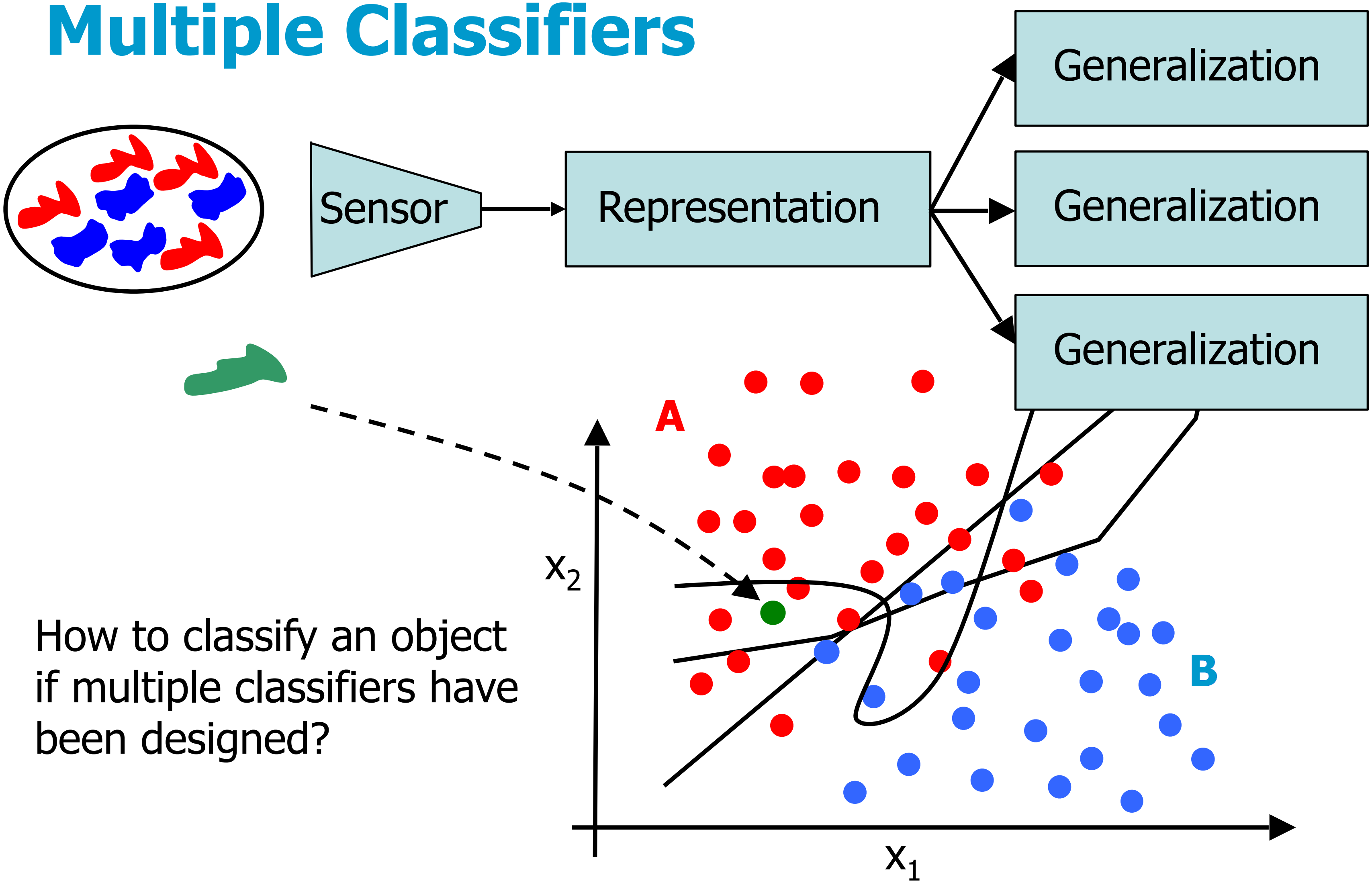
Introduction to Combining Classifiers

D.M.J. Tax
Delft University of Technology

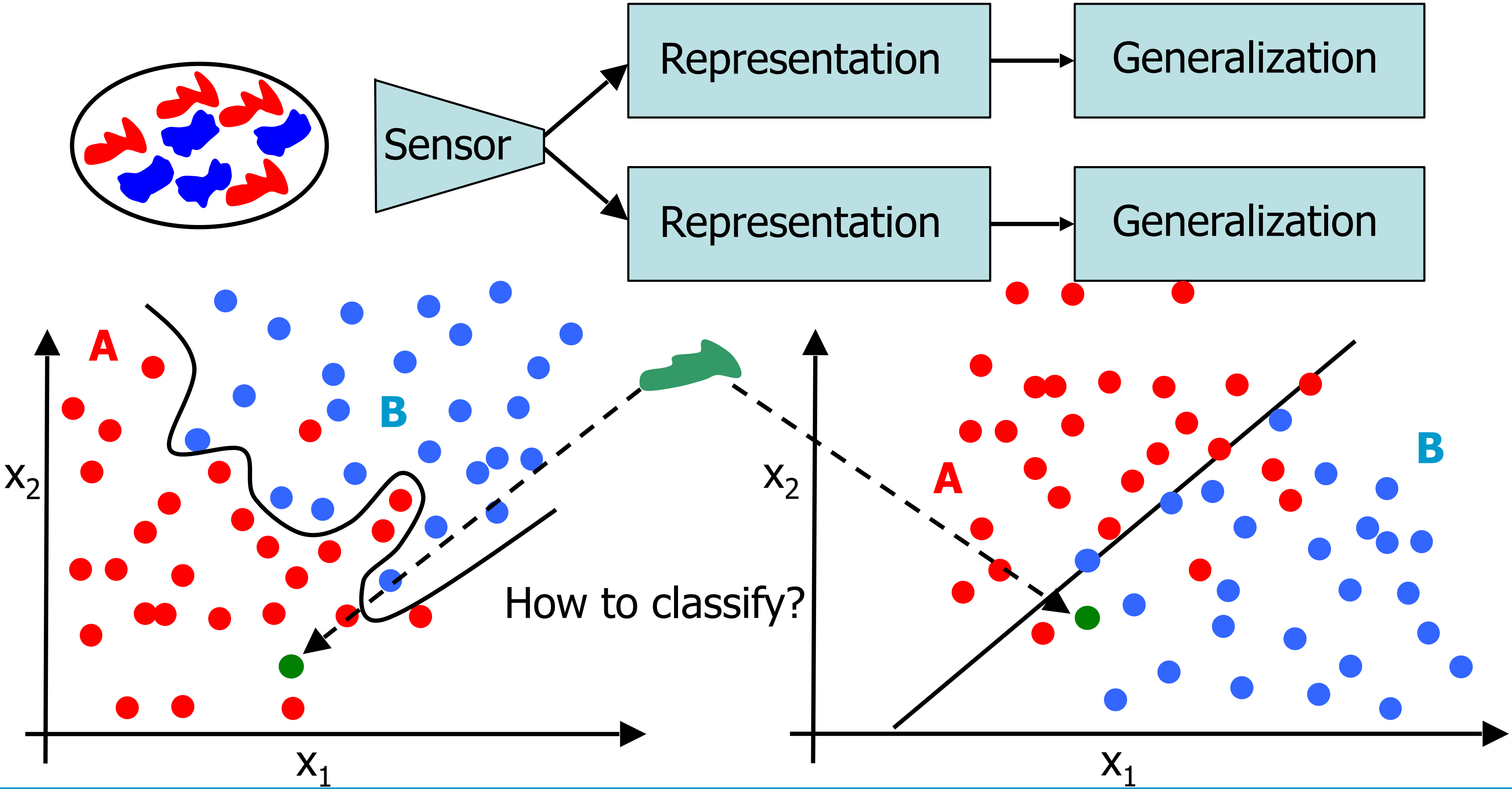
Multiple Classifiers



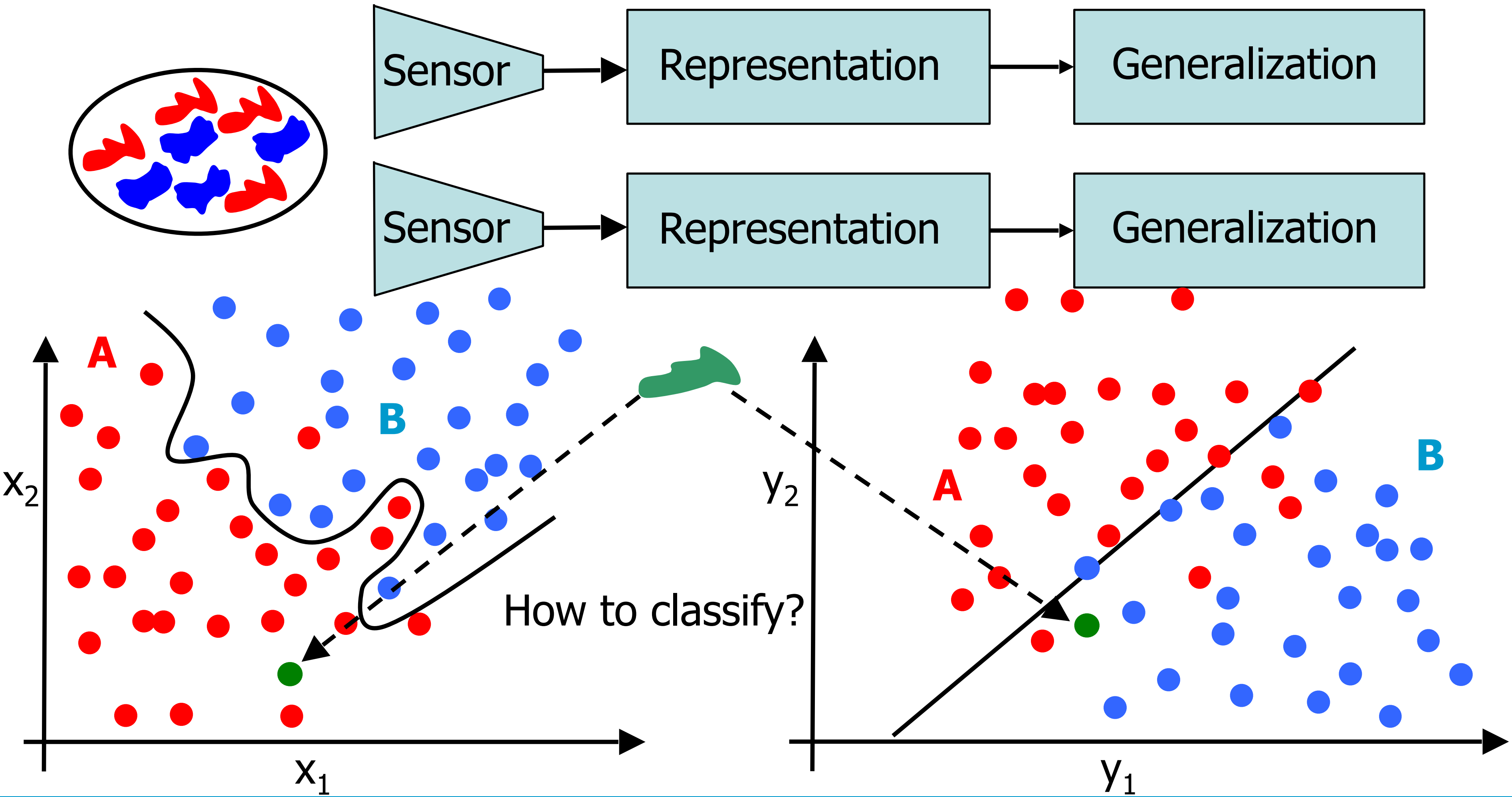
How to classify an object if multiple classifiers have been designed?

TU Delft

Multiple Representations

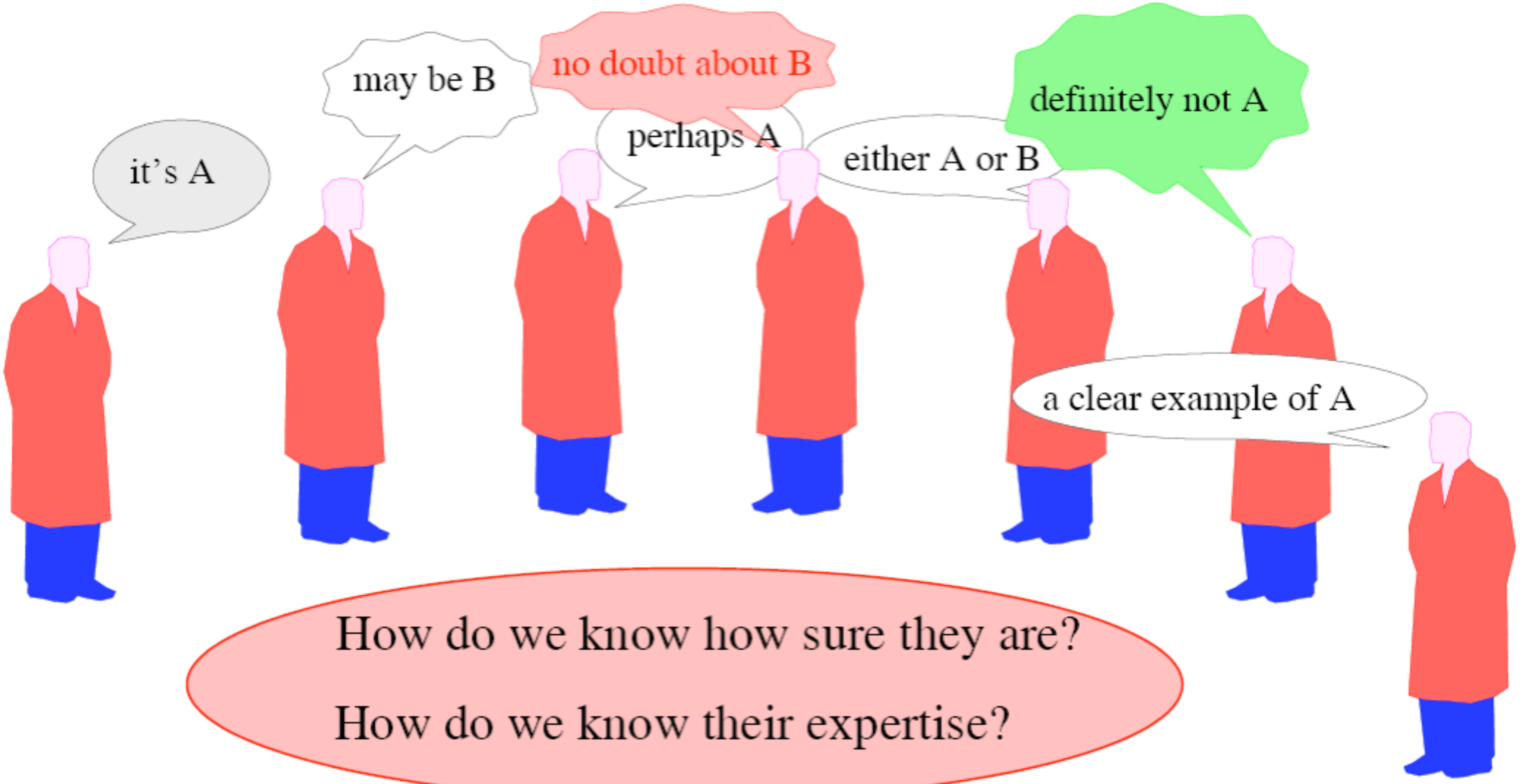


Multiple Sensors



TU Delft

Multiple Experts



The Combiner and the Base Classifiers

The basic questions:

- How to reach a committee decision?

→ How to design a combining classifier?

- How to constitute a committee?

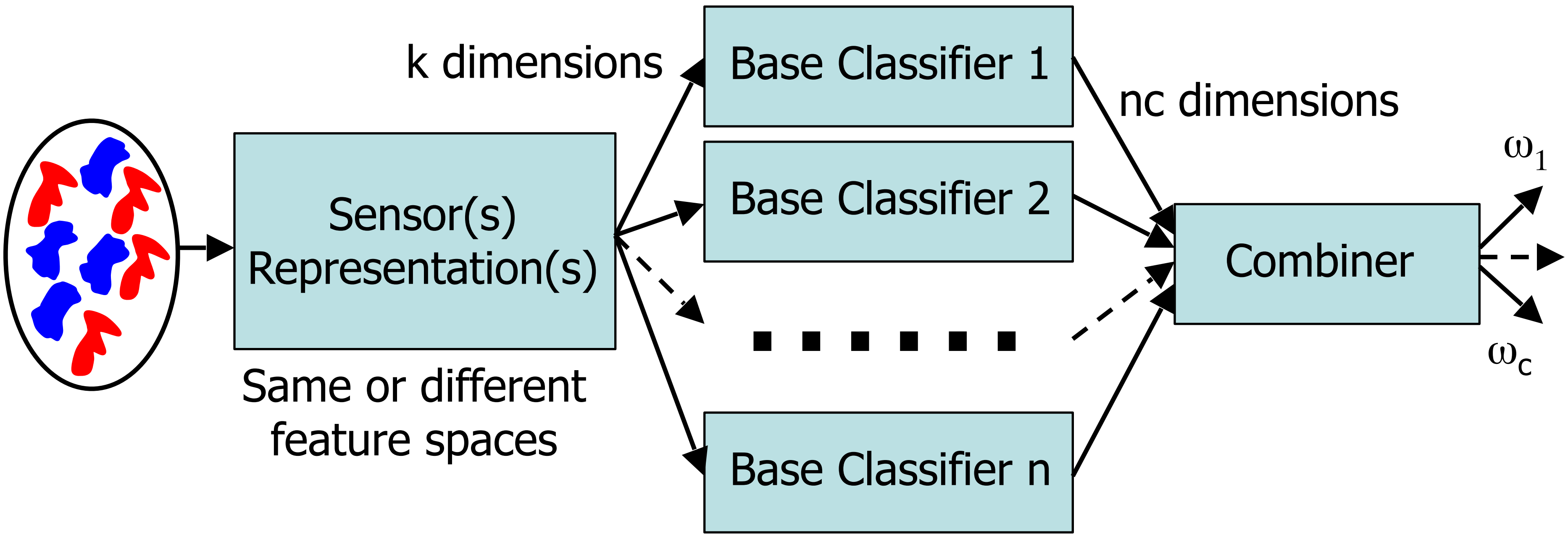
→ How to generate the base classifiers?

Combining Classifiers

Part I

The Combiner

Combining Classifier Architecture



Combining scheme

- Objects: $\{o_i\}$
- Features: $x = F(o)$

  User defined representation
- Base classifiers: $y = S(x \mid par\_base)$

  par_base : parameters optimized by training set
- Combining classifier $z = C(y \mid par\_comb)$

  par_comb : parameters optimized by training set
  (sometimes fixed, untrained combiners are used)

TU Delft

Combiners

- Fixed rules based on crisp labels or confidences (estimated posterior probabilities).
- Special trained rules based on ‘classifier confidences’.
- General trained rules interpreting base-classifier outputs as features.

Fixed Combining Rules

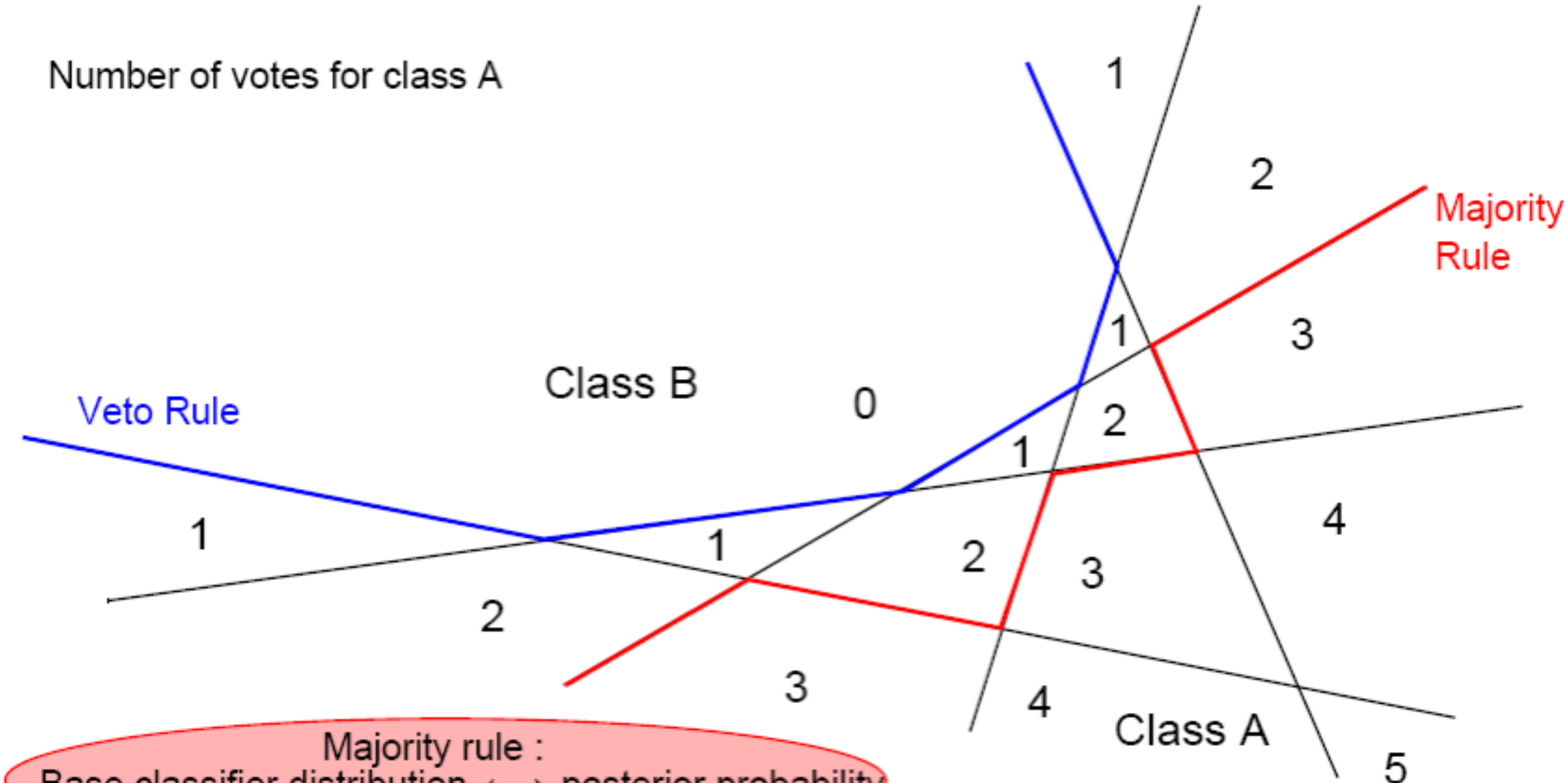
$$C(y) = \omega_i \text{ if } \operatorname{argmax}_i(\text{comb\_rule}_{j|i}(y_{ij} = S_{ij}(x_j))) = i$$

An object is assigned to class ω_i if the combination of the outcomes y_{ij} for class ω_i over all classifiers $y_j = S_j(x)$ is maximum.

- Voting, over labels $y_{ij} \in \{0,1\}$
- Product, minimum $y_{ij} \in [0,1]$
- Sum, median $y_{ij} \in [0,1]$
- Maximum $y_{ij} \in [0,1]$

TU Delft

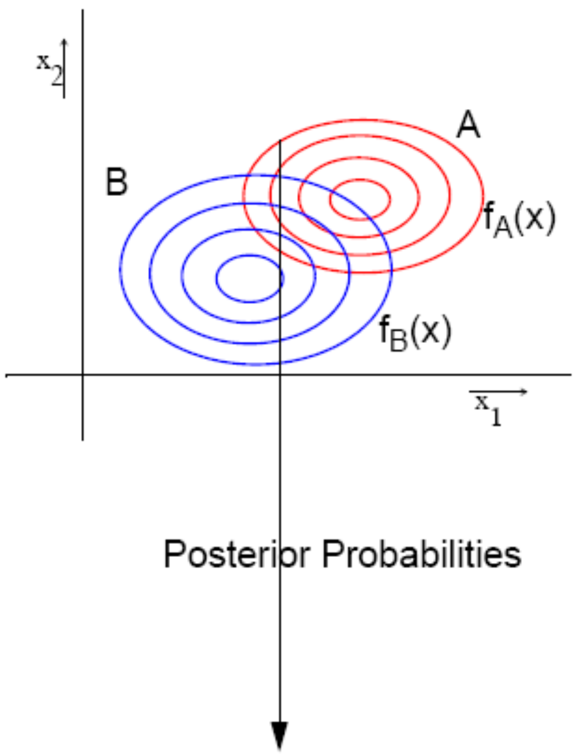
Voting; Majority Rule, Veto Rule

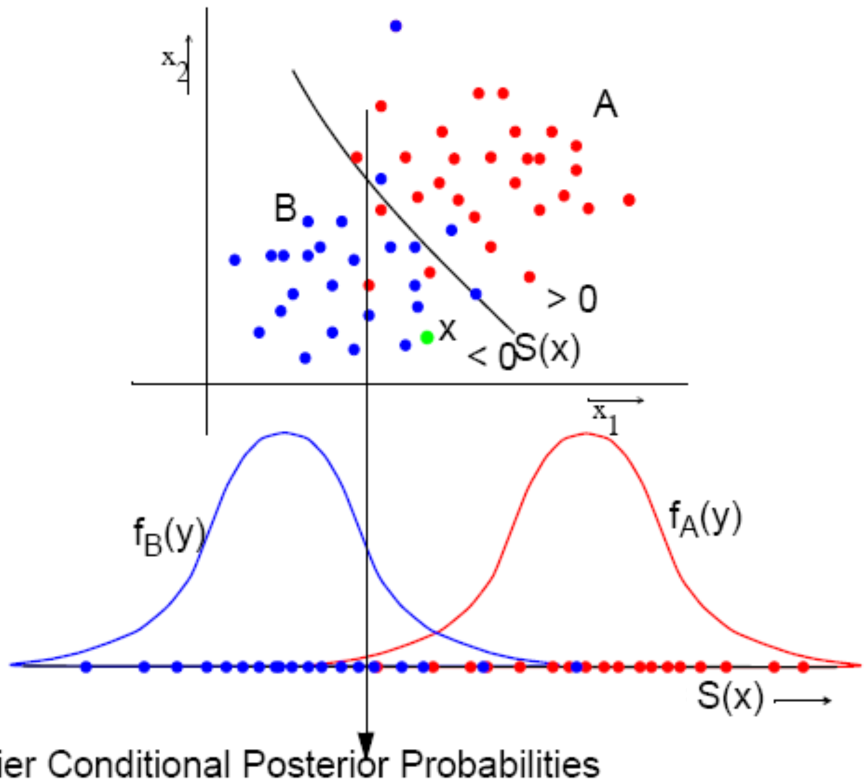


Majority rule :
Base classifier distribution $\longleftrightarrow$ posterior probability

TU Delft

Confidences, Posterior Probabilities



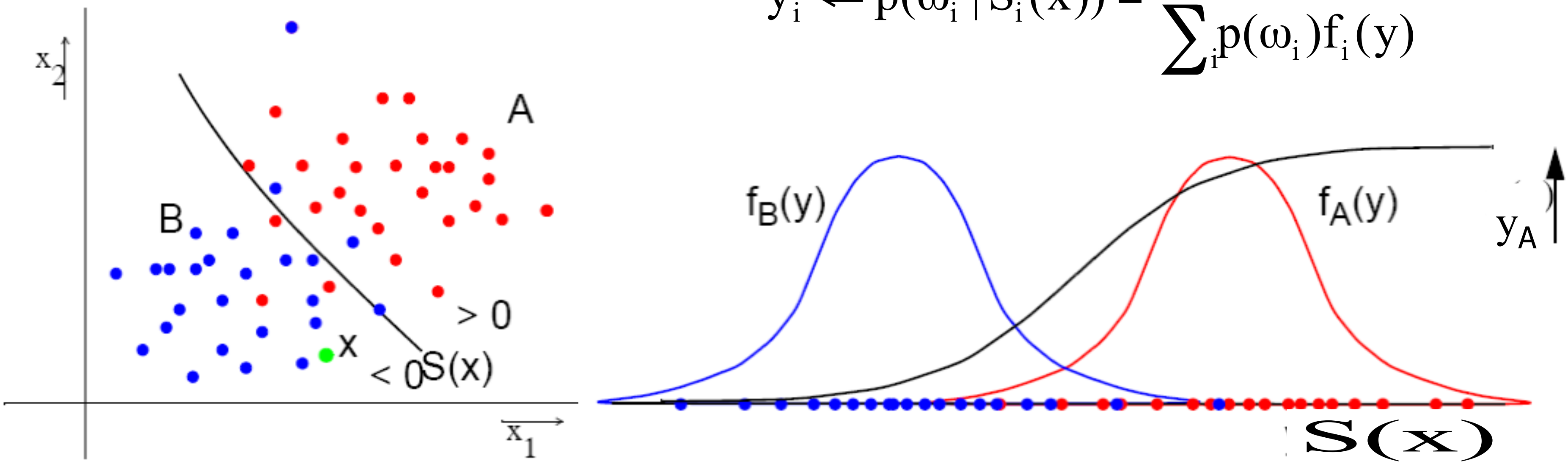


$$y_i = S_i(x) = p(\omega_i \mid x) = \frac{p(\omega_i) f_i(x)}{\sum_i p(\omega_i) f_i(x)}$$

$$y_i \Leftarrow p(\omega_i \mid S_i(x)) = \frac{p(\omega_i) f_i(y)}{\sum_i p(\omega_i) f_i(y)}$$

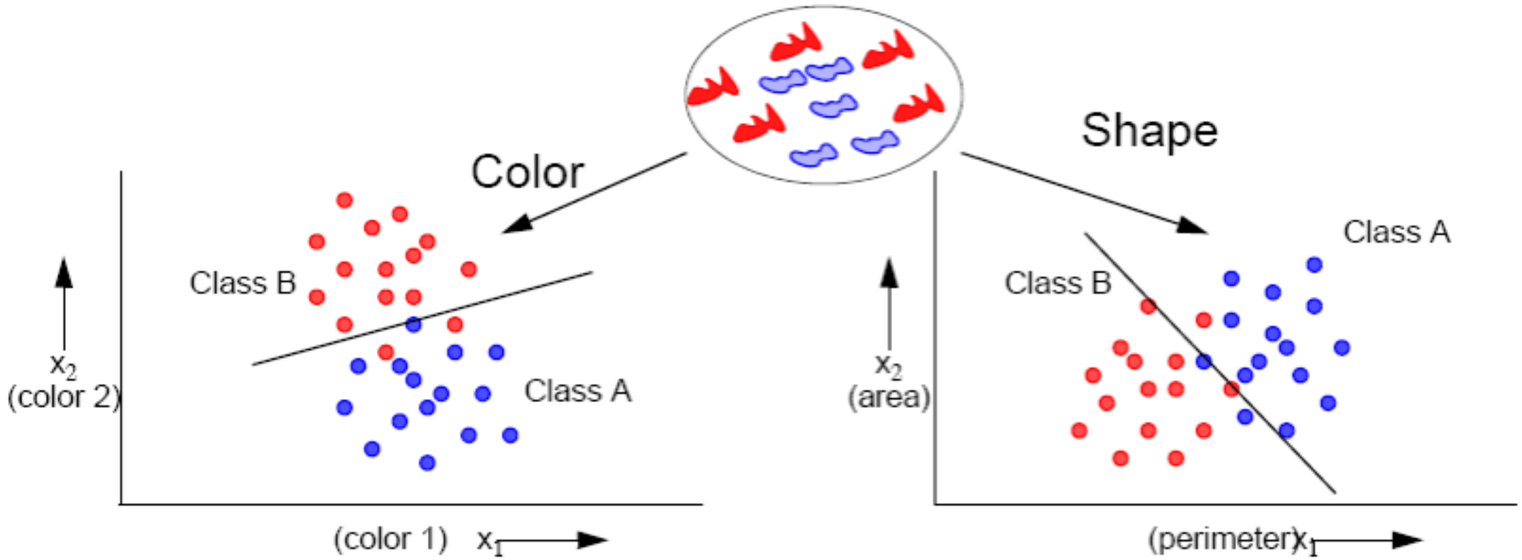
Optimal Scaling for Classifier Conditional Posterior Probabilities

$$y_i \Leftarrow p(\omega_i \mid S_i(x)) = \frac{p(\omega_i) f_i(y)}{\sum_i p(\omega_i) f_i(y)}$$



Fit a logistic function or sigmoid to the data such that $\prod_{x \in \text{Trainingset}} y(S(x))$ is maximum conditional to $y = 0.5$ for $S(x) = 0$.

Combining Different Representations



Base classifier j posterior probabilities for class A : $y_{A_j} = \mathrm{Prob}_j(A|x_j)$

Product Rule: $y_A = \prod \mathrm{Prob}_j(A|x_j)$, $y_B = \prod \mathrm{Prob}_j(B|x_j)$,

prodc

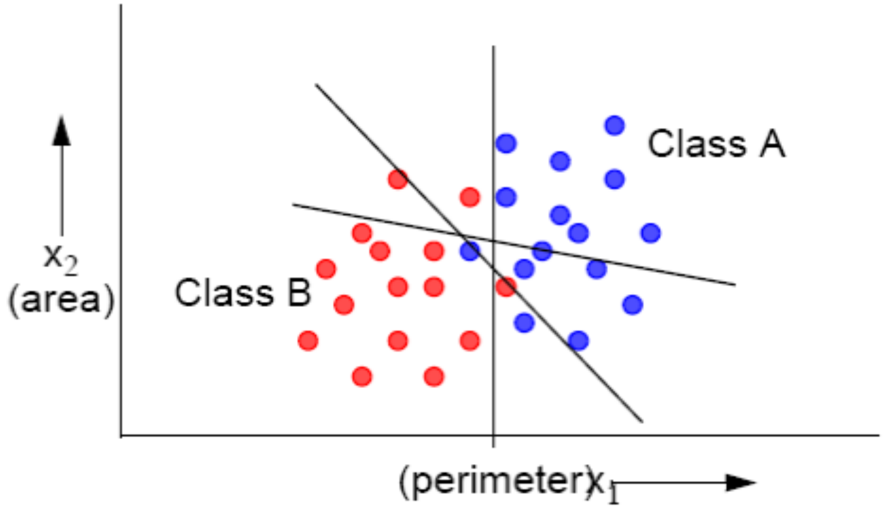
Useful for 'independent' feature spaces (logical 'AND', experts should agree)

Minimum Rule: $y_A = \mathrm{Min}\{\mathrm{Prob}_j(A|x_j)\}$, $y_B = \mathrm{Min}\{\mathrm{Prob}_j(B|x_j)\}$

Assign according to 'least objecting expert'

minc

Combining Different Classifier Estimates



Base classifier j posterior probabilities for class A : $y_{Aj} = \mathrm{Prob}_j(A|\mathbf{x})$

meanc

Sum (Mean) Rule: $y_A = \sum \mathrm{Prob}_j(A|\mathbf{x})$, $y_B = \sum \mathrm{Prob}_j(B|\mathbf{x})$,

Useful for improved estimates of posterior probabilities

Also: Median and Majority Voting

medianc

Improvement by averaging out mistakes of experts

The Product and the Minimum Rule

Base classifier j posterior probabilities for class A : $y_{A_j} = \mathrm{Prob}_j(A|x_j)$

Product Rule: $y_A = \prod \mathrm{Prob}_j(A|x_j)$, $y_B = \prod \mathrm{Prob}_j(B|x_j)$,

Useful for 'independent' feature spaces, see *Kittler, IEEE-PAMI-20(3),1998*

Minimum Rule: $y_A = \mathrm{Min}\{\mathrm{Prob}_j(A|x_j)\}$, $y_B = \mathrm{Min}\{\mathrm{Prob}_j(B|x_j)\}$

Assign according to 'least objecting classifier'

objects	Classifier 1		Classifier 2		Product		Minimum	
	Class A	Class B	Class A	Class B	Class A	Class B	Class A	Class B
1	0.4	0.6	0.2	0.8	0.08	0.48	0.2	0.6
2	0.1	0.9	0.7	0.3	0.07	0.27	0.1	0.3
3	0.3	0.7	0.4	0.6	0.12	0.42	0.3	0.6
4	0.5	0.5	0.2	0.8	0.10	0.40	0.2	0.5
5	0.0	1	0.9	0.1	0.00	0.10	0.0	0.1
6	0.8	0.2	0.2	0.8	0.16	0.16	0.2	0.2

Fixed Combining Rules Overview

Product, Minimum

- Independent feature spaces
- Different areas of expertise
- Error free posterior probability estimates

prodc minc

Ever optimal?

Sum (Mean), Median, Majority Vote

- Equal posterior-estimation distributions in same feature space
- Differently trained classifiers, but drawn from the same distribution
- Bad if some classifiers (experts) are very good or very bad

meanc majorc

Maximum

- Trust the most confident classifier / expert
- Bad if some classifiers (experts) are badly trained

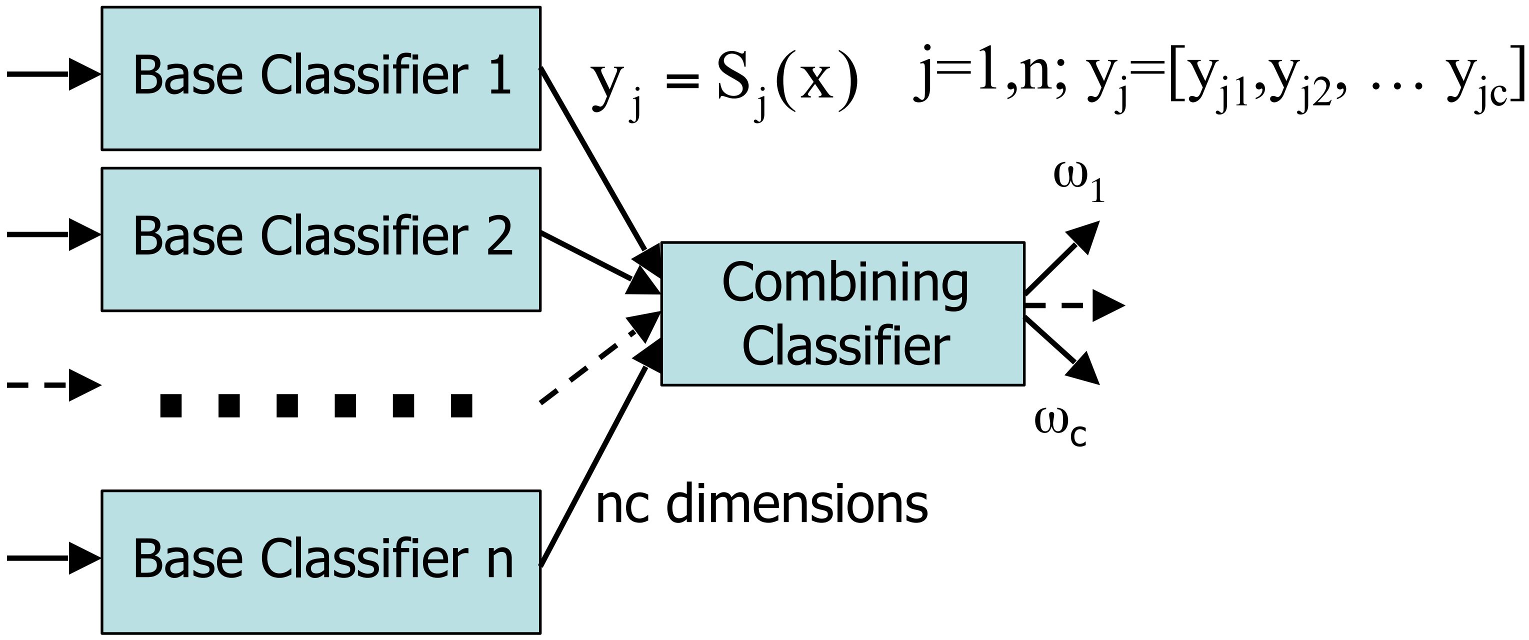
maxc

Fixed Combining Rules are Sub-optimal

- Base classifiers are never really independent (product)
- Base classifiers are never really equally imperfectly trained (sum, median, majority)
- Sensitivity to over-confident base classifiers (product, min, max)

- Fixed combining rules are never optimal
- Larger training sets do not really improve this (except max?)

Trained Combiners



General rules neglect the classification-confidence characteristic of the base classifier outputs, as they are treated as general feature values.

Trained Combiners (2)

Special Trained Combiners

- DT: Decision Templates (~ Nearest Mean)
- (BKS: Behavioral Knowledge Space)
- (DCS: Dynamic Classifier Selection)
- (ECOC: Error Correcting Output Coding)
- NN: Neural Networks

General Classifiers

- Nearest Mean
- Fisher
- Decision Trees
- ……

TU Delft

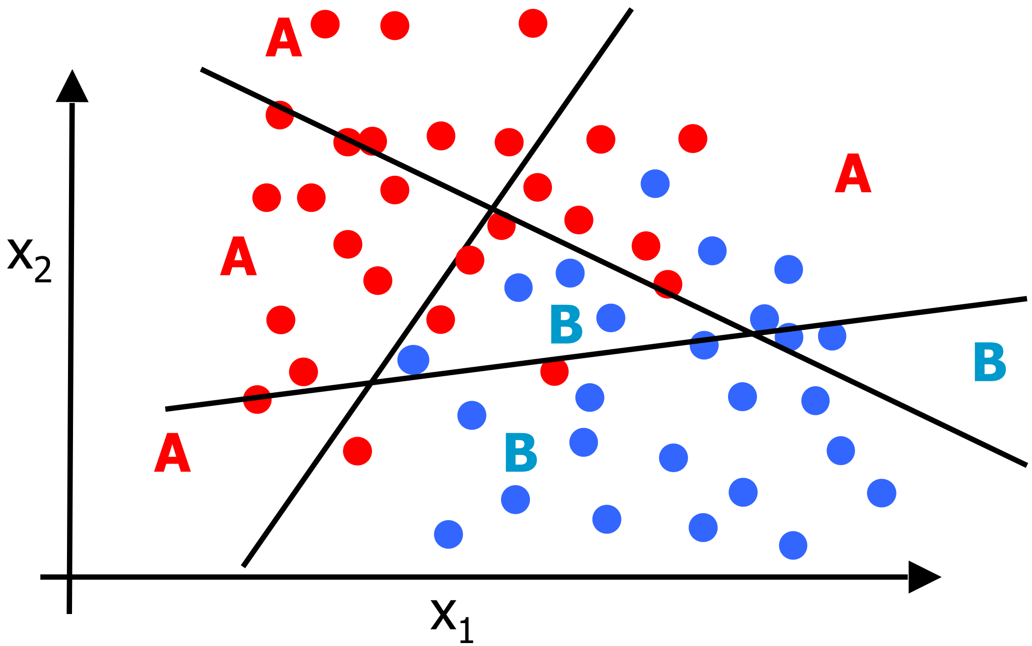
Decision Templates

- Determine, using a training set, the average outcomes of the base classifiers per class (decision templates, i.e. class means in the base-classifier outcome space).

- Assign new objects to the class of the nearest decision template in the base-classifier outcome space.

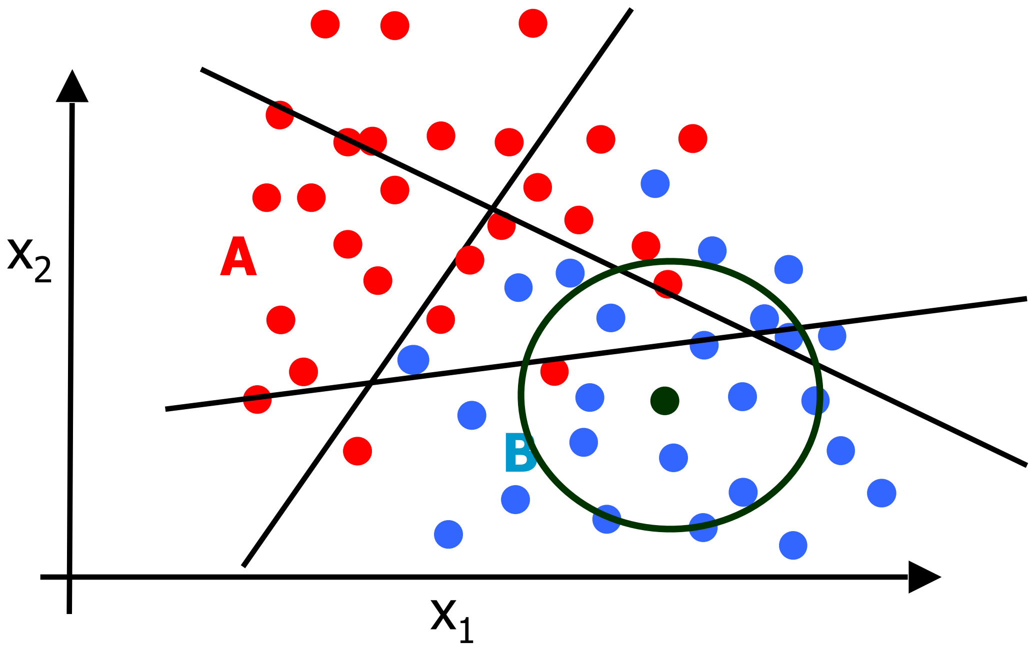
(Issue: is it good to have posterior probabilities for the base-classifier outputs, or yield inverse sigmoids a better scaling?)

Behavioral Knowledge Space



Determine on the basis of a training set for every cell in the original feature space the preferred class and assign new objects accordingly.

Dynamic Classifier Selection



- Select the classifier that classifies most of the k nearest neighbors of a test object correctly.

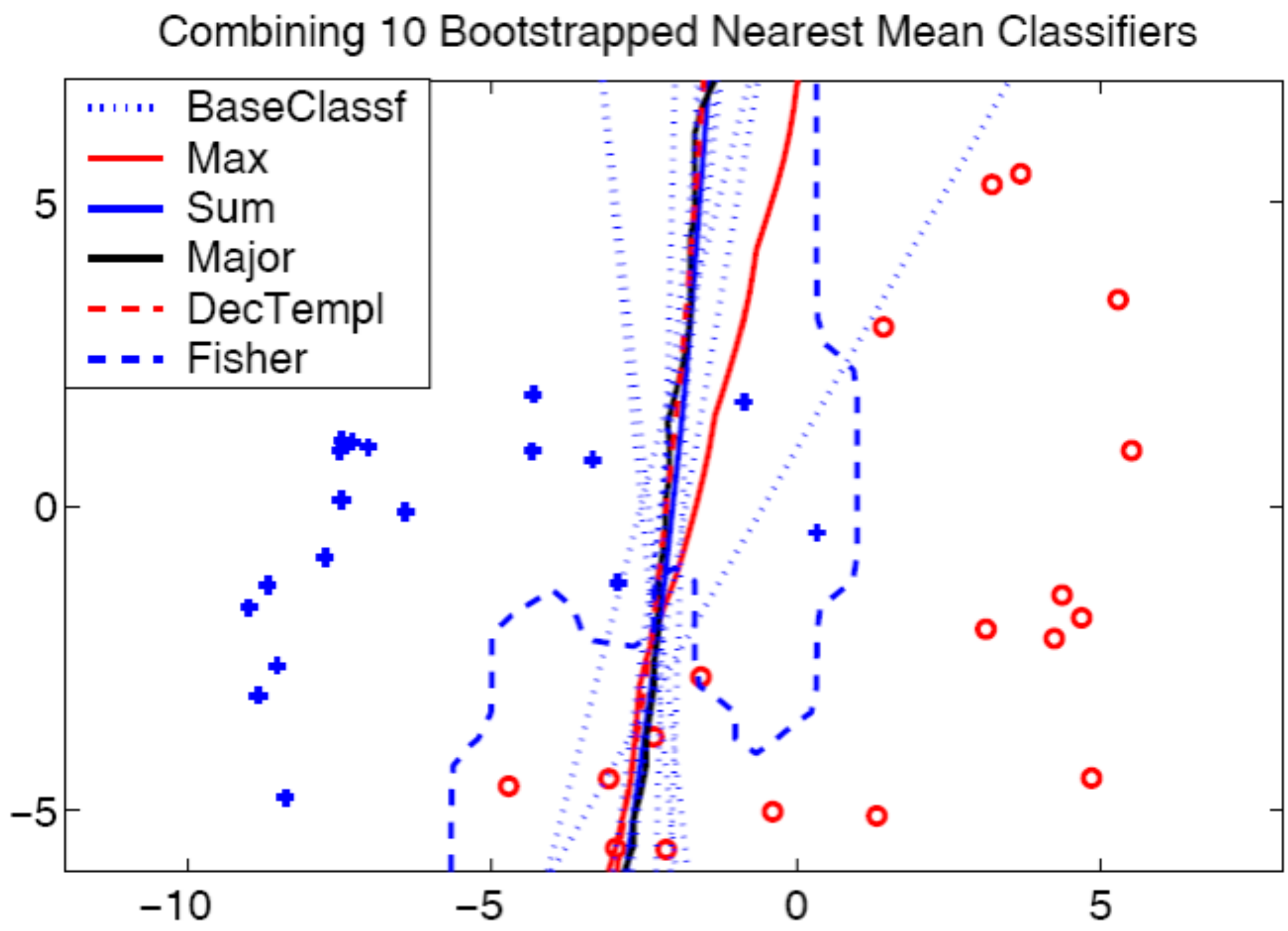
- Use this classifier to classify the test object.

ECOC: Error Correcting Output Coding

- ECOC is a system to use a small set of binary (i.e. two-class) classifiers for a large set of c classes.
- n classifiers can distinguish at most $c=2^n$ classes.
- If $n > {}^2\log(c)$ the system of classifiers is more robust
- ECOC studies mainly discuss the coding scheme, not the way base classifiers are trained. Combining is done by using the crisp, {0,1}-labels.

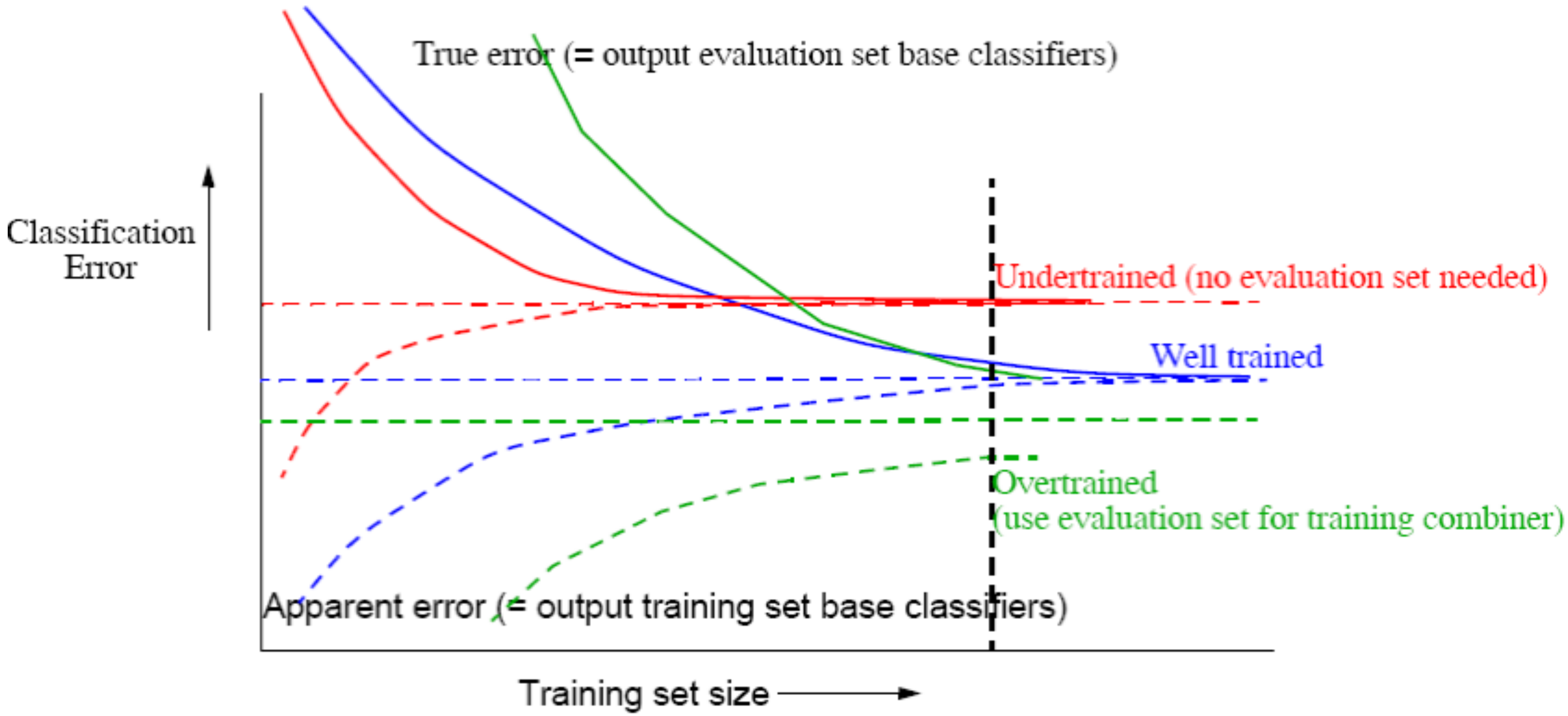
TU Delft

Example



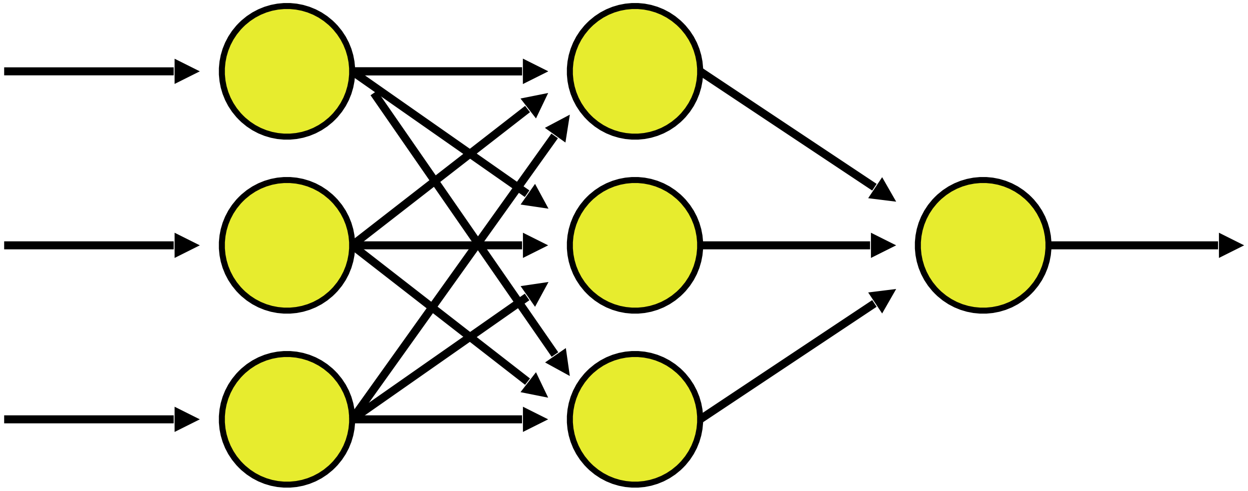
Combining 10 Bootstrapped Nearest Mean Classifiers
BaseClassf
Max
Sum
Major
DecTempl
Fisher
5
0
−5
−10
−5
0
5

Undertrained, Well Trained, Overtrained



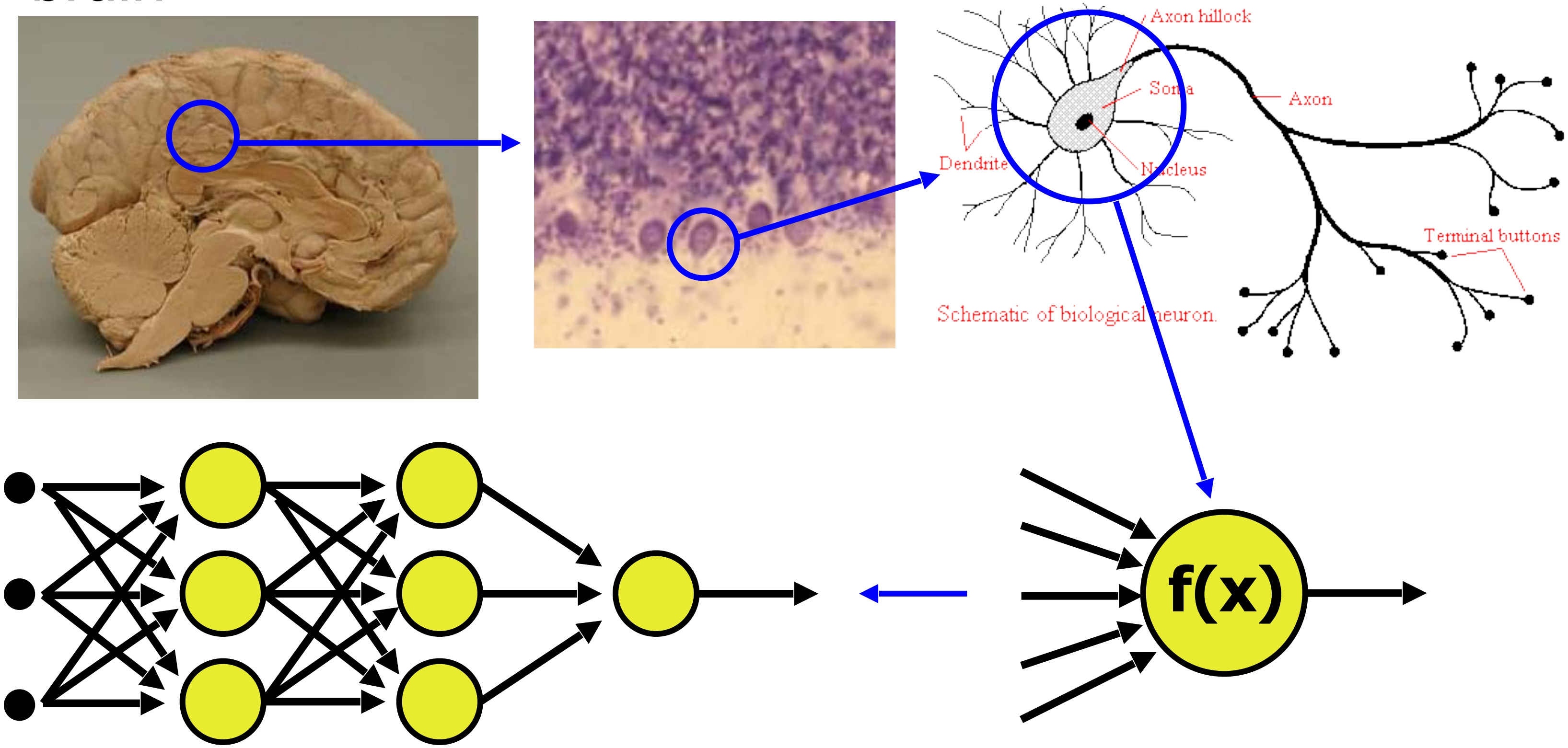
Artificial neural networks (1)

- Large, densely interconnected networks of simple processing units



Artificial neural networks (2)

- Some (not all!) networks originally inspired by the brain



Artificial neural networks (3)

- Research started in the 1950s
- Took off after 1986 – big hype for about 10-15 years
  - brought together psychologists, neurologists, philosophers, machine learners, statisticians...
  - helped thinking about, among others, pattern recognition
  - resulted in a lot of grant money

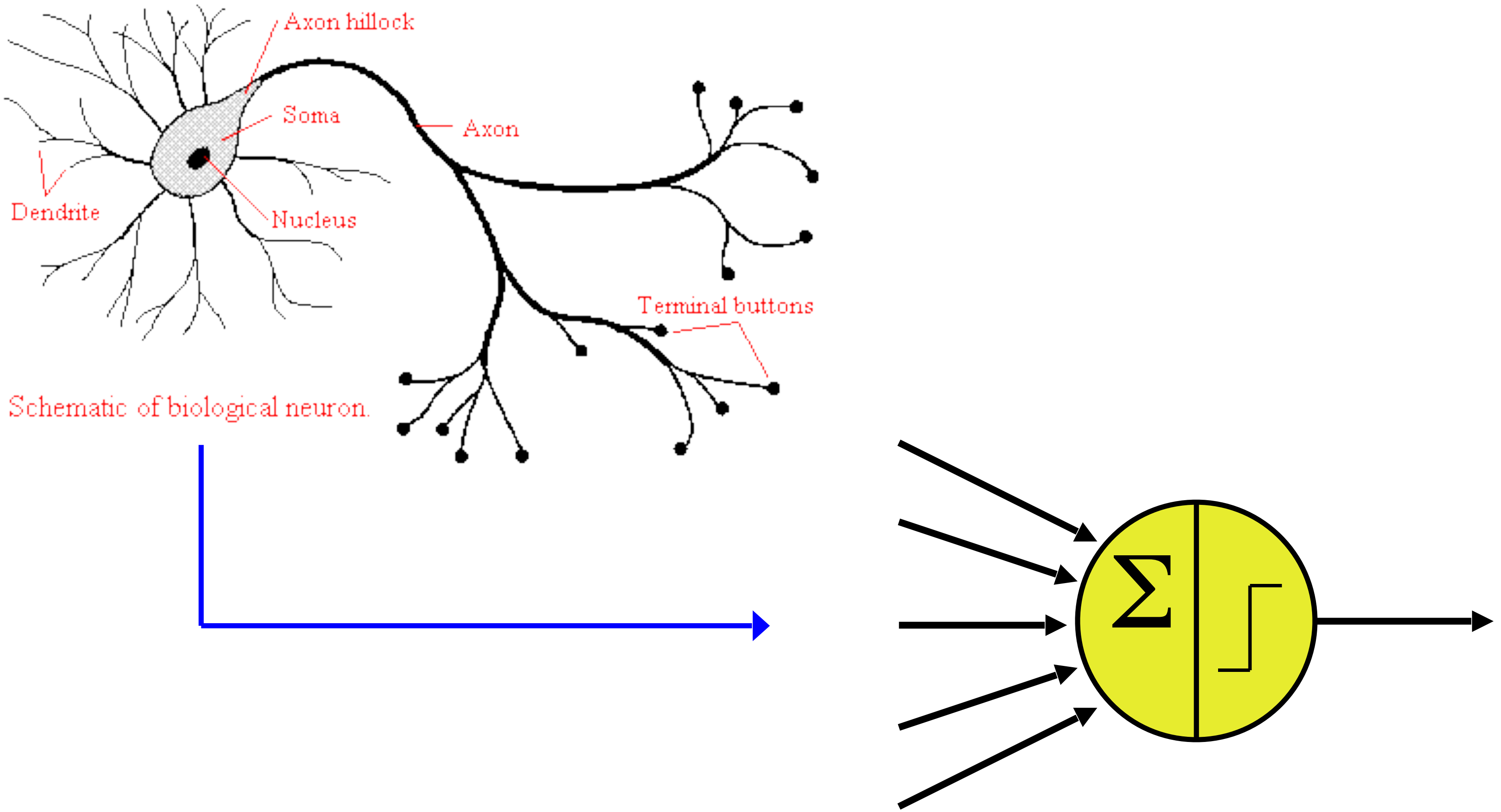
- Now: useful tool in many areas, with many pitfalls

TU Delft

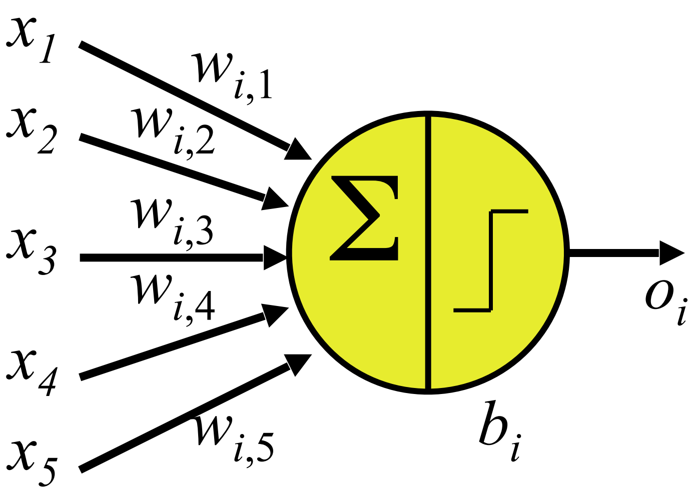
History

- 1943 : McCulloch and Pitts: model of neuron
- **1958** : Rosenblatt: perceptron
- 1960s : Rosenblatt, Nilsson work on perceptrons
- 1968 : Minsky and Papert point out limitations: perceptrons are linear
- 1982 : Hopfield network (associative memory), Kohonen's self-organising map (clustering), Fukushima's Neocognitron (vision)
- **1986** : Rumelhart, Hinton and Williams: training of nonlinear networks
- 1980s, 1990s: various theoretical developments
- 2010s : More data, faster hardware (GPUs)

McCulloch-Pitts model



McCulloch-Pitts model (2)



weights inputs

output $$o_i = \phi\left(\sum_j w_{ij} x_j - b_i\right)$$

threshold or bias

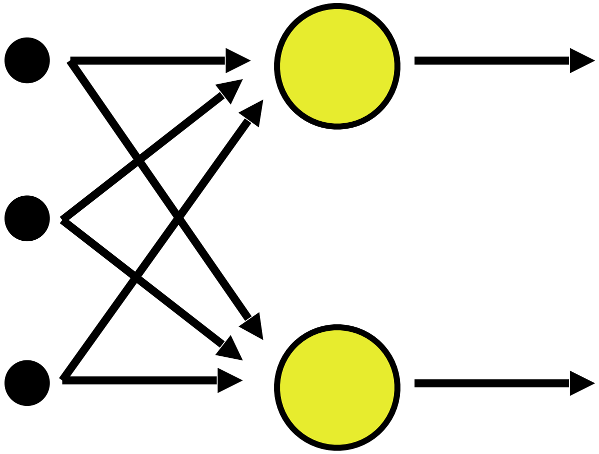
with $$\phi(a) = \begin{cases} 1 & a \geq 0 \\ 0 & a < 0 \end{cases}$$

or $$\phi(a) = \frac{1}{1 + \exp(-a)}$$

transfer function
or
activation function

Perceptron

- Networks of McCulloch-Pitts models can perform universal computation, given the right weights $\boldsymbol{w}$: it can do anything a binary computer can do
- ...but how can we find the right weights $\boldsymbol{w}$?
- Rosenblatt (1956): possible for single layer networks, perceptrons



Perceptron (2)

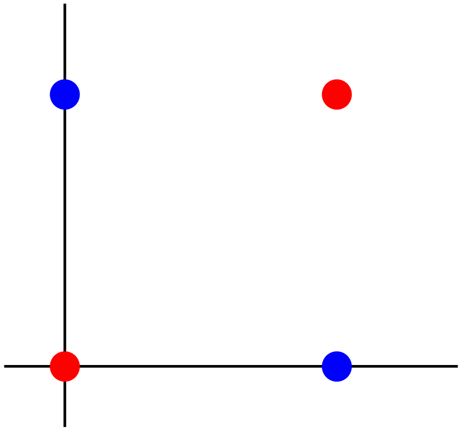
- Perceptron is a trainable two-class linear discriminant
- Training algorithm can be proven to converge to correct solution for separable classes
- Possible to extend to multiple classes
- When classes are not linearly seperable:
  - indefinite training, weights will blow up
  - solution: decrease ρ during training

TU Delft

Perceptron (3)

- Minsky & Papert (1969): perceptrons are limited

x_1	x_2	ω
0	0	A
1	0	B
0	1	B
1	1	A



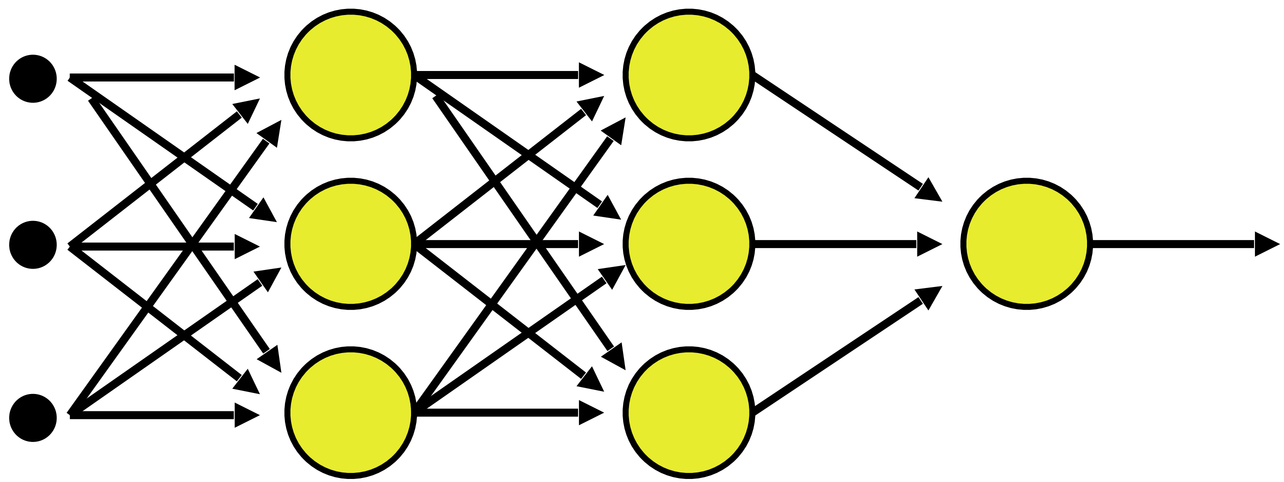
The XOR problem cannot be solved by a linear discriminant such as the perceptron

- When classes are nonlinearly separable:
  - nonlinear transfer functions
  - multilayer perceptron – but how to find weights...?
  - Rumelhart et al. (1986): use the chain rule

Multilayer perceptron

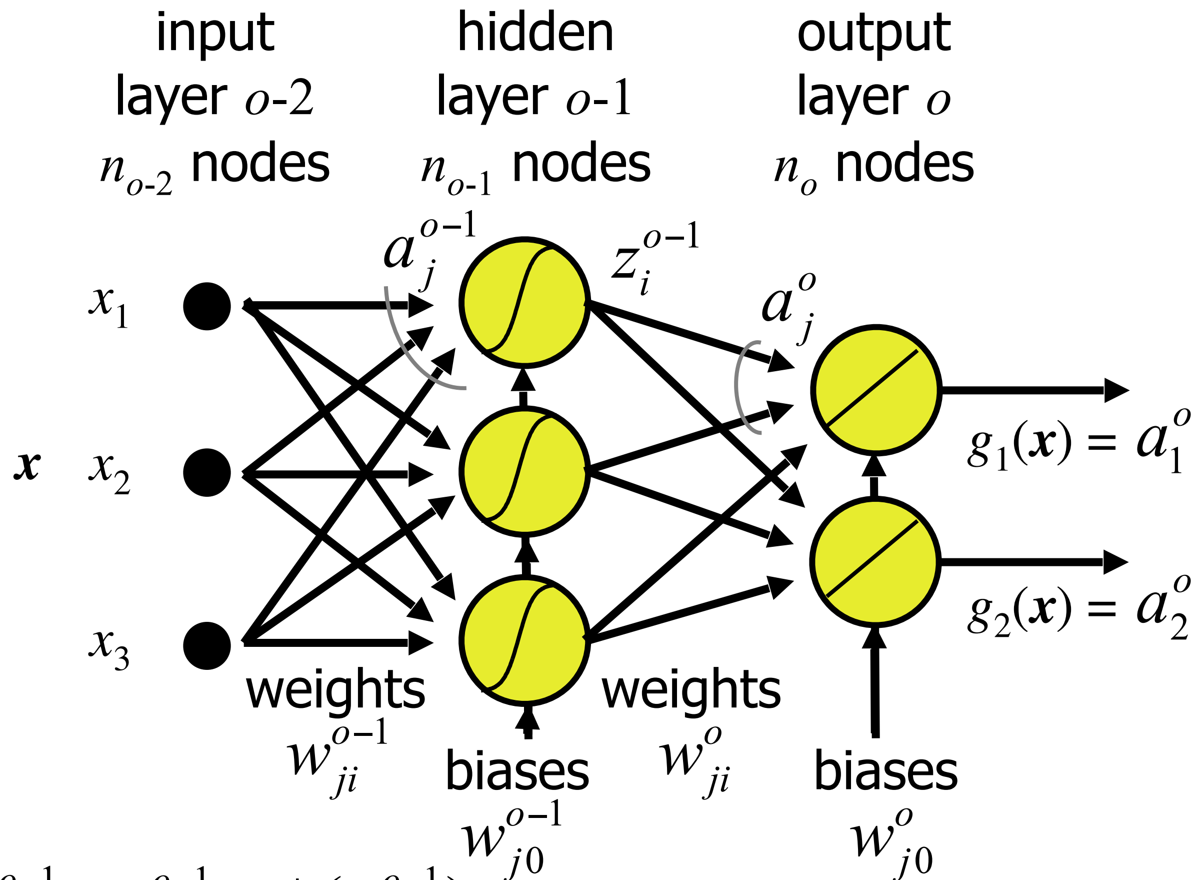
- Stacked perceptrons: feedforward networks
- Each unit has a nonlinear transfer function,

e.g. $\phi(a) = \dfrac{1}{1+\exp(-a)}$ sigmoid or logistic function



Backpropagation training

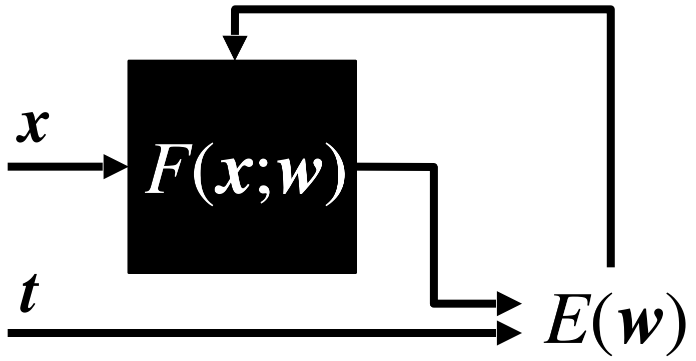
- To explain training: simple network
- One hidden layer, linear output units



- $a_j^o = \sum_{i=1}^{n_{o-1}} w_{ji}^o z_i^{o-1}, \quad z_j^{o-1} = \phi(a_j^{o-1})$

Other training algorithms

- Backpropagation training is simple gradient descent, but implemented in a useful way: all updates can be calculated locally (in parallel)



- Other view: simply optimise MSE E w.r.t. weight vector w
  using any optimisation routine, e.g.
  - second order (Newton, pseudo-Newton)
  - conjugate gradient descent
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
  - Levenberg-Marquardt (LM, in PRTools)

TUDelft

Multilayer perceptrons (2)

- Choices:
  - targets (0/1, 0.1/0.9, 0.2/0.8) t
  - **number of hidden layers**
  - **number of units per hidden layer** n_i
  - transfer functions $\phi\ (a)$
  - initialisation $w^{(0)}$
  - training algorithm
  - parameters (learning rate ρ etc.)
  - convergence decision E_{thr} or test set selection
  - ...
- All of these influence results!

TU Delft

Multilayer perceptrons (3)

- Number of weights = number of parameters = $\sum_{l=1}^{o-1}(n_l+1)n_{l+1}$
  e.g. for p = 10, C = 2, 2 20-unit hidden layers:
  $(10+1)\cdot 20+(20+1)\cdot 20+(20+1)\cdot 2$ = 682 parameters

- Danger of overtraining!

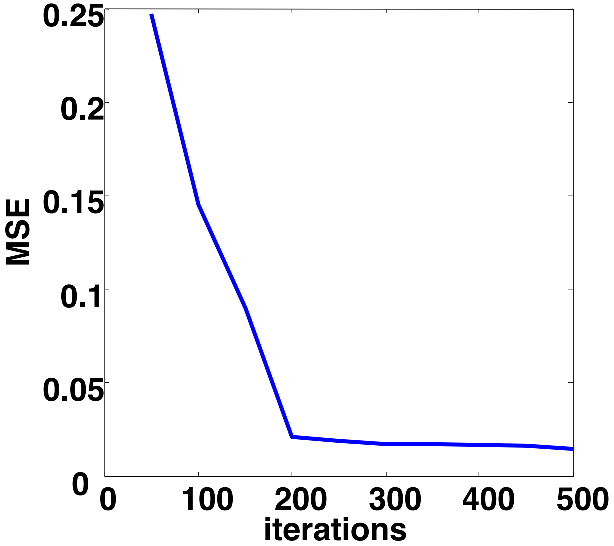
- Prevention:
  - use small networks
  - regularise: minimise $E(\boldsymbol{w})+\lambda\|\boldsymbol{w}\|$
  - small w's: low complexity, training slowly increases w's;
    so when stopping in time: automatic regularisation!

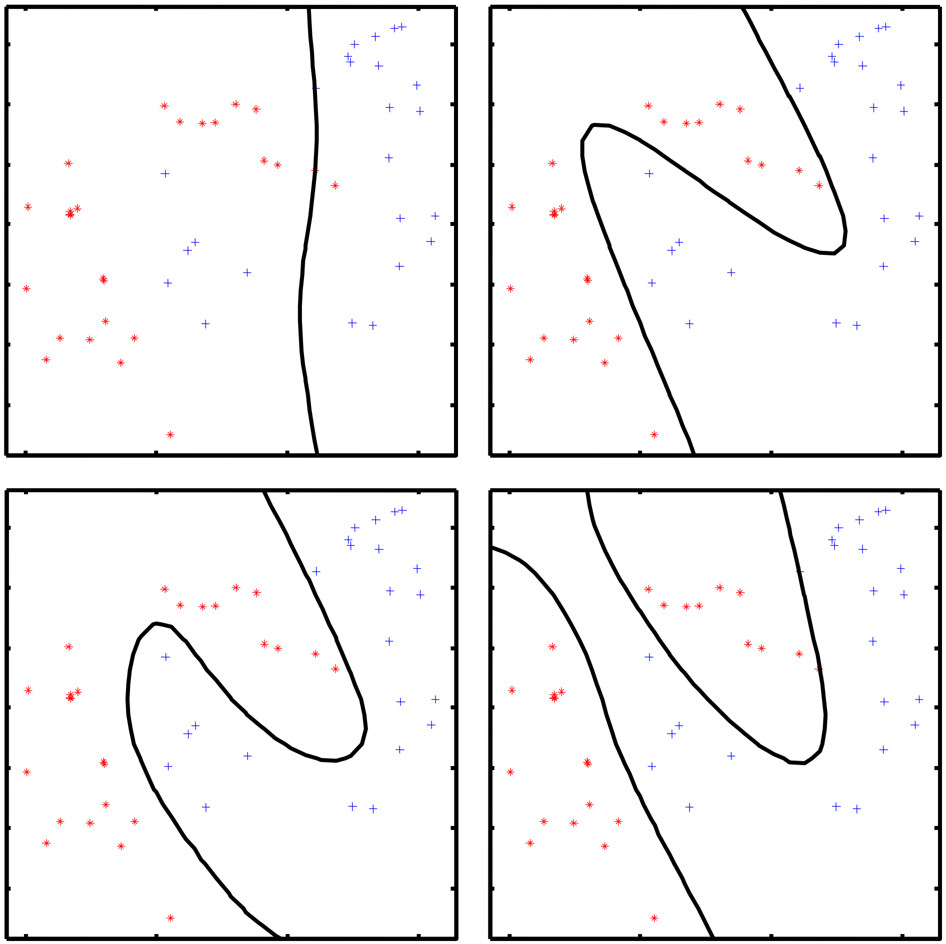
TUDelft

Multilayer perceptrons (4)

- Examples:

1 hidden layer
of 3 units,
2 initialisations





2 hidden layers
of 5 units each,
2 initialisations

TU Delft

Combining Classifiers

Part II
Generation of Base Classifiers

Generation of base classifiers

- Random Subspace Approach
- Bagging
- Boosting

Random Subspace Approach

- Select a dimensionality k’ << k that fits well with the training size
- Select at random n subsets of k’ features
- Train n classifiers
- Combine

- When better than feature selection?
- When better than feature extraction?

Bagging (Bootstrap and Aggregate)

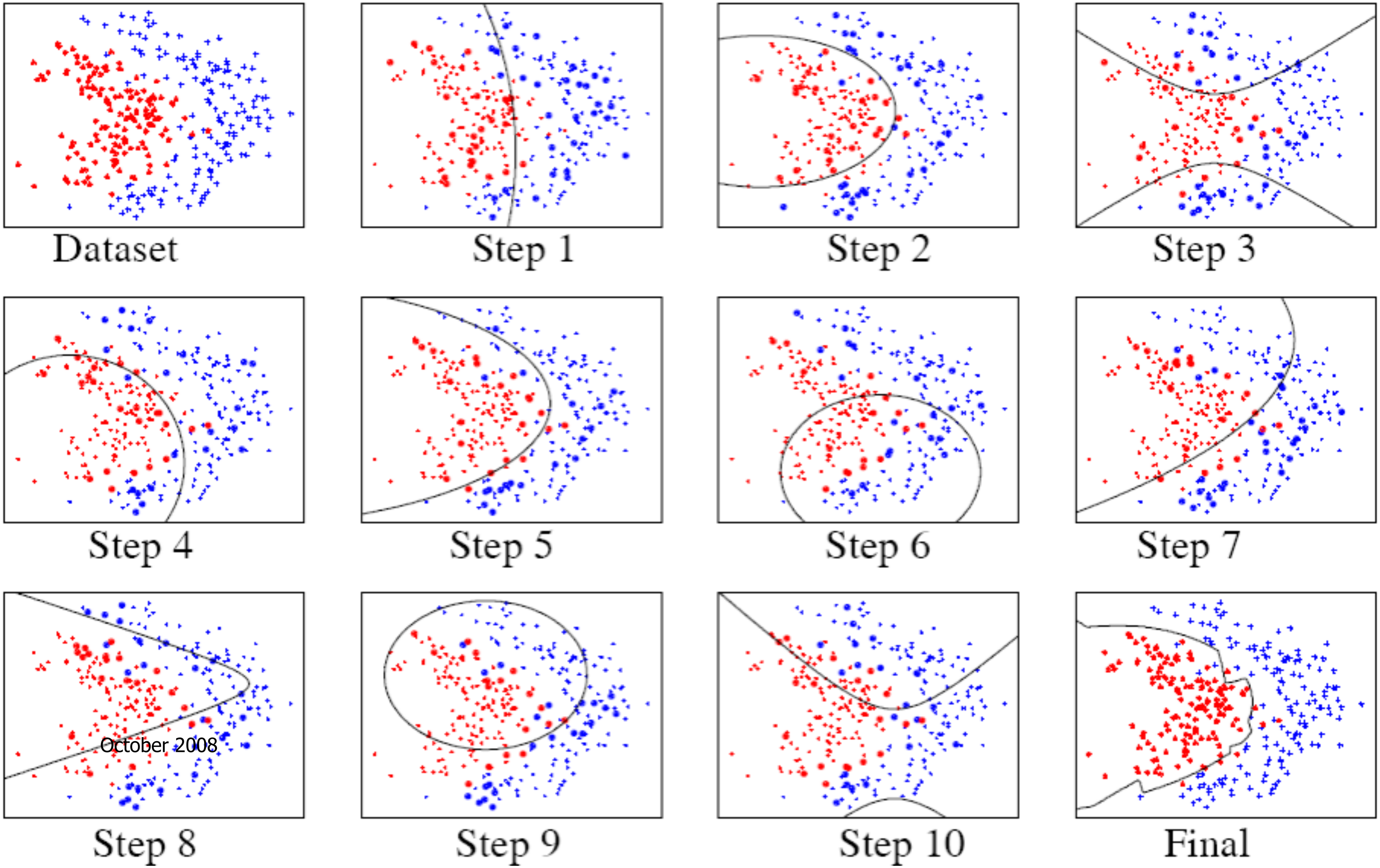
- Select a training set size m' < m
- Select at random n subsets of m' training objects (original: bootstrap)
- Train a classifier (original: decision tree)
- Combine (original: vote)

- Bagging decision trees combined with random subspace approach gives: Random Forest classifier

Boosting

- Initialize all objects with an equal weight
- Select a training set size m′ < m according to the object weights
- Train a weak classifier
- Increase the weights of the erroneously classified objects
- Repeat as long as needed
- Combine

Boosting: Emphasize Difficult Objects



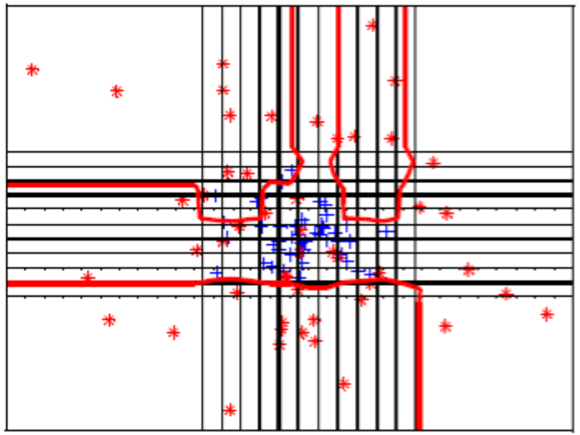
Adaboost Algorithm

1. **Sample** the training set according to a set of object weights (initially equal)
2. Use it for **training a** simple (weak) **classifier** w_i
3. **Classify** the entire data set, using the weights, **error** ε_i
   Store classifier weight $\alpha_i = 0.5 \log((1-\varepsilon_i)/\varepsilon_i)$
4. **Multiply weights** of erroneously classified objects with $\exp(\alpha_i)$
   Multiply weights of correctly classified objects with $\exp(-\alpha_i)$
5. **Goto 1** as long as needed
6. **Final classifier**: weighted voting, weights α_i

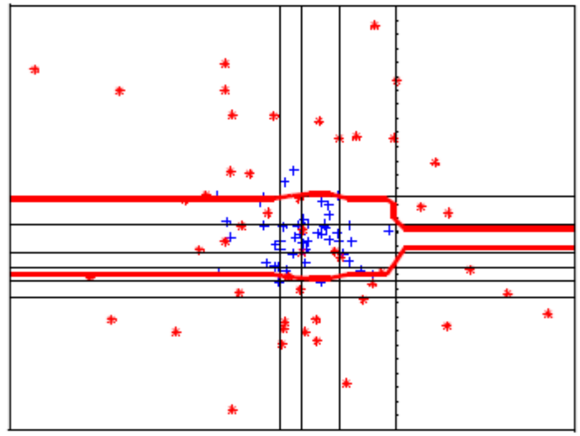
Assumes decision stump as weak classifier, and minimises exponential

loss (See course CS4230 Machine Learning 2)

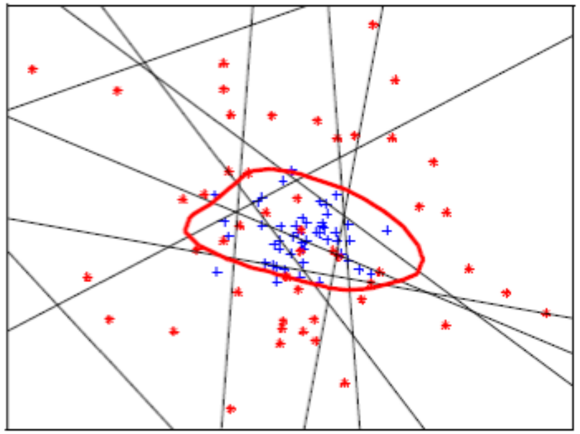
Adaboost - 2D Example



100 dec. stumps, wvote



10 dec. stumps, Fisher



10 Fisher, Fisher

```
w=adaboostc(a,stumpc,100)
```

```
w=adaboostc(a,fisherc,10,fisherc)
```

```
w=adaboostc(a,stumpc,10,fisherc)
```

Discussion on Base Classifiers

- Are to be combined
- Simple, not overtrained, especially not for trained combiners
- Many: fast training, fast execution
- Soft outputs might be helpful
- Traditional: decision trees, decision stumps, linear, quadratic
- Weak classifiers: simple, should do something,
- not sufficient for the problem,
- large bias, large variance

TU Delft

Discussion on Combining

- Base classifiers are trained on systematically different training sets.
- They may be different in importance → weighting is appropriate.

$$S(x) = \text{sign}\left(\sum_i \alpha_i w_i(x)\right)$$

- Weights should follow from the performance → training
- Weights are not optimized for the ensemble of classifiers (avoid overtraining)
- Weights operate on the crisp outputs of wi(x): weighted voting
- Alternative:
  Fisher: soft outputs, optimize weights over ensemble of base classifiers
  Large set of base classifiers may overtrain

TU Delft