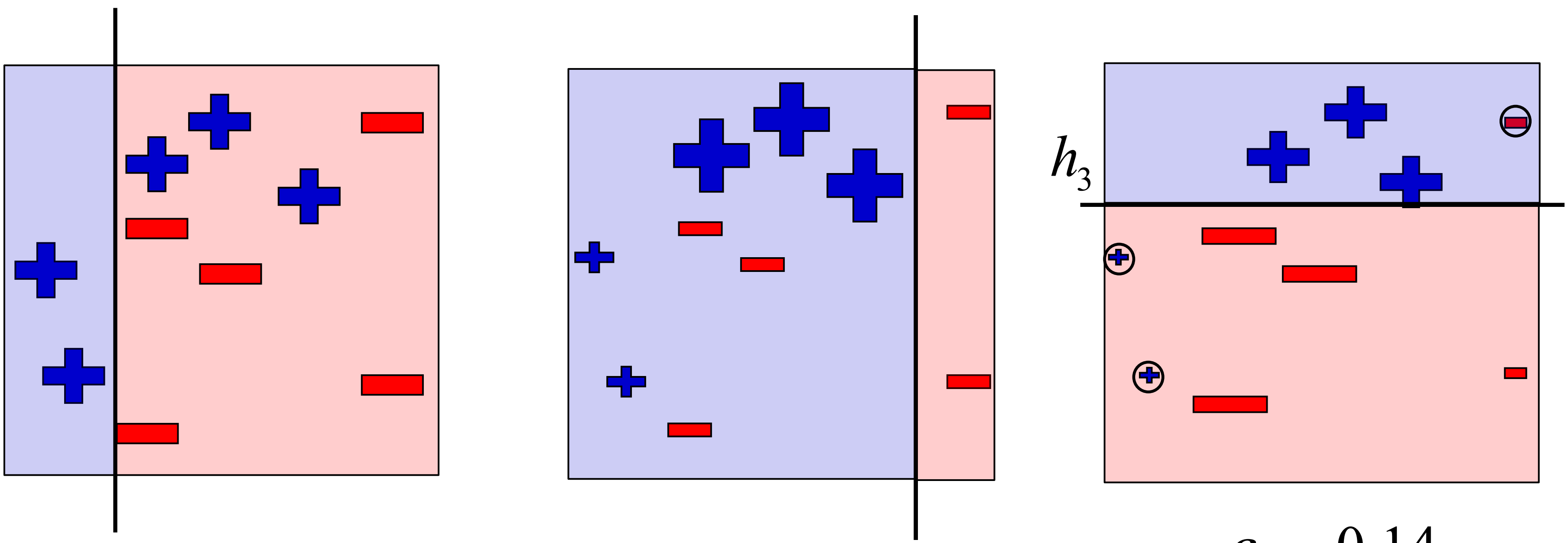
Round Two:



$$\varepsilon_2 = 0.21$$

$$\alpha_2 = 0.65$$

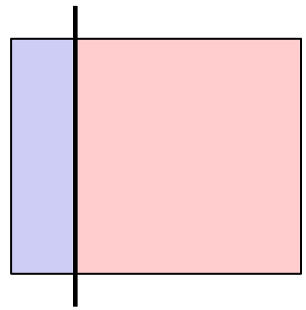
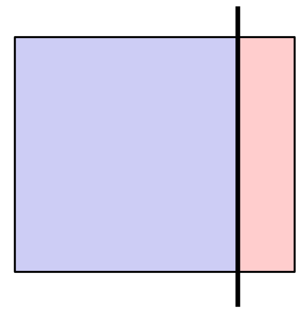
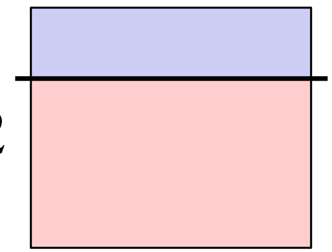
Round Three:

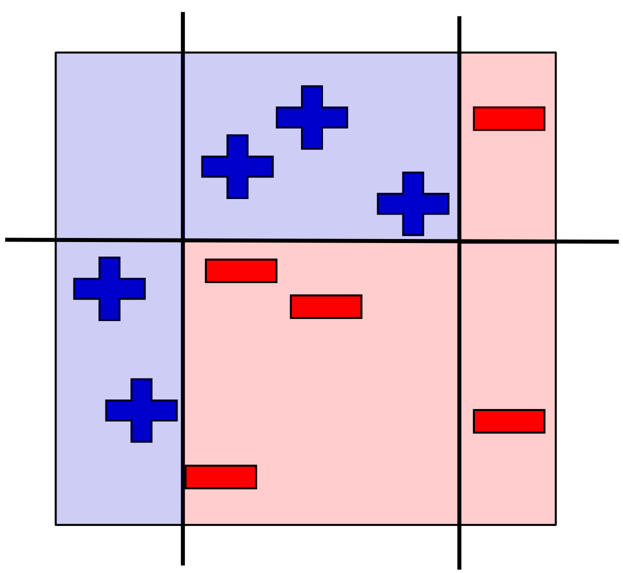


$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Final Classifier:

$H_{final} = sign\Big(0.42$  $+0.65$  $+0.92$  $\Big)$

$=$ 

Based on these principles of ***AdaBoost Algorithm***, many variants exist depending on:

- how to set the weights and
- how to combine the hypotheses

AdaBoost is quite popular!

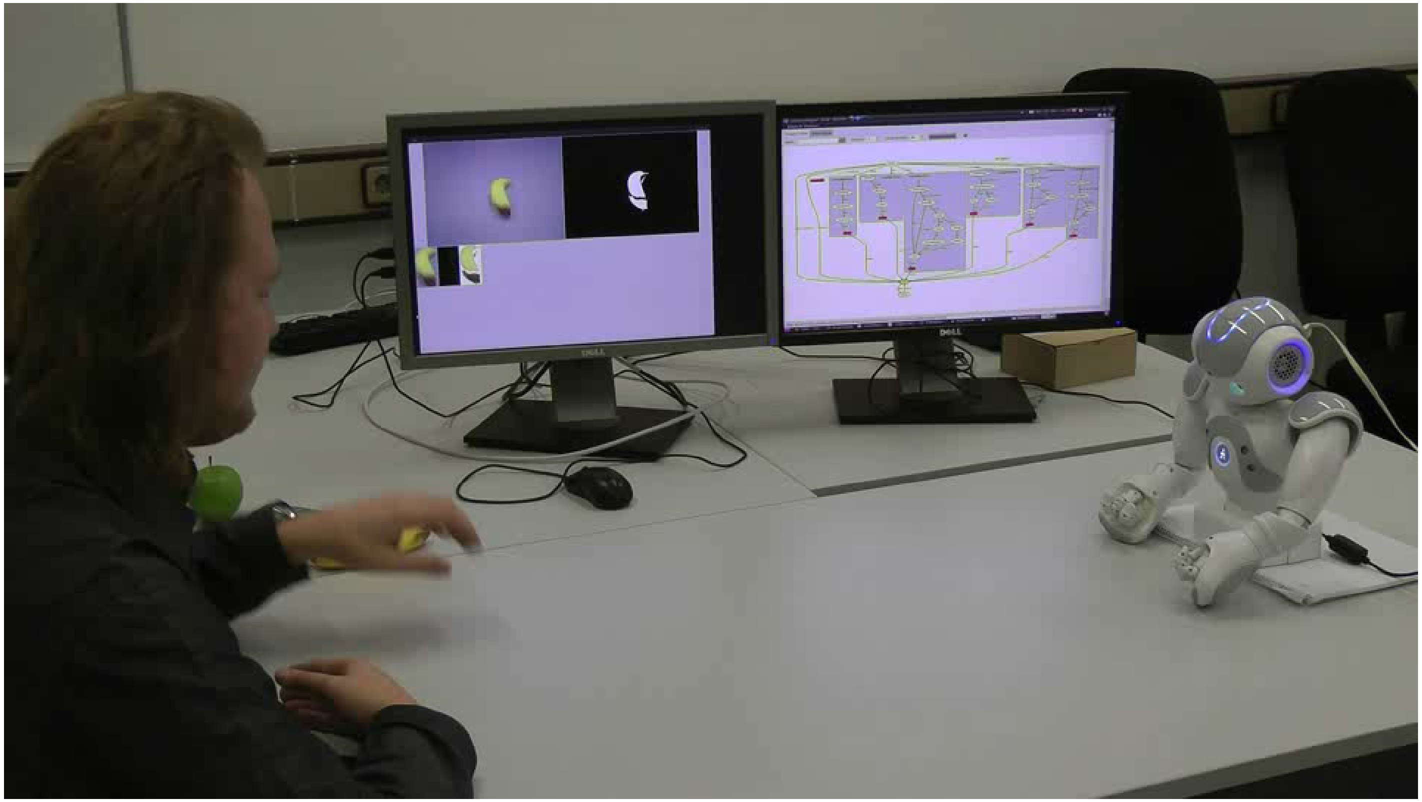
Boosting Summary (1)

- Originally developed by computational learning theorists – [Schapire, 1990] (weak learner).
- Revised to become a practical algorithm, ***AdaBoost***, for building ensembles that empirically improves generalization performance [Freund & Shapire, 1996]
- ***AdaBoost*** - key insights:
  - Instead of sampling (as in bagging) re-weigh examples!
  - Final classification based on weighted vote of weak classifiers
  - Needs smaller number of training samples than bagging

Boosting Summary (2)

- Advantages of boosting
  - Flexibility in the choice of weak learners
  - Testing is fast
  - Easy to implement
  - Integrates classification with feature selection
  - Complexity of training may be linear instead of quadratic in the number of training samples
- Disadvantages
  - Often doesn’t work for many-class problems
  - Minimizes classification error but not, e.g., false negatives
  - Can overfit in the presence of noise

Hybrid Ensemble Learning with the NAO

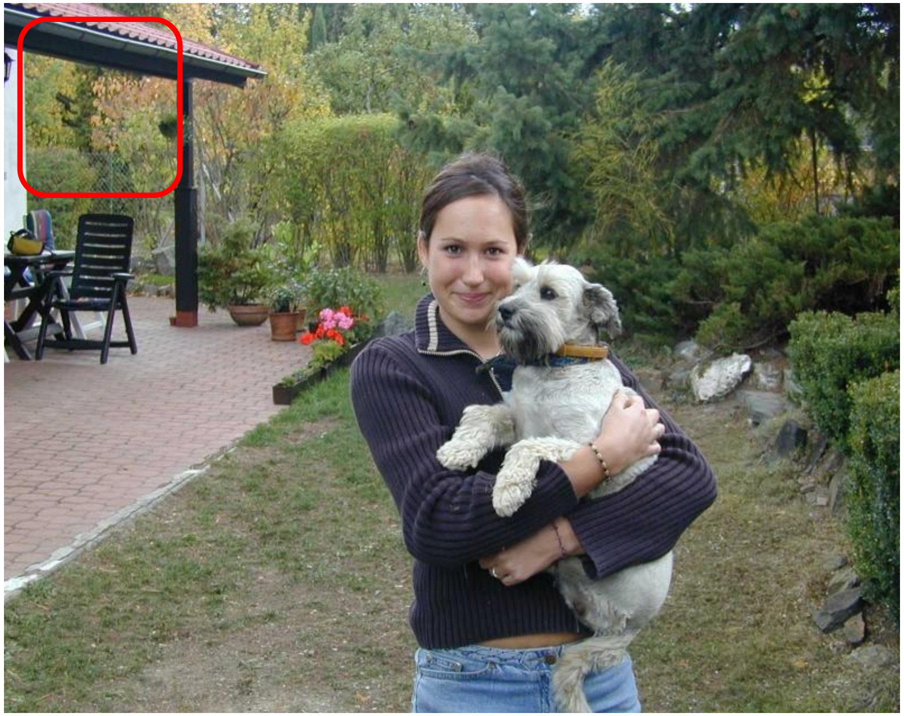


NAO learns objects based on an ***ensemble*** of ***neural networks***

- Every network classifies based on ***different features***:
  pixel patterns, color & texture, or SURF features

http://www.informatik.uni-hamburg.de/WTM/teaching/WiSe11_HumanRobotInteraction_Pj.shtml

Boosting for Face Detection (1)



- *Basic idea:* slide a window across image and evaluate a face model at every location

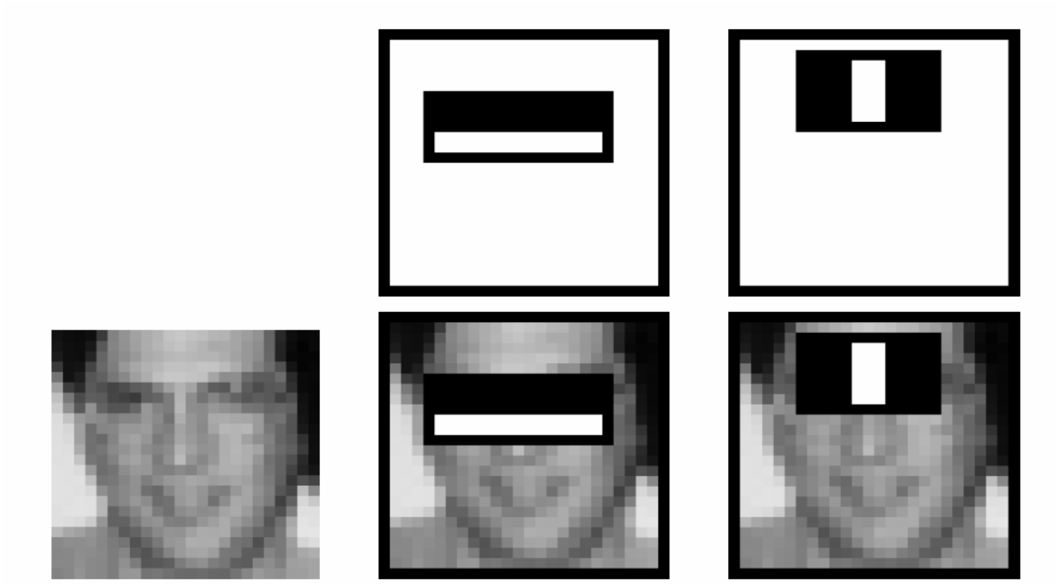
Boosting for Face Detection (2)

- Define ***weak learners*** based on rectangle features
- For each round of boosting:
  - Evaluate each rectangle filter on each sample
  - Select best filter/threshold combination
  - Reweight samples

- Computational ***complexity*** of learning: O(MNK)
  - M rounds, N samples, K features

Boosting for Face Detection (3)

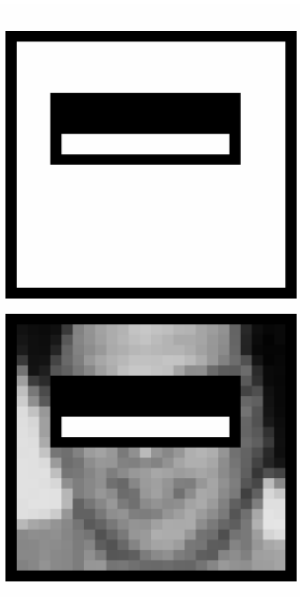
- First two features selected by boosting:

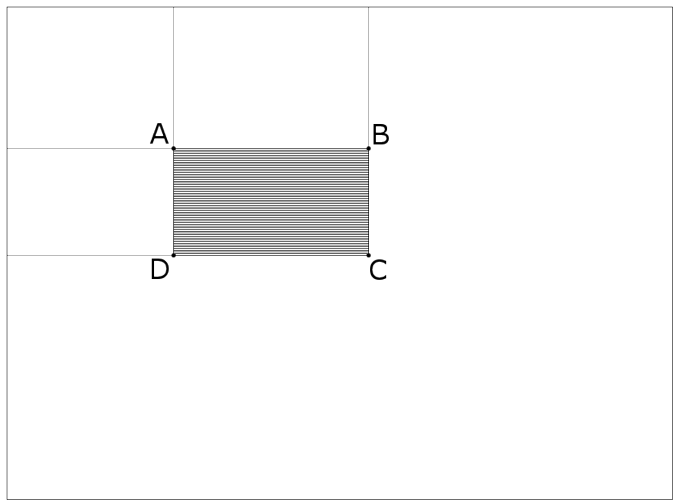


- This feature combination can yield 100% detection rate, however, while also finding many of false positives

Boosting for Face Detection (4)

- Efficient computation of rectangle sums via integral image:





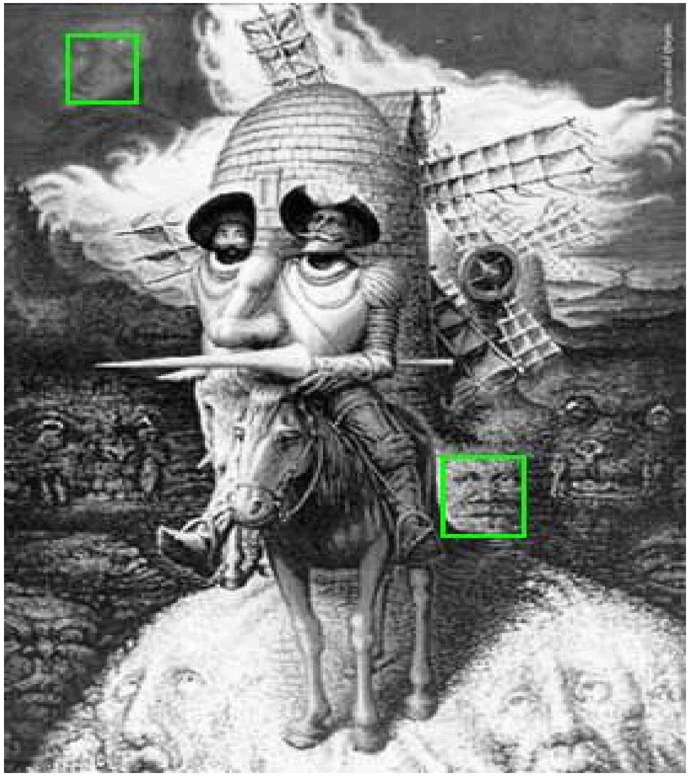
$$I(x,y) = \sum_{\substack{x' < x \\ y' < y}} i(x', y')$$

rectangle sum: $I(A) + I(C) - I(B) - I(D)$

Boosting for Face Detection (5)

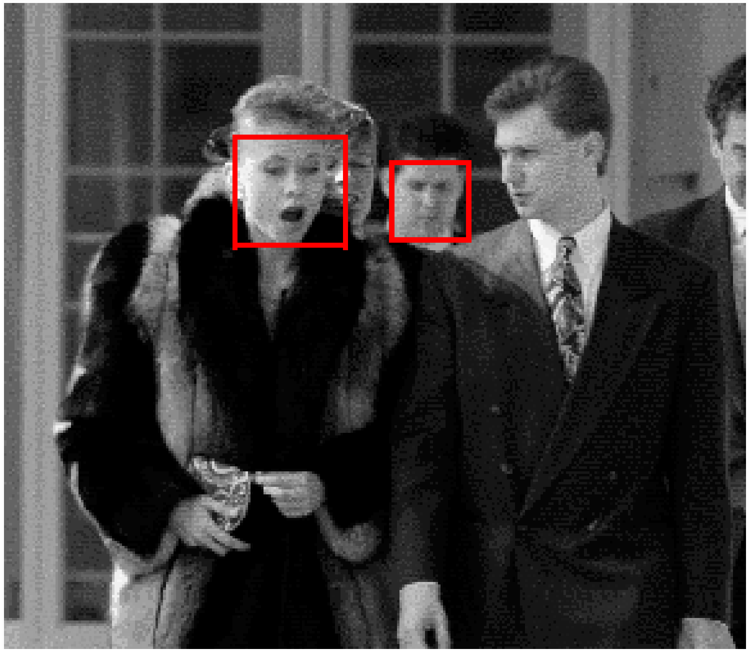
Boosting for Face Detection (5)

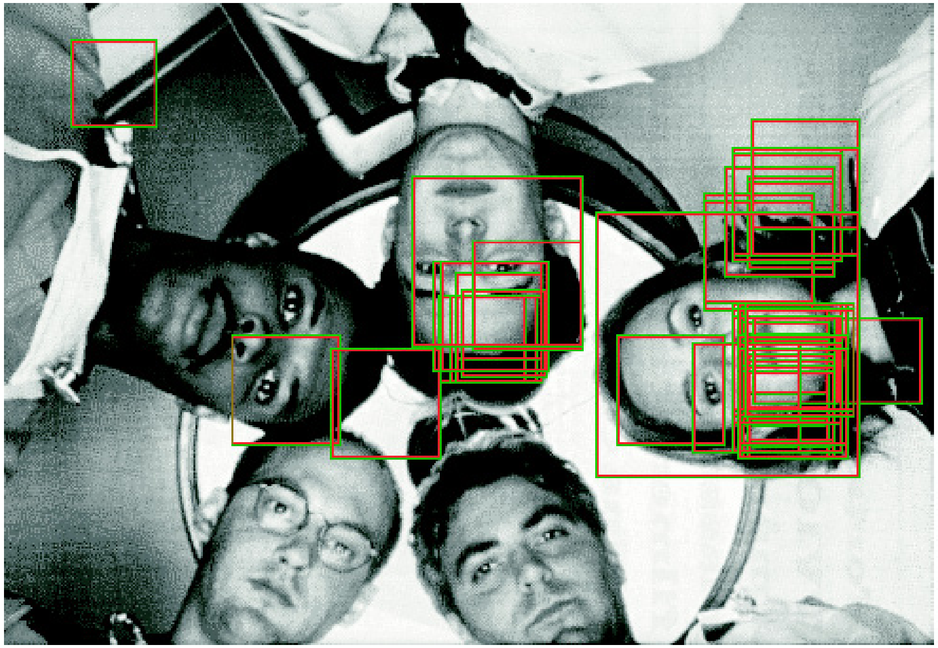
Boosting for Face Detection (5)



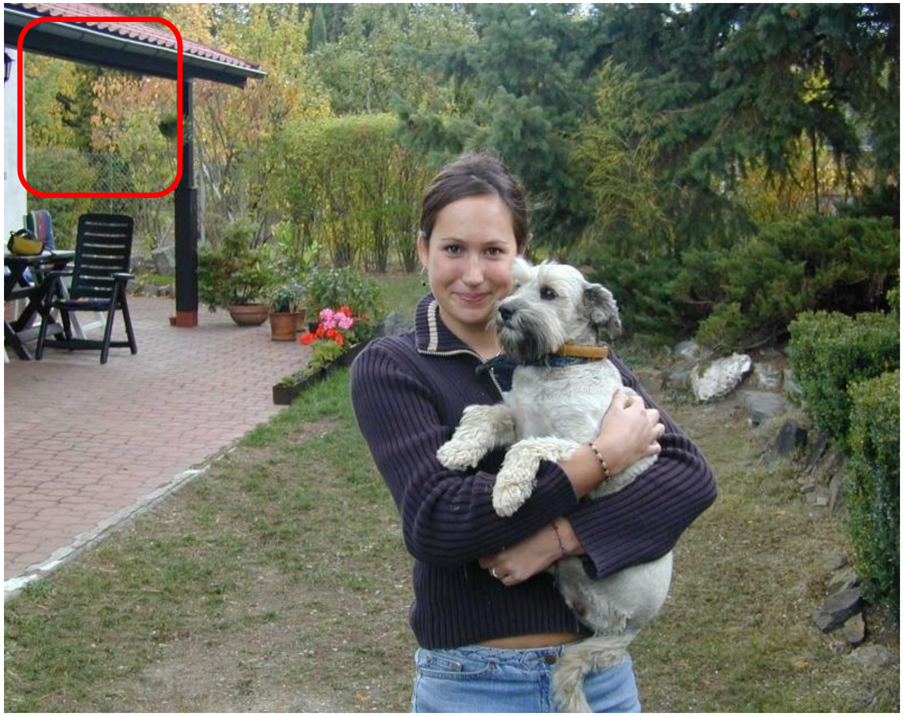
Boosting for Face Detection (6)

- Scale- and shift invariance are built-in
- Limitations with occlusion and rotations





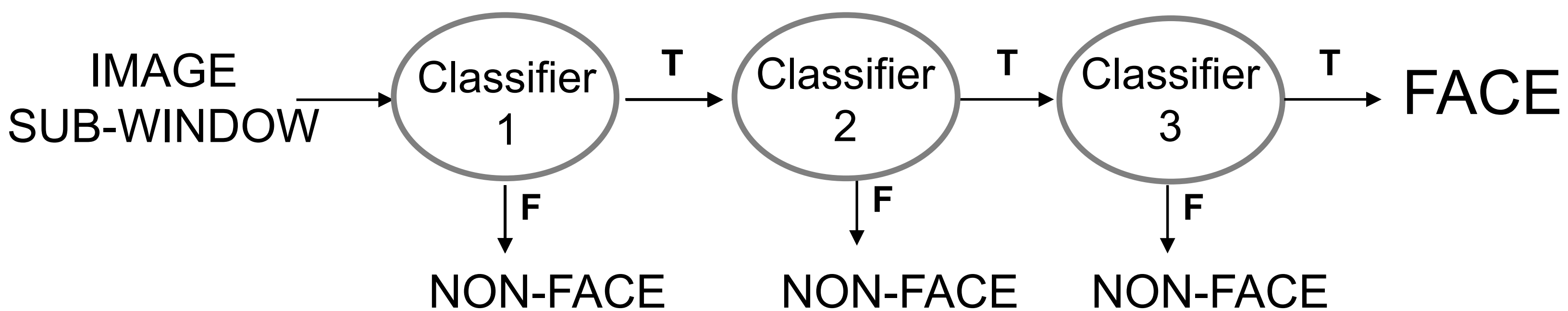
… Boosting for Face Detection …



- *Inefficient:* detailed analysis of large image regions

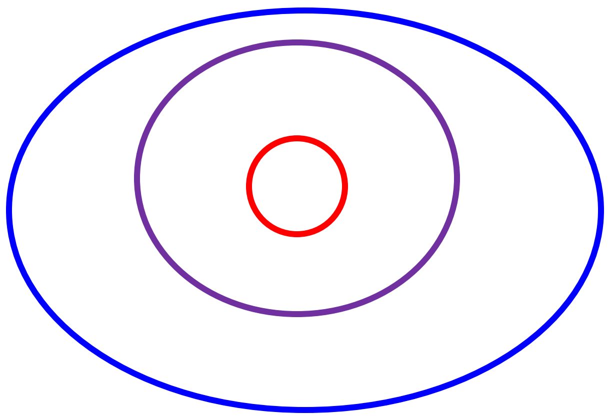
Attentional Cascades

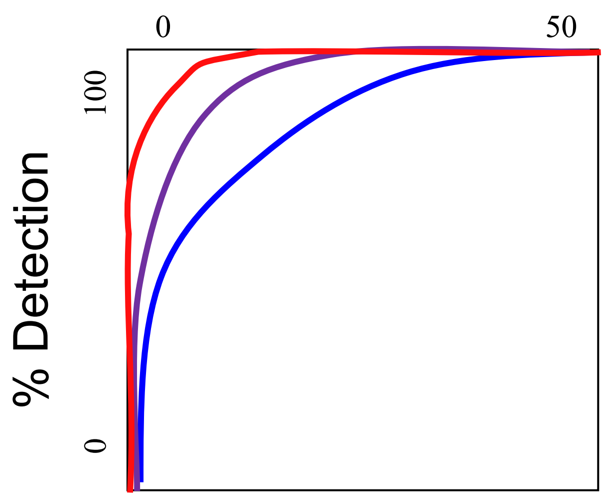
- Start with simple classifiers which reject many of the negative sub-windows while detecting (almost) all positive sub-windows
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on…
- A negative outcome at any point leads to the immediate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



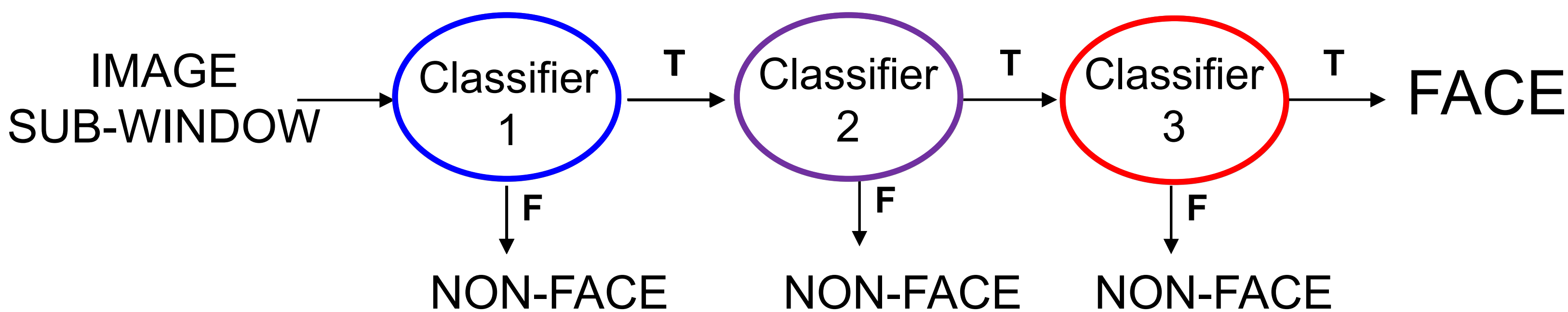
Attentional Cascades (2)

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





Receiver operating characteristic



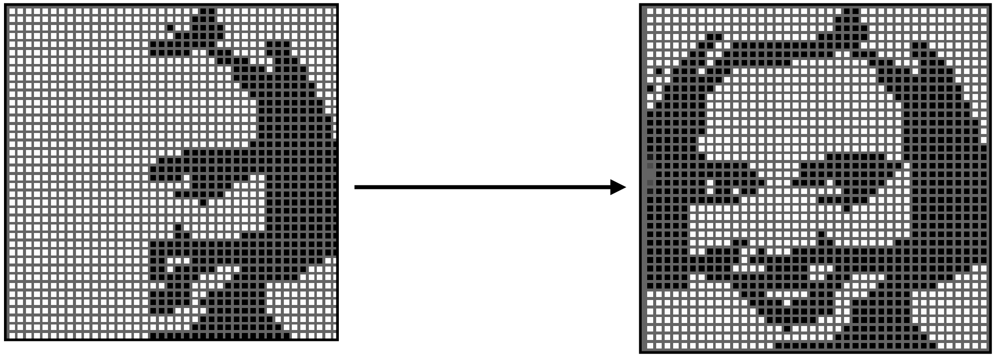
Hopfield Neural Networks and Boosting for Face Detection

- Hopfield Neural Networks application: real-time face detection for autonomous robots
  - Networks classify faces based on a set of features
  - Hopfield networks can reconstruct a learned pattern from ***noisy input***

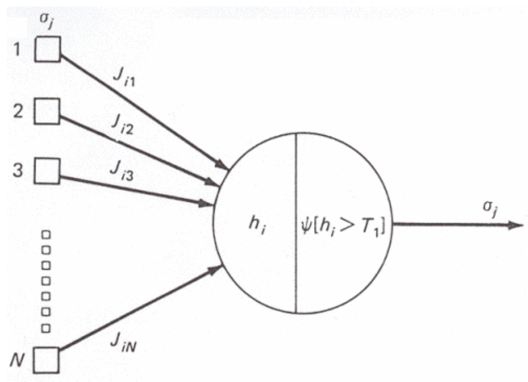
Meins, Wermter , Weber 2012

Hopfield Neural Networks and Boosting for Face Detection

- Reproduce the example in the course where parts of a pattern can be used to recover the whole pattern



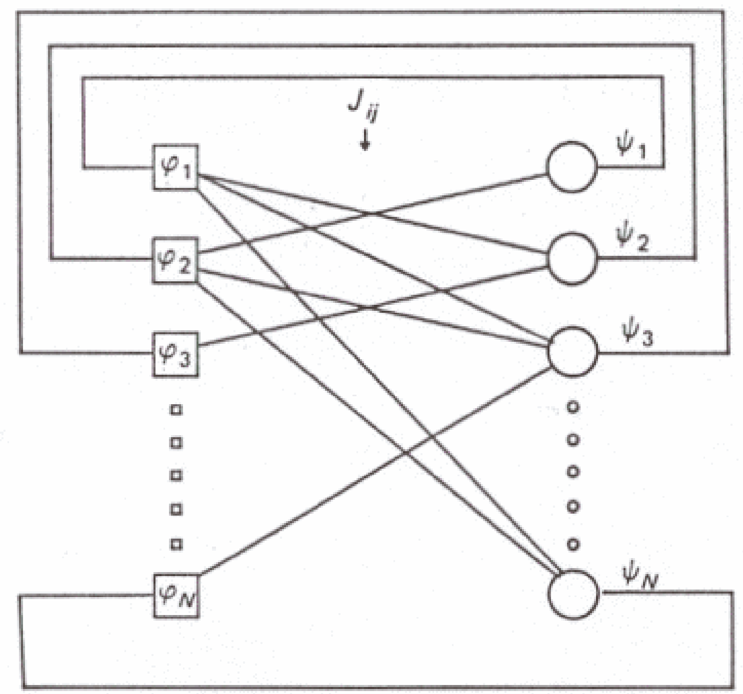
Hopfield Network



- Single neuron:

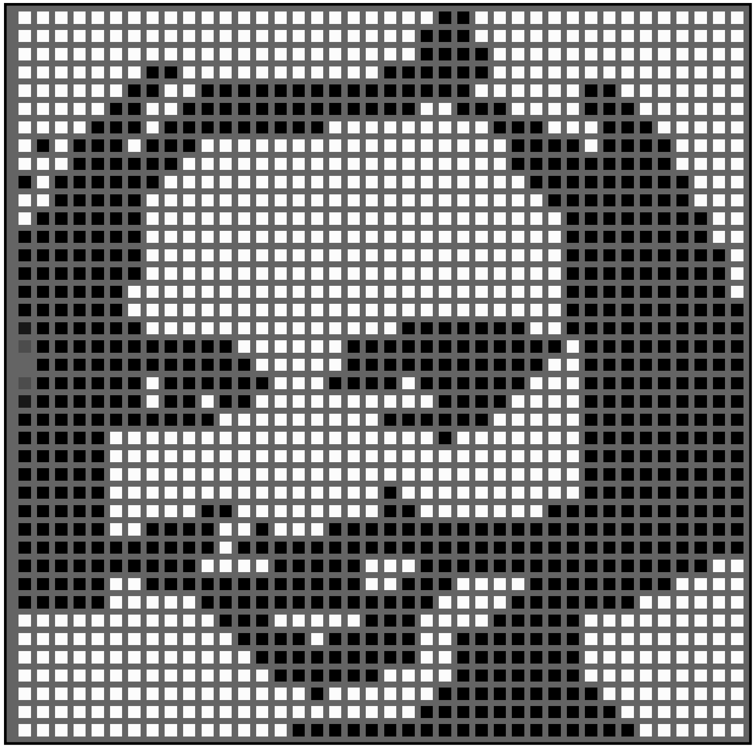
$$h_i = \sum_{j=1}^{N} J_{ij}\sigma_j$$

$$\sigma' = \psi[h_i > T_i]$$



- The output of binary neurons feeds back into their input
  → dynamical system

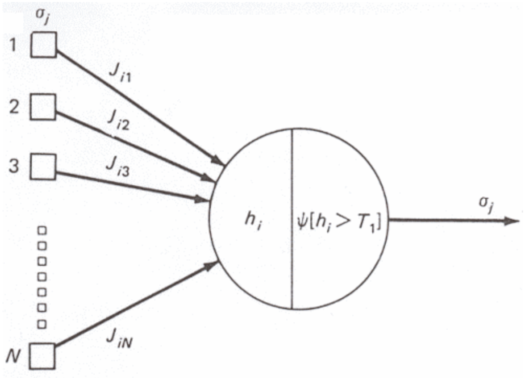
Hopfield Network



- Hebb rule:

$$J_{ij} = \sum_{n}^{D} \sigma_i^n \sigma_j^n$$

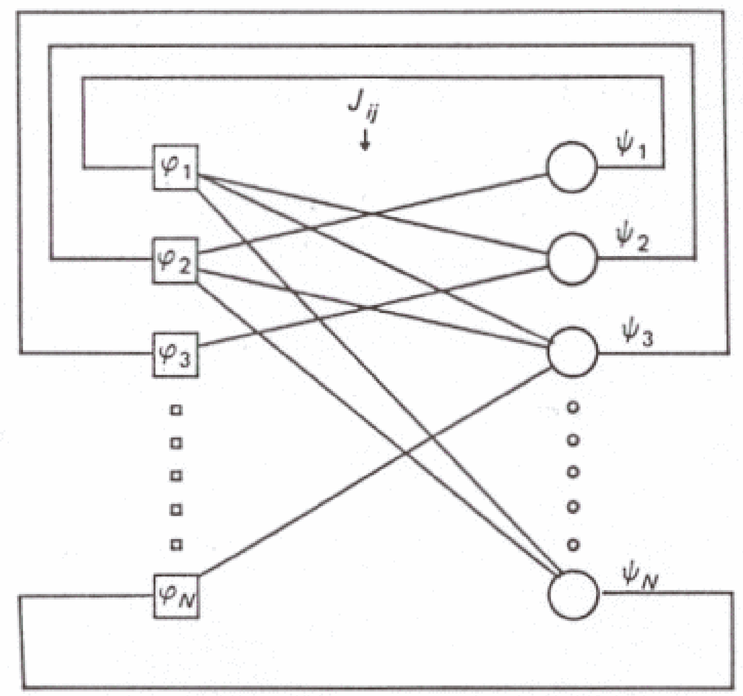
Hopfield Network



- Single neuron:

$$h_i = \sum_{j=1}^{N} J_{ij}\sigma_j$$

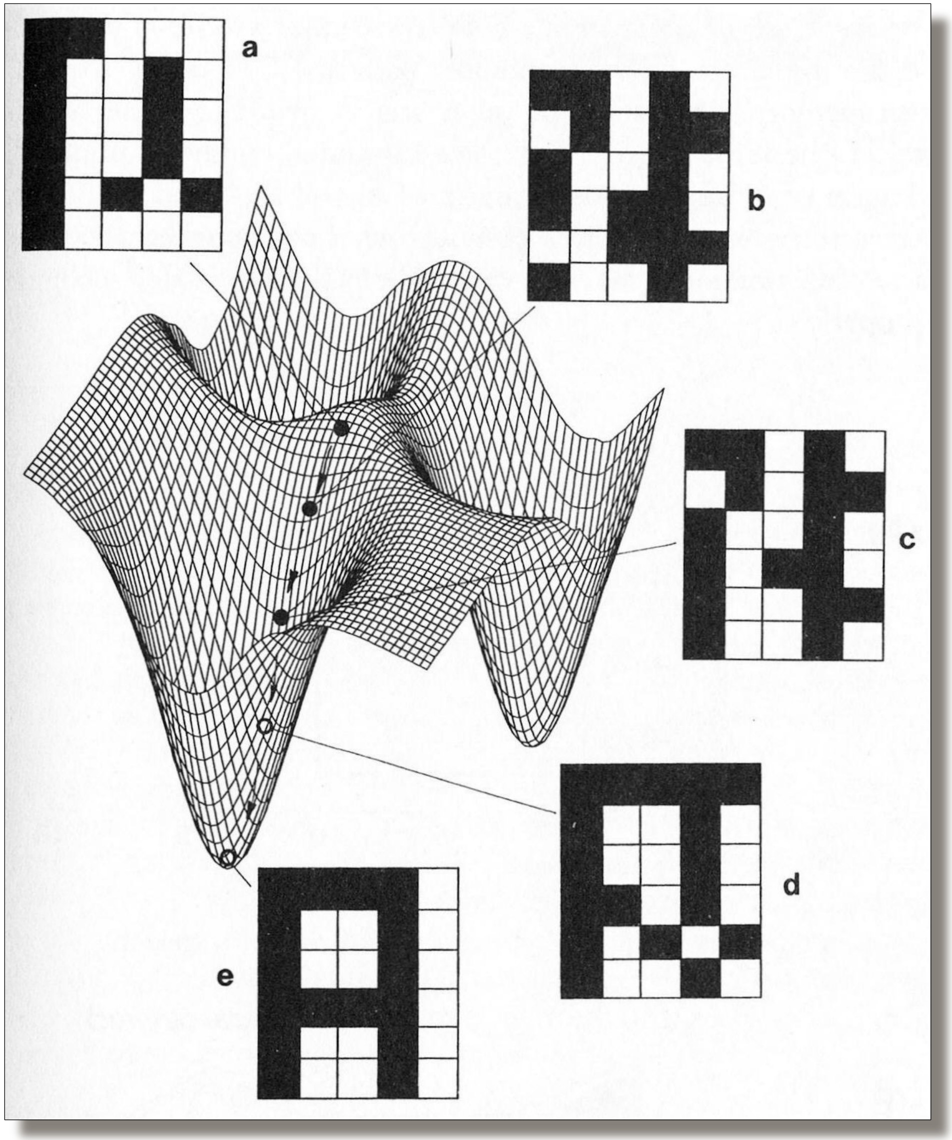
$$\sigma' = \psi[h_i > T_i]$$



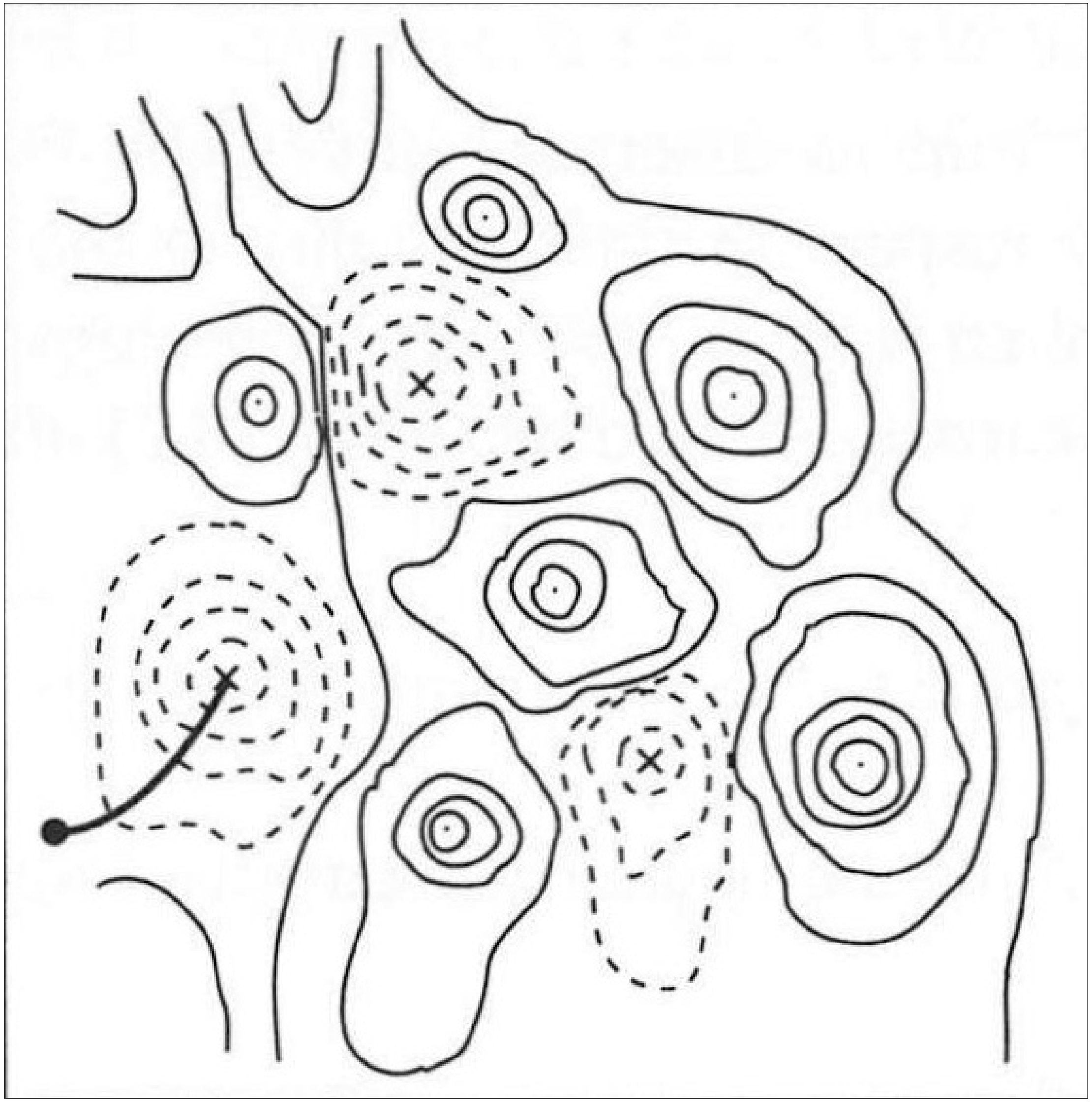
- The output of binary neurons feeds back into their input
  $\rightarrow$ dynamical system
- Energy function:

$$\boxed{E(\sigma) = -\frac{1}{2}\sum_{i>j} J_{ij}\sigma_i\sigma_j}$$

Descent on Energy Surface

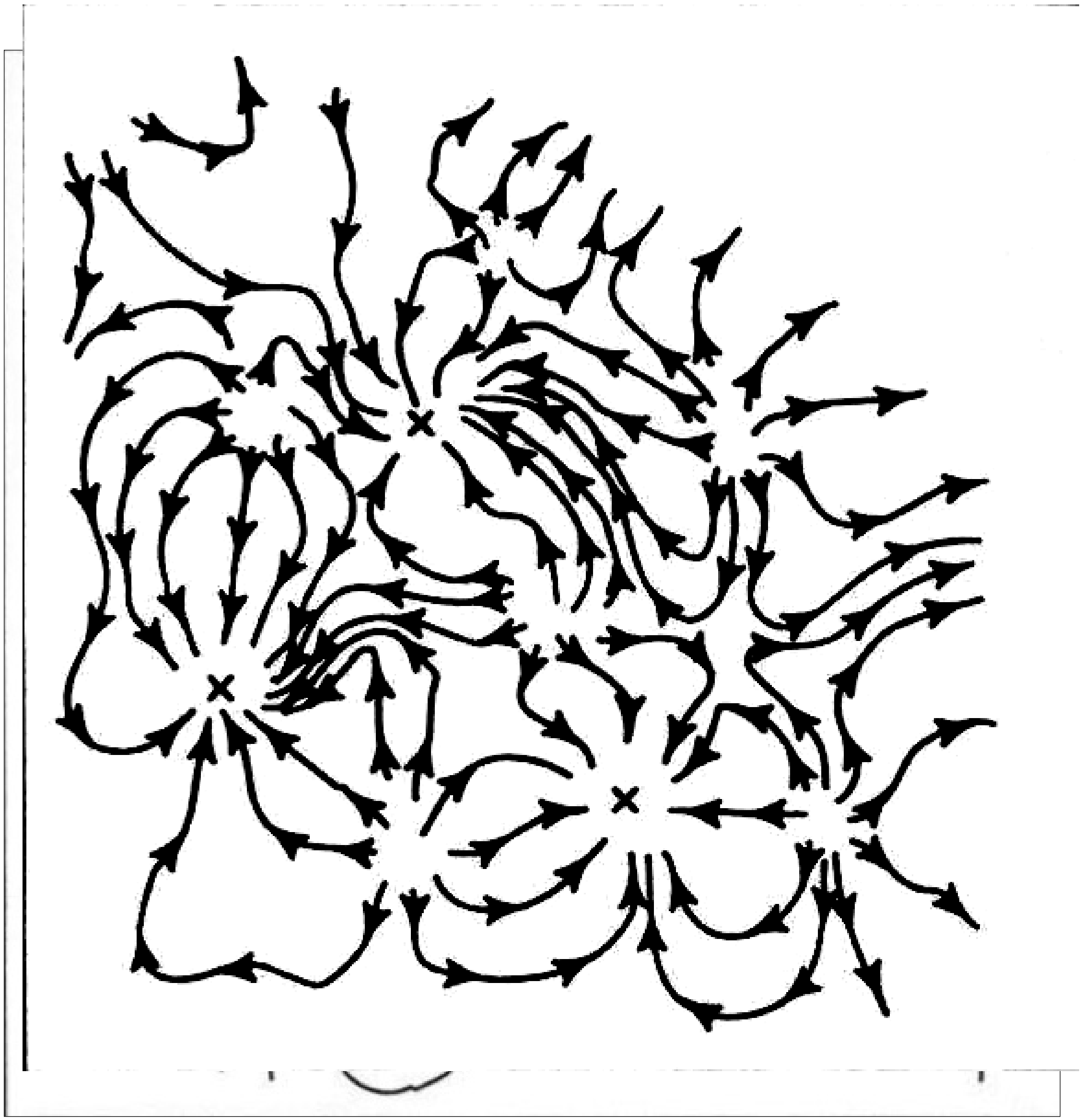


(fig. from Solé & Goodwin)



Energy Surface

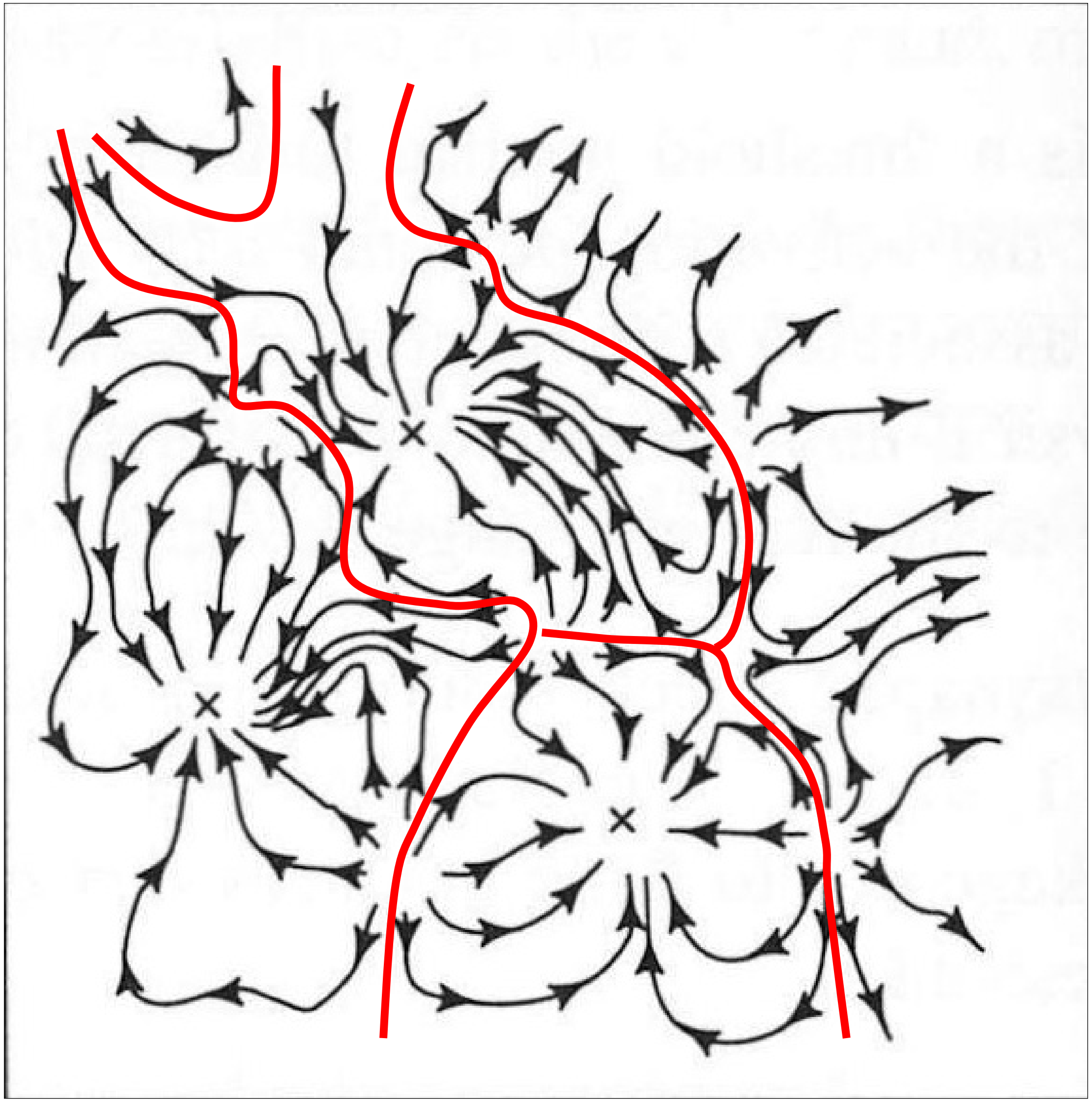
(fig. from Haykin *Neur. Netw.*)



Energy Surface + Flow Lines

(fig. from Haykin *Neur. Netw.*)

Flow Lines

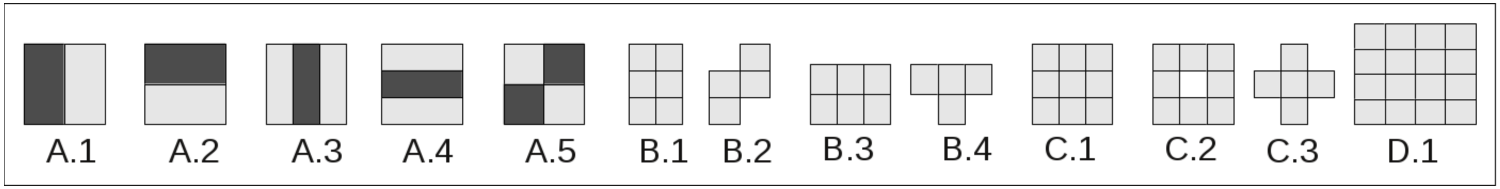


Basins of Attraction

(fig. from Haykin *Neur. Netw.*)

Hopfield Neural Networks and Boosting for Face Detection

- Recall: Haar-like ***features***: Small sets of adjacent pixels
  - Efficient method for interesting aspects in images
  - Can be computed very fast



Meins, Wermter , Weber 2012

Ensembles and AdaBoost

- Ensembles combine classifiers to improve the accuracy
  - Act as one strong classifier
  - Simple ensemble example: equal voting over all members
  - In the AdaBoost context: ensemble-members are mostly ***weak classifiers***

- `AdaBoost`: Algorithm to select the classifier with the lowest error on a training set
  - Taking into account the weights from the single images
  - Get different weak classifiers that complement each other
  - The result is a weighted voting over all weak classifiers

Pattern for the Hopfield-Net

- Use the single values of the detected rectangles as the input vector

25	54
217	124

a1	a2
a3	a4

  - Original Haar-feature: $v = a1 + a4 - (a2 + a3)$
  - Hopfield net: use whole vector as input: $v = (a1, a2, a3, a4)$

Use of the Hopfield-Net

1. Pretraining: train network weights on positive examples
2. During ensemble learning, memorize all attractors:
   1. Label all during positive examples
   2. Label all during negative examples
3. During classification, Hopfield network converges to an attractor, and its identity tells whether positive or negative
4. If attractor not known, then regard as negative

Classification

- Apply logistic transfer function

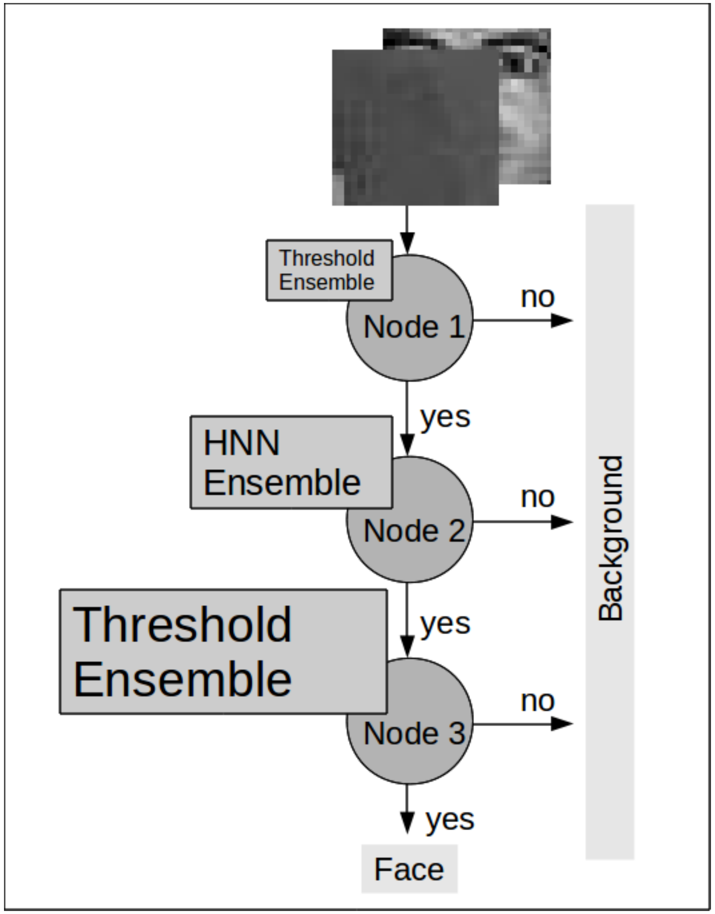
$$s_i = \text{sign}\left(\sum_{j=1}^{n} w_{ij} s_j\right)$$

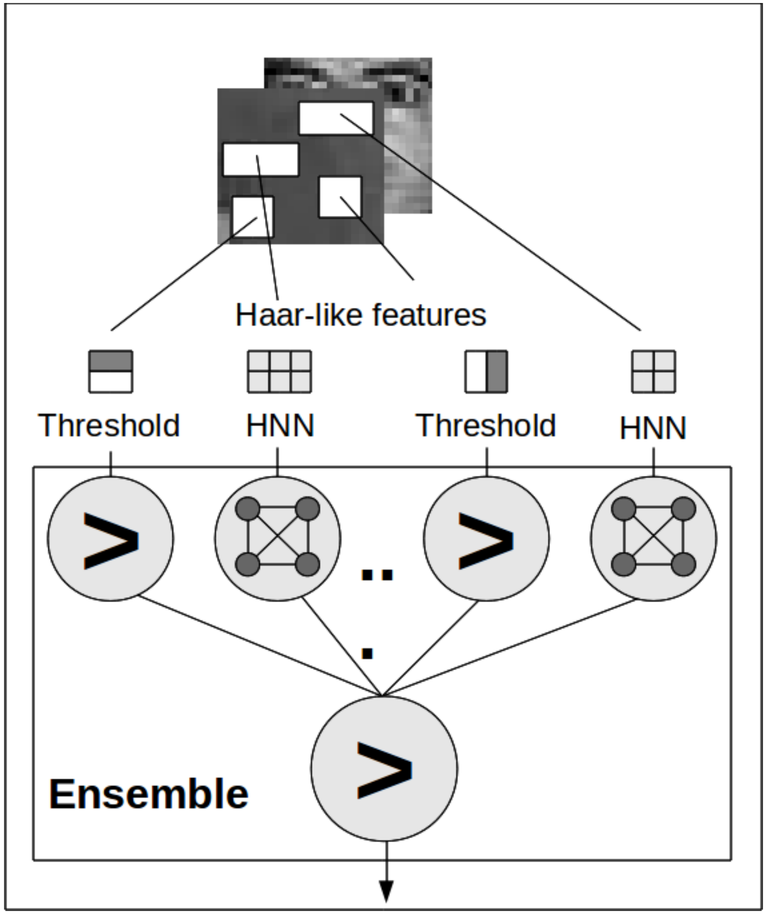
$$s_i = \frac{2\beta}{1 + e^{-u_i}} - \beta \qquad u_i = \sum_{j=1}^{n} w_{ij} s_j$$

where β is the maximal number of learned patterns

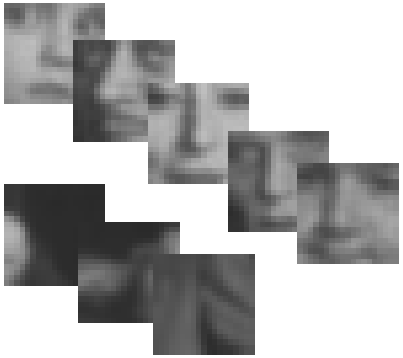
- After the HNN has reached a stable state s, compare state with learned pattern p:
  - If Euclidean distance ***d*** from the stable state pattern ***p*** is less than a threshold θ, then it will be classified as positive

Hybrid Ensembles Detection Framework

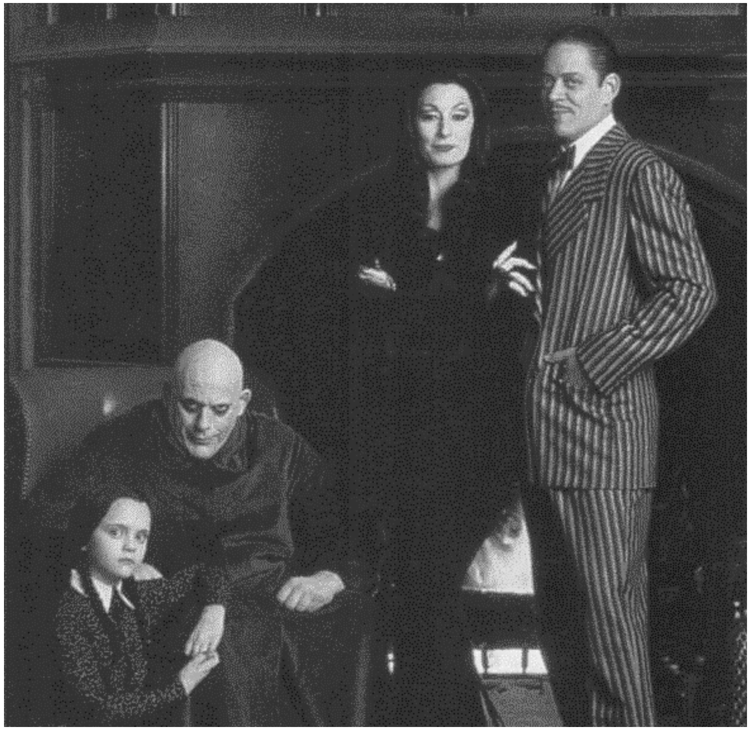




Experimental Analysis



- Train Hybrid Ensembles Detection Framework on a large set of face data:
  - 2429 faces & 4548 non-faces



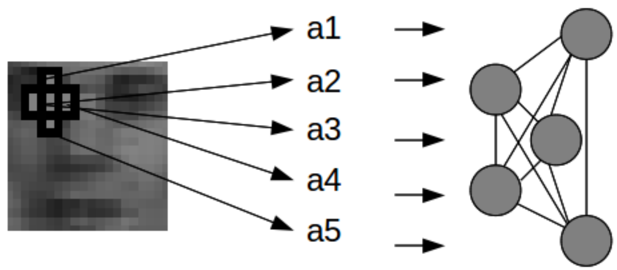
- Test on data sets with various faces in single images

http://cbcl.mit.edu/software-datasets/FaceData2.html

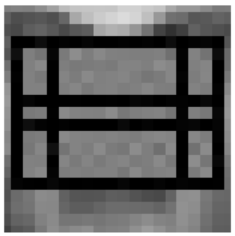
http://vasc.ri.cmu.edu/idb/html/face/frontal_images

Results

- Employed Haar-like features:



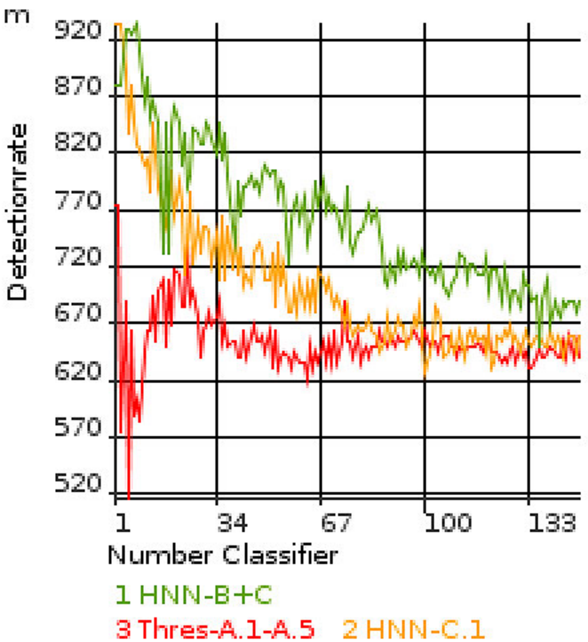
HNN-B+C

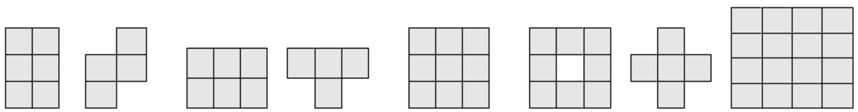


HNN-C.1

Thres-A.1-A.5

- Classification:
  - Hopfield Neural Network Ensembles lead to higher detection rate





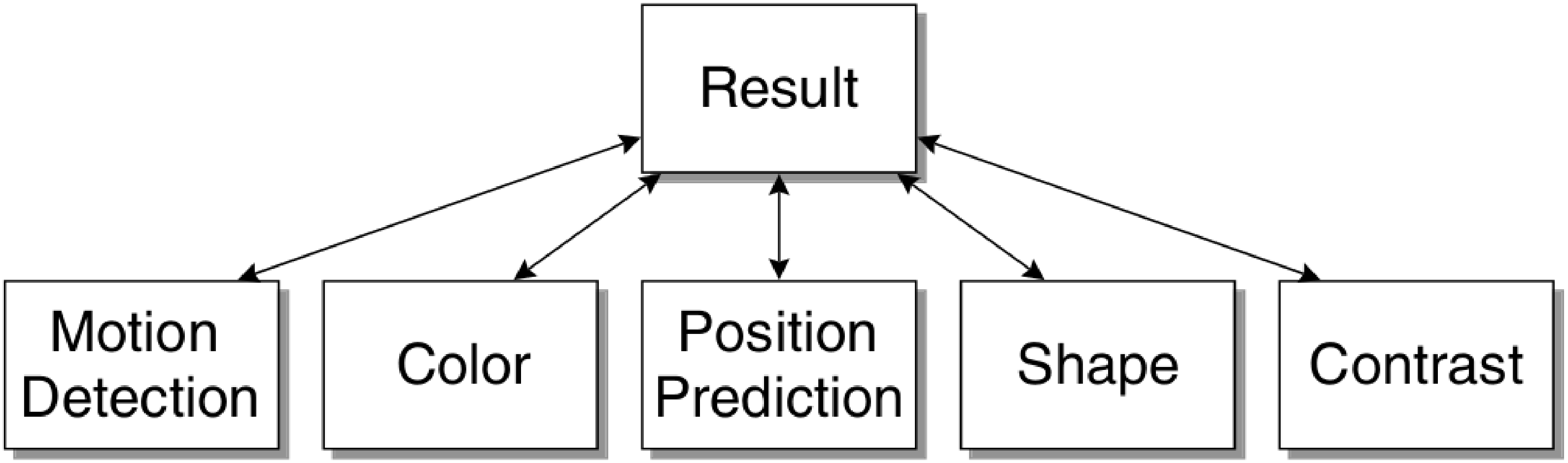
B.1 B.2 B.3 B.4 C.1 C.2 C.3 D.1

Diversity for Ensembles from Data

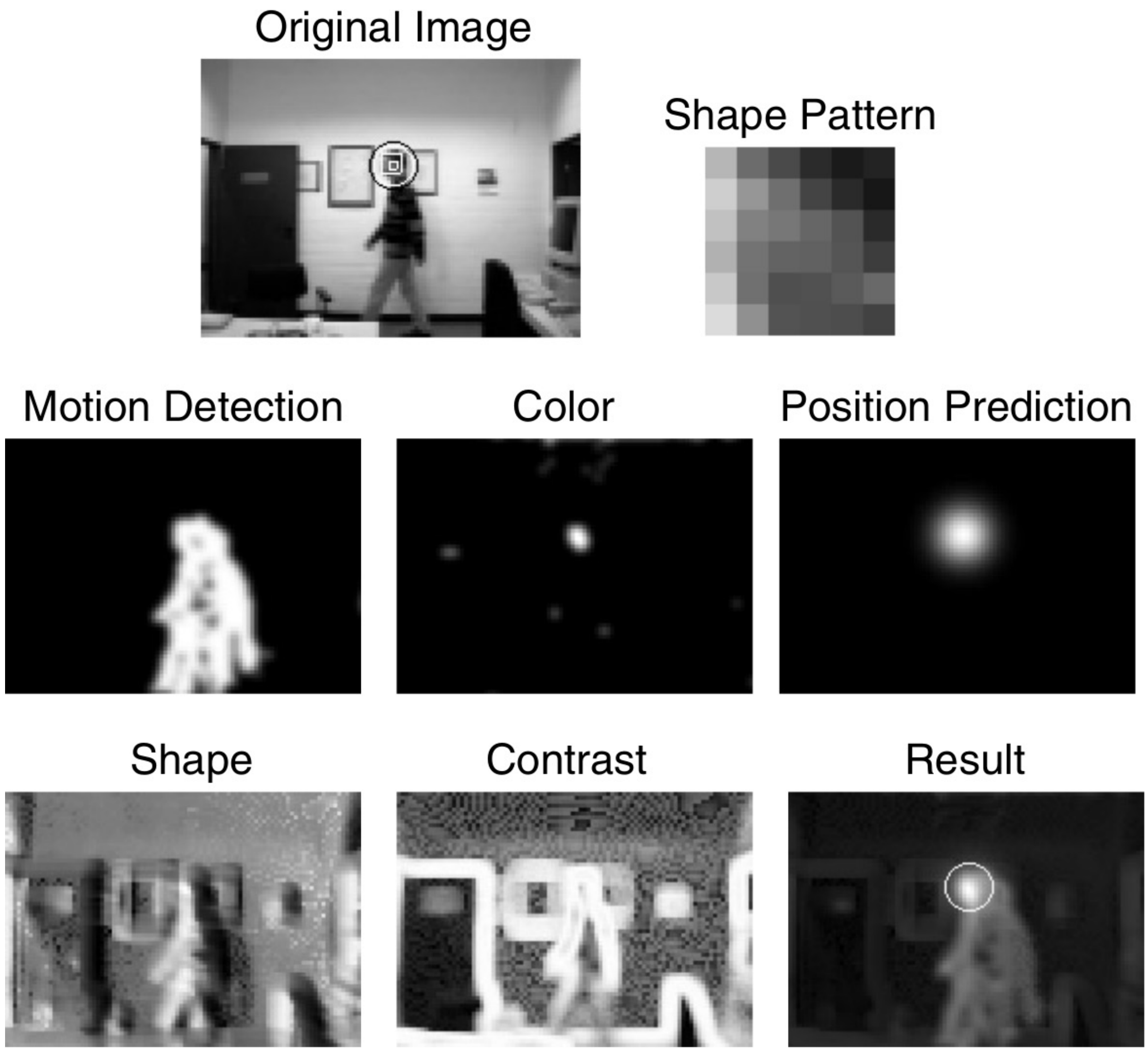
- Visual data has a lot of *diverse features*:
  - Low-level: brightness, contrast, color, motion
  - Medium-level: edges, depth, texture, borders, motion gradient
  - High-level features: prototypical shapes, motion (e.g. looming)
- Features are often redundant, i.e. if one cue fails, others suffice for recognition / classfication
- We can use the majority vote to learn about the additional features

Democratic Integration of Adaptive Cues

- Face detection in video can benefit from additional “cues”:
  - Shape / Contrast
  - Color
  - Motion – background is typically static, but faces not so
  - History – a face’s position does not jump → face tracking
- Any individual cue in isolation is unreliable, but
  an ensemble estimate based on several cues gets reliable



Democratic Integration of Adaptive Cues (2)



Triesch, Malsburg. Democratic Integration: Self-Organized Integration of Adaptive Cues. Neur Comp, 2001

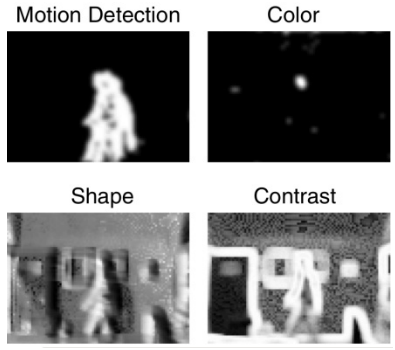
Adaptive Weights and Adaptive Cues (3)

- Cues that prove to be reliable will receive higher weights
- Reliability measured based on the majority vote:
  a cue that predicted the vote of the group well is reliable
- Weights get mistuned when the majority vote is wrong
- Some cues are given, i.e. non-adaptive (e.g. motion)

- Cues` internal parameters adapt to the winning region
- With few assumptions, cues can adapt to track any person, and from then on home-in on the tracked person
- Model robust to natural noise and changes, e.g., switching on a light, pose changes, distractors

Triesch, Malsburg. Democratic Integration: Self-Organized Integration of Adaptive Cues. Neur Comp, 2001

Democratic Cue Integration (4)

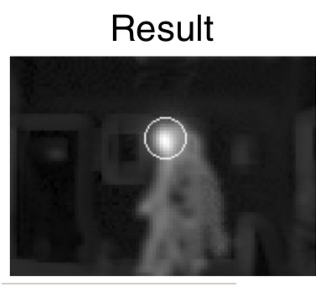


Saliency map of each cue i

$$H_i(x,t) = S_i(P_i, I(x,t))$$

where S_i measures similarity of image region I around x to prototype P_i of the cue

The result is



$$H(x,t) = \sum_i r_i(t) H_i(x,t)$$

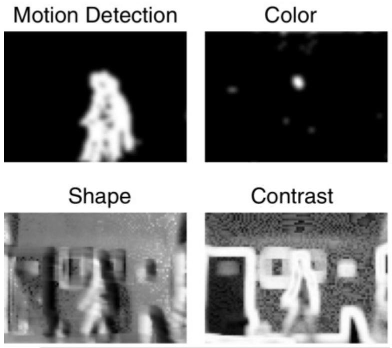
where r_i informs how reliable cue i is.

Final result

$$\hat{x}(t) = \arg\max_x H(x,t)$$

Triesch, Malsburg. Democratic Integration: Self-Organized Integration of Adaptive Cues. Neur Comp, 2001

Democratic Cue Integration (5)



- Quality of a cue

$$q_i(t) \approx R\left(H_i(\hat{x},t) - \frac{1}{\#x}\sum_x H_i(x,t)\right)$$

where R is a ramp function, and $\sum_i q_i = 1$

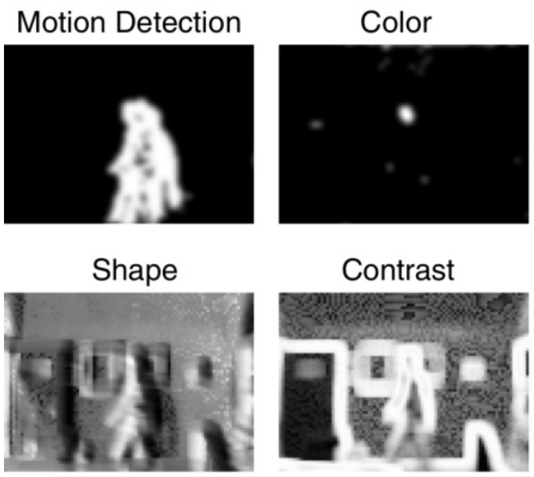
- Reliabilities are a running average of quality

$$\tau \dot{r}_i(t) = q_i(t) - r_i(t)$$

Reliabilities are weights that express how reliable a cue predicted the result in the past

Triesch, Malsburg. Democratic Integration: Self-Organized Integration of Adaptive Cues. Neur Comp, 2001

Democratic Cue Integration (6)



- A cue prototype extracts a feature f_i

$$P_i(x,t) = f_i(I(x,t))$$

- Feature at current target position:

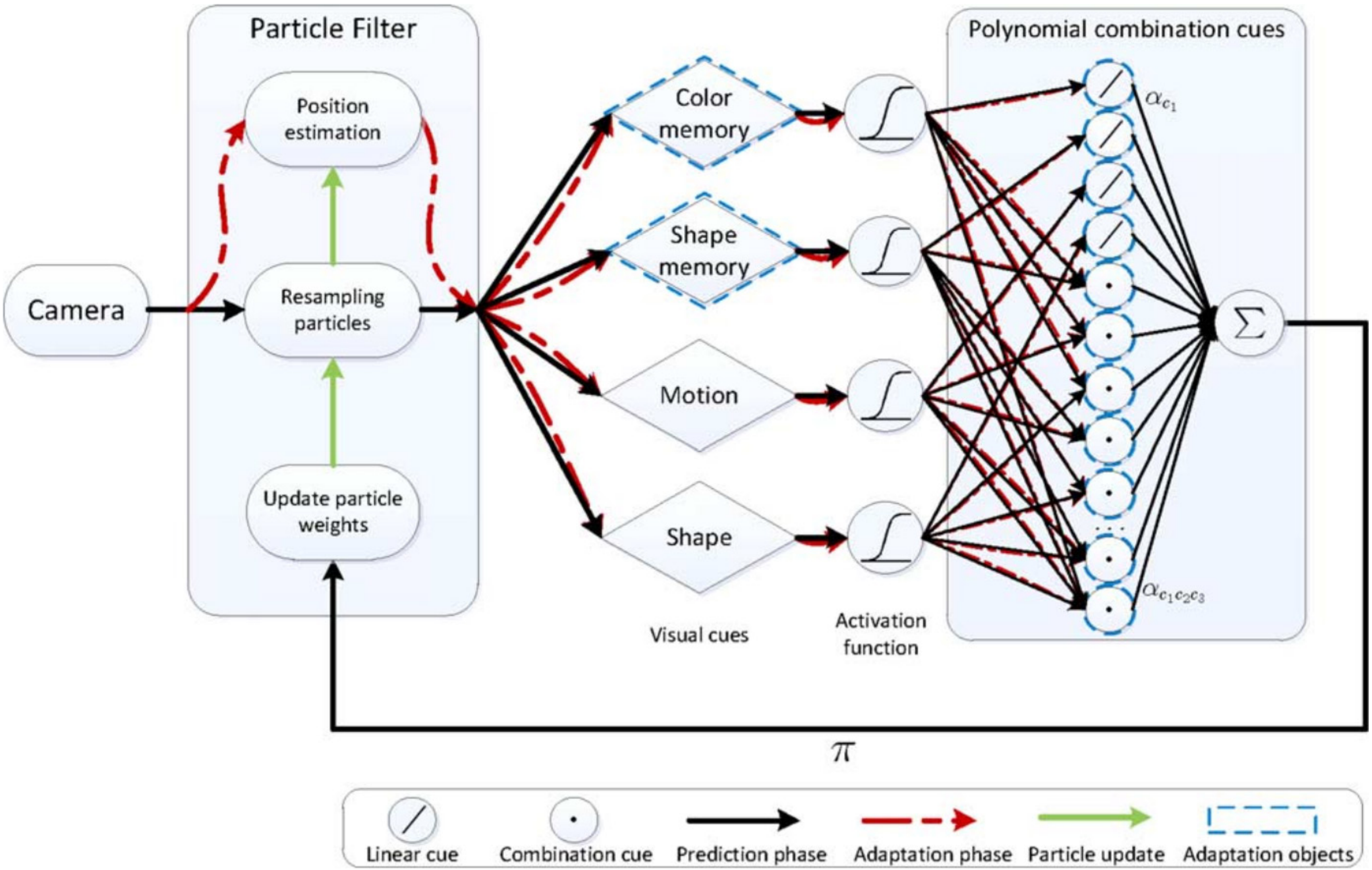
$$\hat{P}_i(x,t) = P_i(\hat{x},t)$$

- A cue's internal parameters adapt so the cue becomes responsive to the winning region

$$\tau \, \dot{\mathrm{P}}_i(t) = \hat{P}_i(t) - P_i(t)$$

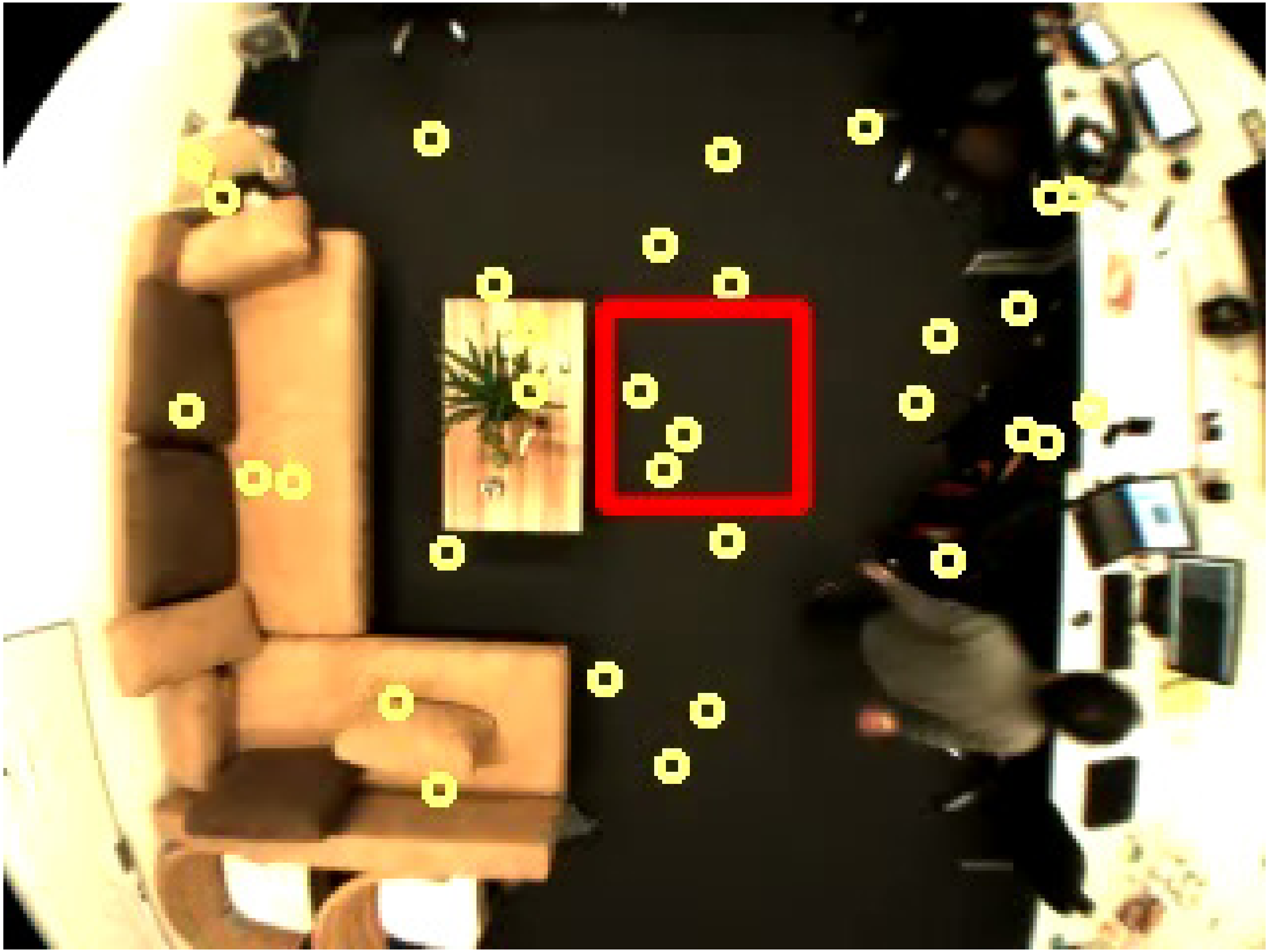
Triesch, Malsburg. Democratic Integration: Self-Organized Integration of Adaptive Cues. Neur Comp, 2001

Person Tracking from a Ceiling Camera



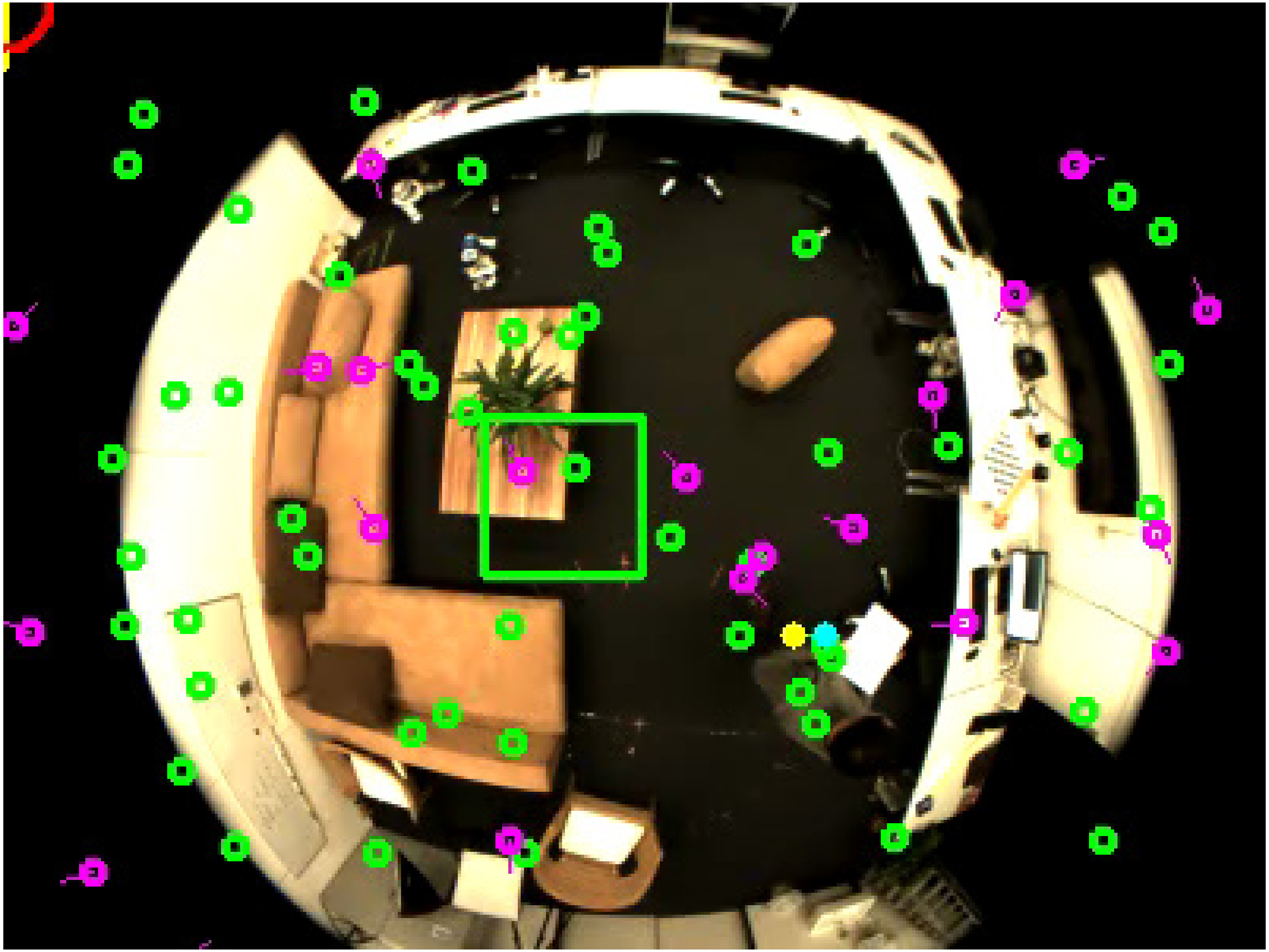
Yan, Weber, Wermter. A hybrid probabilistic neural model for person tracking based on a ceiling-mounted camera. 2011

Person Tracking from a Ceiling Camera (2)



Yan, Weber, Wermter. A hybrid probabilistic neural model for person tracking based on a ceiling-mounted camera. 2011

Use of Person Tracking



Yan, Weber, Wermter. A neural approach for robot navigation based on cognitive map learning. 2012

Summary

- Ensembles better than an individual
- Diversity is key
- Bagging – resampling of data
- Boosting – reweighting of data – AdaBoost
- AdaBoost with Hopfield features
- Democratic Integration of Adaptive Cues