Machine Learning for Bioinformatics & Systems Biology

4. Selected topics

Marcel Reinders *Delft University of Technology*
Perry Moerland *Amsterdam UMC, University of Amsterdam*
Lodewyk Wessels *Netherlands Cancer Institute*

Some material courtesy of Robert Duin, David Tax, & Dick de Ridder

Selected topics

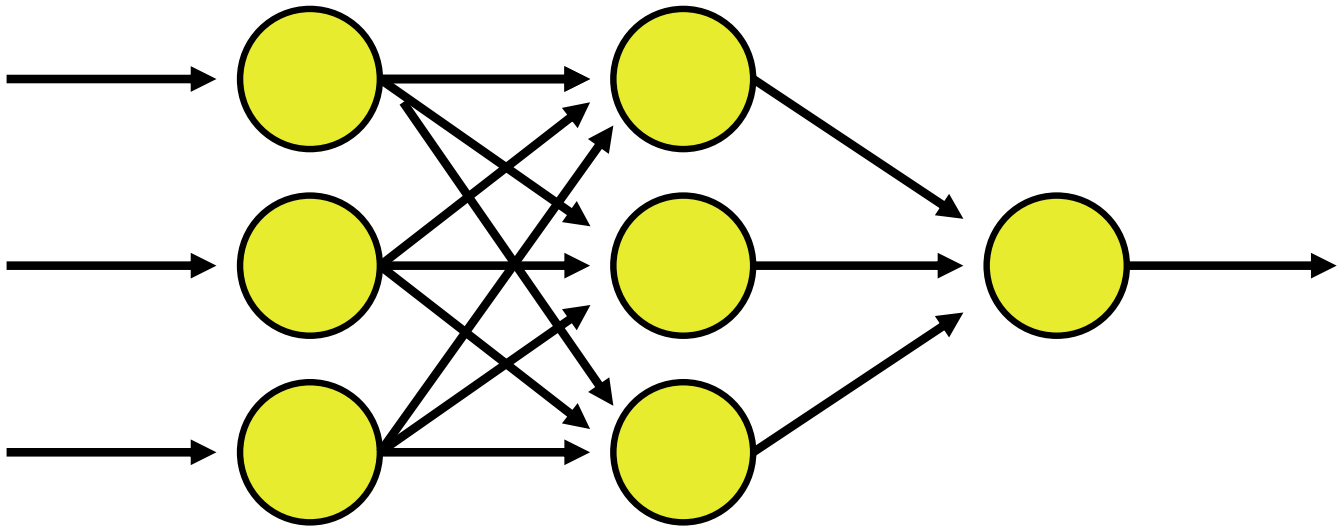
- Famous classifiers
  - Artificial neural networks
  - Support vector classifiers
  - Classifier combination
- The fundamental pattern recognition trade-off
  - Complexity

BioSB

Artificial neural networks

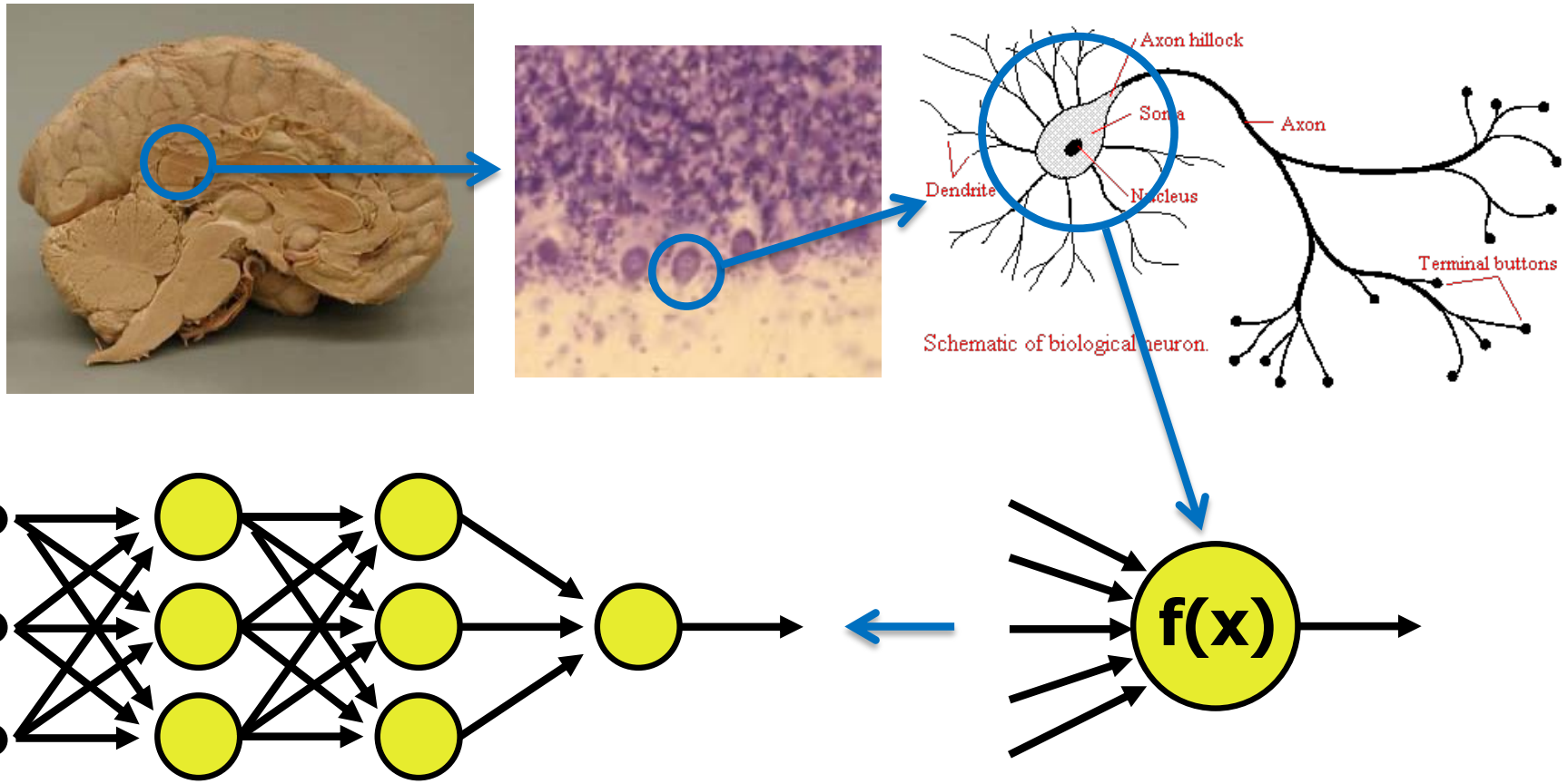
Artificial neural networks (2)

- Large, densely interconnected networks of simple processing units



Artificial neural networks (3)

- Inspired by the brain



Artificial neural networks (4)

- 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
- From 2005/2009 – renewed interest
  - Extension to deep learning (deep nets)
  - Advances in hardware (GPUs) made it possible to learn these networks
  - Major steps in performance improvement (10%)
  - Development of several toolboxes Keras/Tensorflow/Theano/...
  - World attention, also from outside Machine Learning field

made people realize they were doing pattern recognition
Let PR researchers do stuff they never did before (speech recognition)

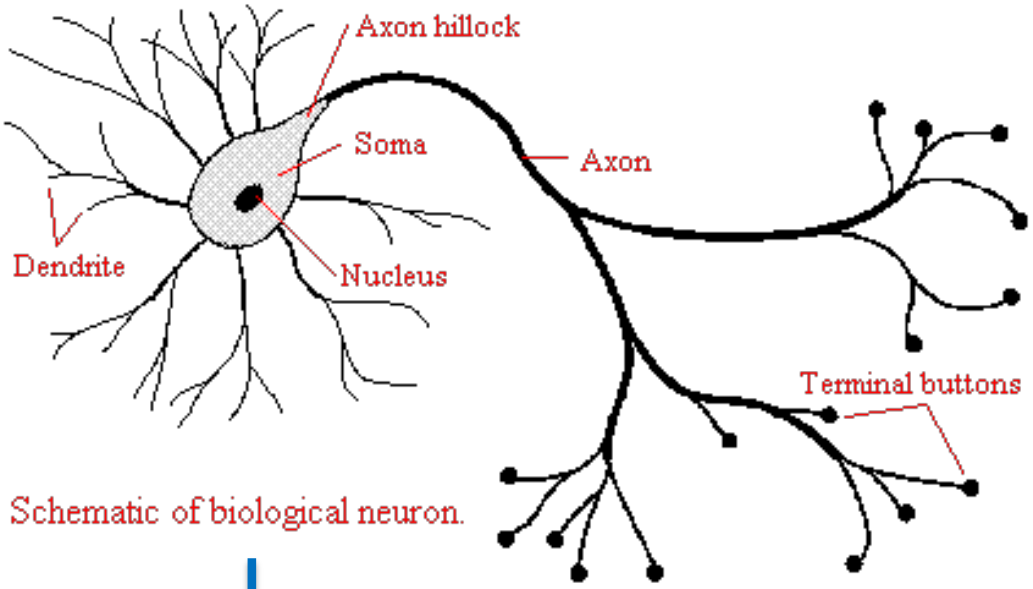
BioSB

History

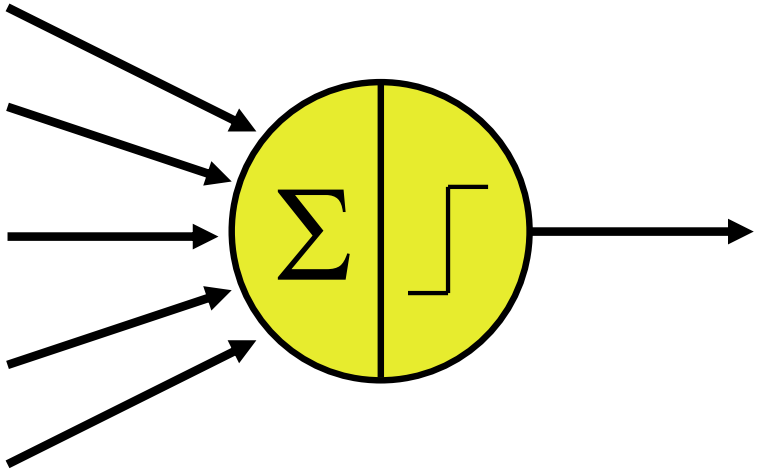
- 1943 : McCulloch and Pitts: model of neuron
- **1958** : Rosenblatt: perceptron
- 1960s : Rosenblatt, Nilsson work on perceptrons
- 1969 : Minsky and Papert point out limitations: perceptrons are linear
- 1982 : Hopfield network (associative memory), Kohonen's self-organising map (clustering), Fukushima's Neocognitron (vision)
- **1986** : Rumelhart, Hinton and Williams: training of nonlinear networks
- 1997 : Hochreiter and Schmidhuber introduce Long Short-term memory (LSTM), recurrent neural net
- **2006** : Hinton showed effective training one-layer at a time
- 2009 : Nvidia involved in "big bang" of "deep learning", 100x time improvement

BioSB

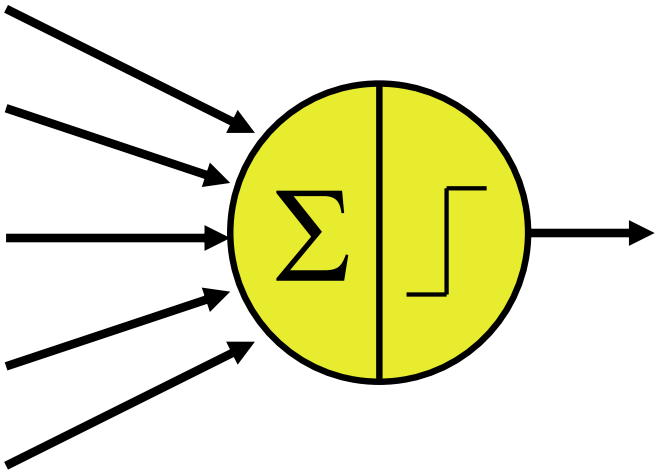
McCulloch-Pitts model (1943)



Schematic of biological neuron.



McCulloch-Pitts model (2)



weights *inputs*

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

threshold or *bias*

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

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

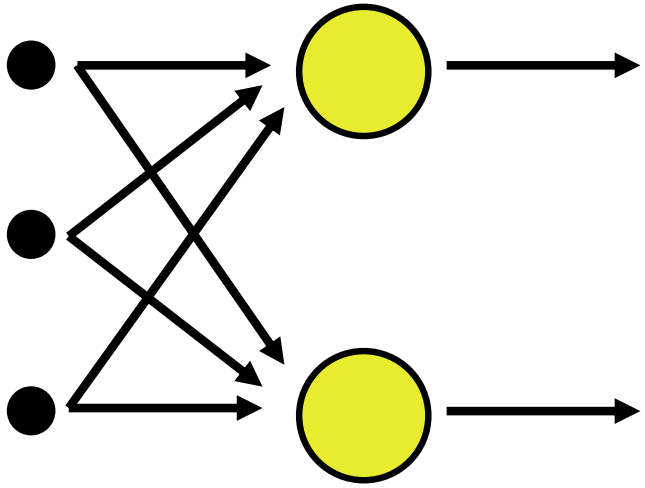
transfer function
or
activation function

"Fire" if total input exceeds a threshold

BioSB

Perceptron

- Networks of McCulloch-Pitts models can perform *universal computation*, given the right weights w: it can do anything a binary computer can do
- ...but how can we find the right weights w ?
- Rosenblatt (1958): possible for single layer networks, *perceptrons*



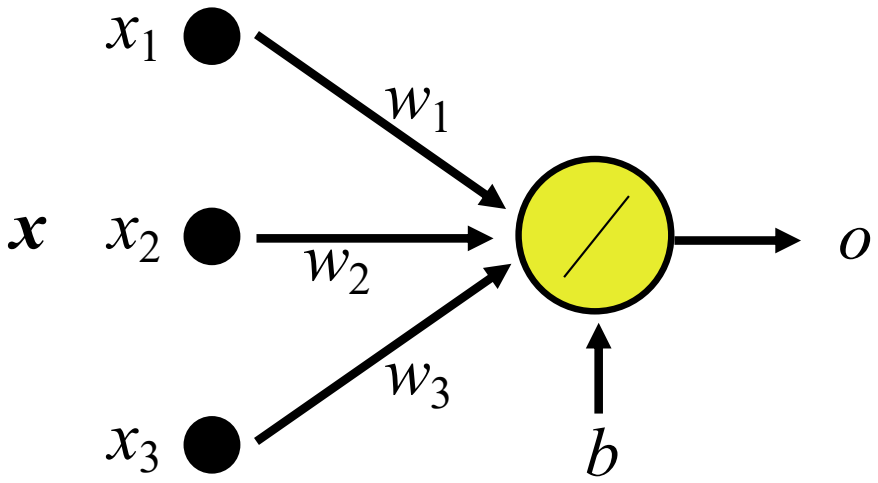
BioSB

Perceptron (2)

- Goal:

$$o(\boldsymbol{x}) = \phi(\boldsymbol{w}^T \boldsymbol{x} + b)$$

$$\begin{cases} > 0 & \boldsymbol{x} \in \omega_1 \\ < 0 & \boldsymbol{x} \in \omega_2 \end{cases}$$



- Trick #1: add bias as weight with constant input

$$\left.\begin{array}{l} \boldsymbol{z} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}, \boldsymbol{v} = \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix} \\ \phi(a) = a \end{array}\right\} \Rightarrow o(\boldsymbol{z}) = \boldsymbol{v}^T \boldsymbol{z}$$

BioSB

Perceptron (3)

- For classification, set targets q for every input vector z:

$$z \in \omega_1 : \quad q = 1$$

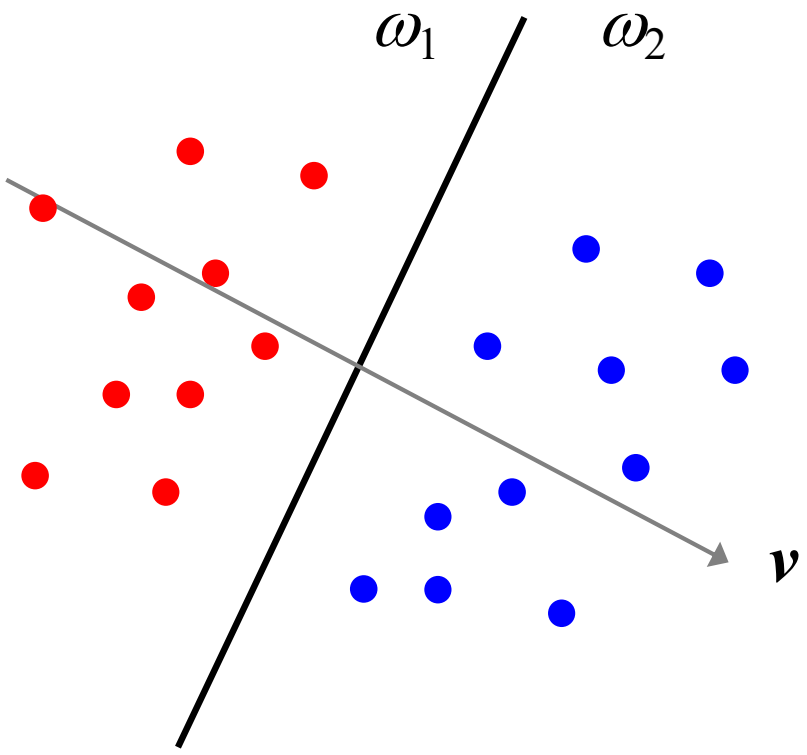
$$z \in \omega_2 : \quad q = -1$$

- Trick #2: use targets to obtain single criterion

$$o(z) = v^T z \begin{cases} > 0 & z \in \omega_1 \\ < 0 & z \in \omega_2 \end{cases}$$

$$\Rightarrow v^T z \cdot q > 0$$

$$\Rightarrow v^T y > 0, \quad y = z \cdot q$$



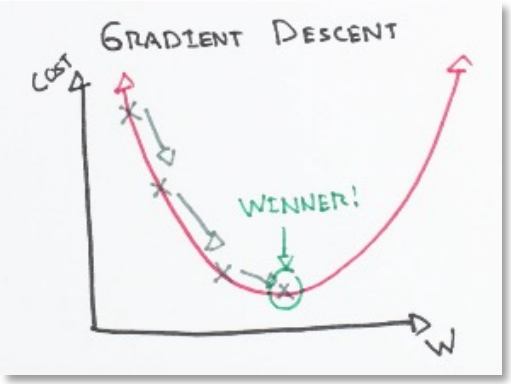
BioSB

Perceptron (4)

- Goal: zero misclassifications, i.e. $\boldsymbol{v}^T \boldsymbol{y}_i > 0 \qquad \forall i$

- Criterion to minimize: $J(\boldsymbol{v}) = \sum_{\boldsymbol{y}_i \in \mathcal{Y}} \left(-\boldsymbol{v}^T \boldsymbol{y}_i\right)$

  where $\mathcal{Y}$ is the set of misclassified samples



- Can use gradient descent: $\partial J(\boldsymbol{v}) / \partial \boldsymbol{v} = \sum_{\boldsymbol{y}_i \in \mathcal{Y}} \left(-\boldsymbol{y}_i\right)$

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k - \rho \frac{J(\boldsymbol{v})}{d\boldsymbol{v}} = \begin{cases} \boldsymbol{v}^k + \rho \sum_{\boldsymbol{y}_i \in \mathcal{Y}} \boldsymbol{y}_i & \textit{batch update} \\ \boldsymbol{v}^k + \rho \boldsymbol{y}_i, \quad \boldsymbol{y}_i \in \mathcal{Y} & \textit{single update} \end{cases}$$

Criterion is somewhat arbitrary, could also count misclassifications

Perceptron (5)

- Perceptron is a trainable two-class linear discriminant (extendable to multiple classes)

- Training algorithm can be proven to converge to correct solution for separable classes

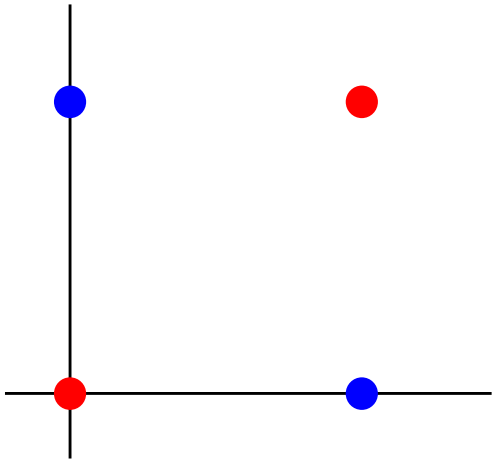
- When classes are not linearly separable:
  - indefinite training, weights will blow up
  - solution: decrease ρ during training, ρ(k), or early stopping

BioSB

Perceptron (6)

- 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!

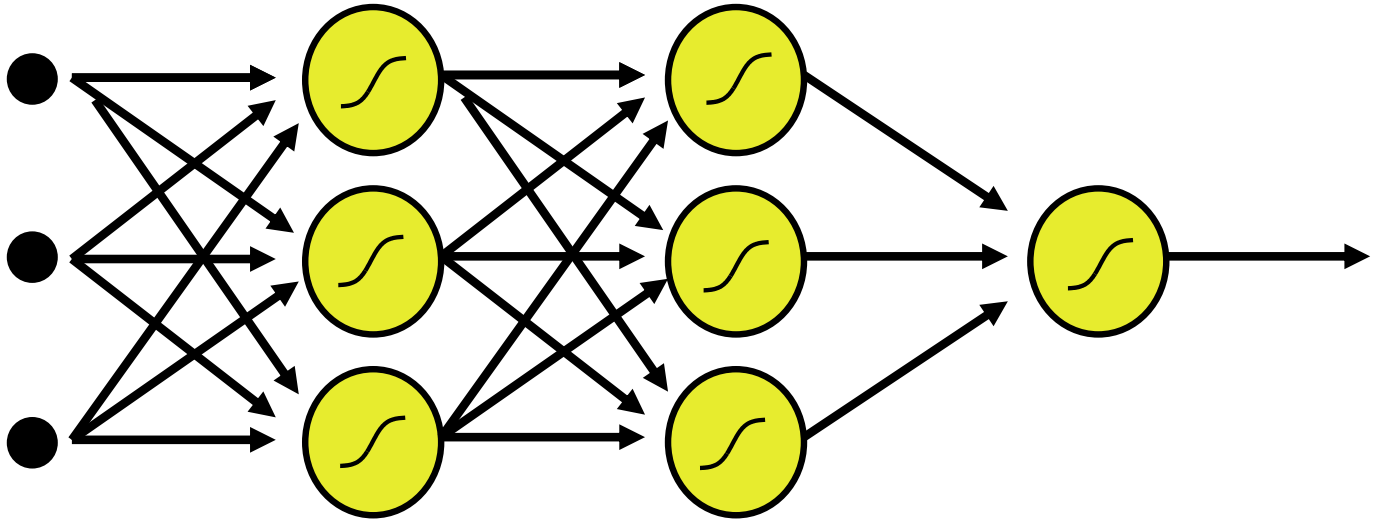
This did in fact take twenty years…

BioSB

Multilayer perceptron (MLP)

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

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



BioSB

Backpropagation training

- Method to distribute weight updates through the network
- Criterion: error E, difference between network output and targets (mean square error between output and target $\sum(e_i - o_i)^2$)
- Initialize weights w to small random values
- While not converged, e.g. while $|E^{old} - E|/E > E_{thr} = 10^{-6}$, or while error on validation set decreases:
  - select a training sample $\boldsymbol{x}_i$
  - for each weight w
    - calculate $\partial E / \partial w$
    - set $w' = w - \rho \, \partial E / \partial w$

      (with ρ a learning rate, e.g. 0.01)
    - or use a momentum term,

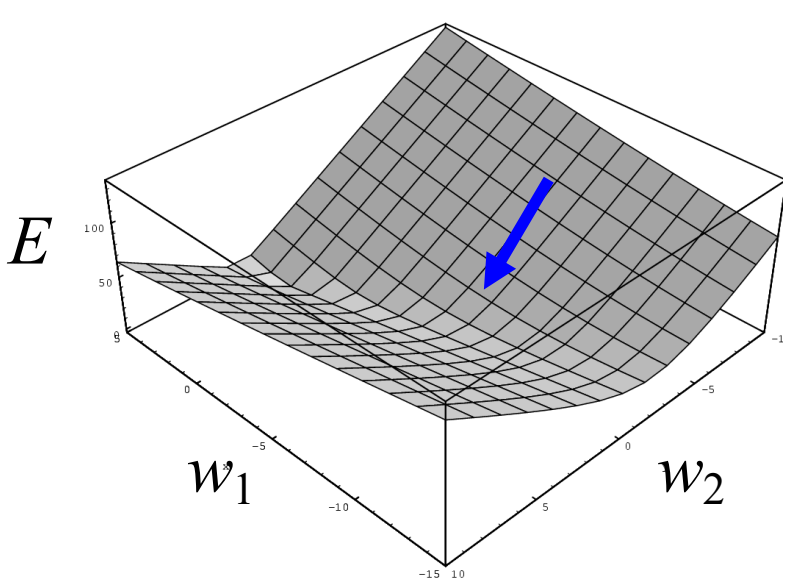
      $w' = w - \rho \, \partial E / \partial w - \alpha \left[\partial E / \partial w\right]^{prev}$

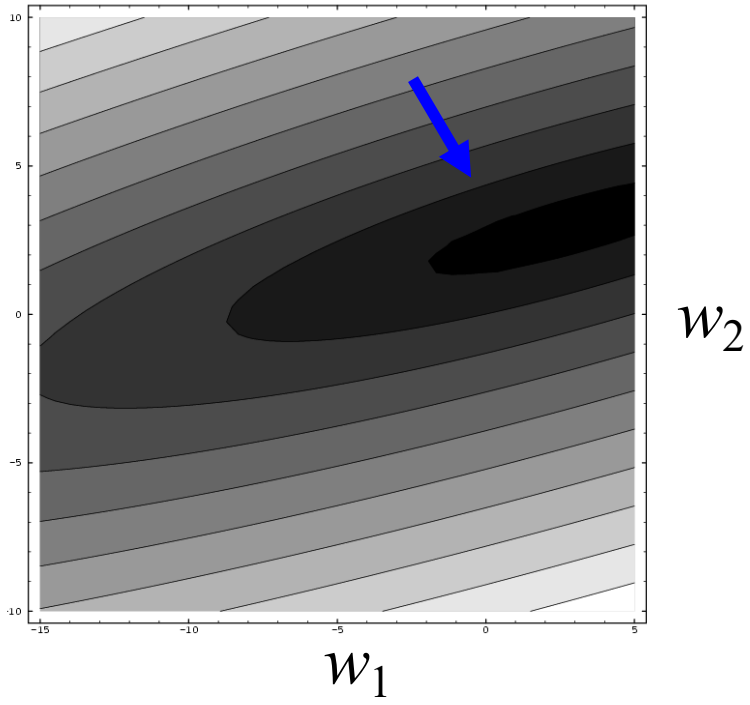
$\boldsymbol{\alpha} >> \boldsymbol{\rho}$: keep moving in previous direction
$\boldsymbol{\rho} >> \boldsymbol{\alpha}$: adapt to new direction

BioSB

Backpropagation training (2)

- Example: two weights

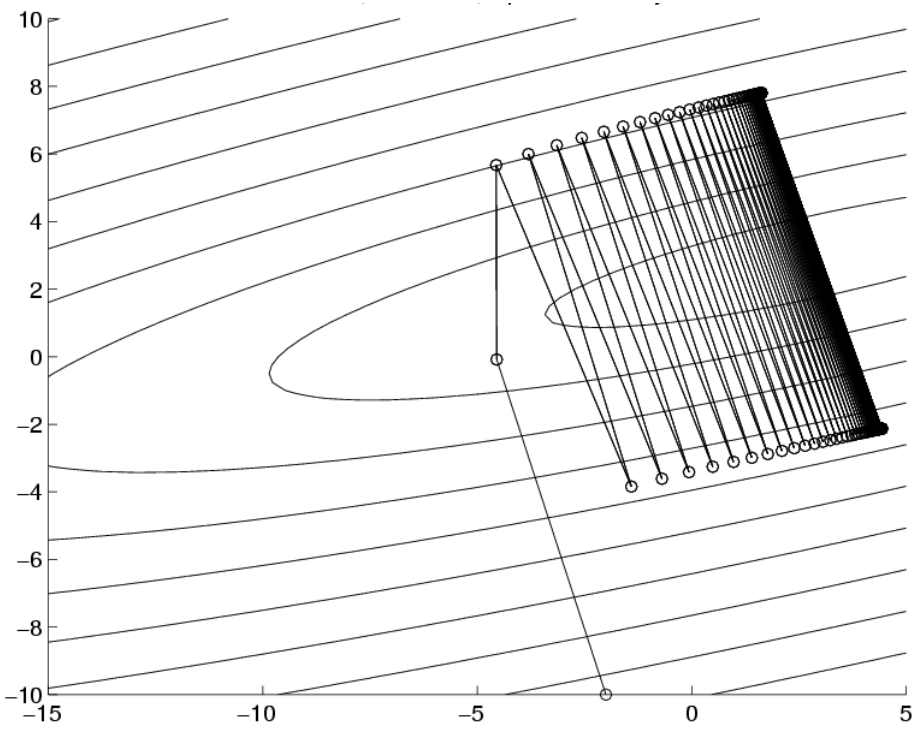


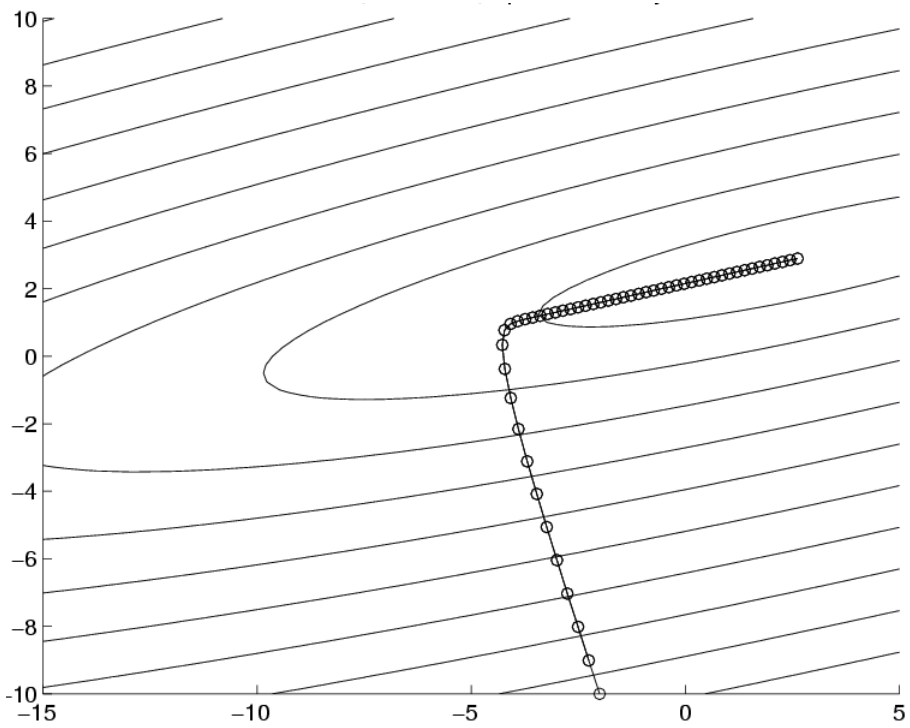


BioSB

Backpropagation training (3)

- Learning rate controls oscillation and speed





$\rho = 1$: >100 iterations

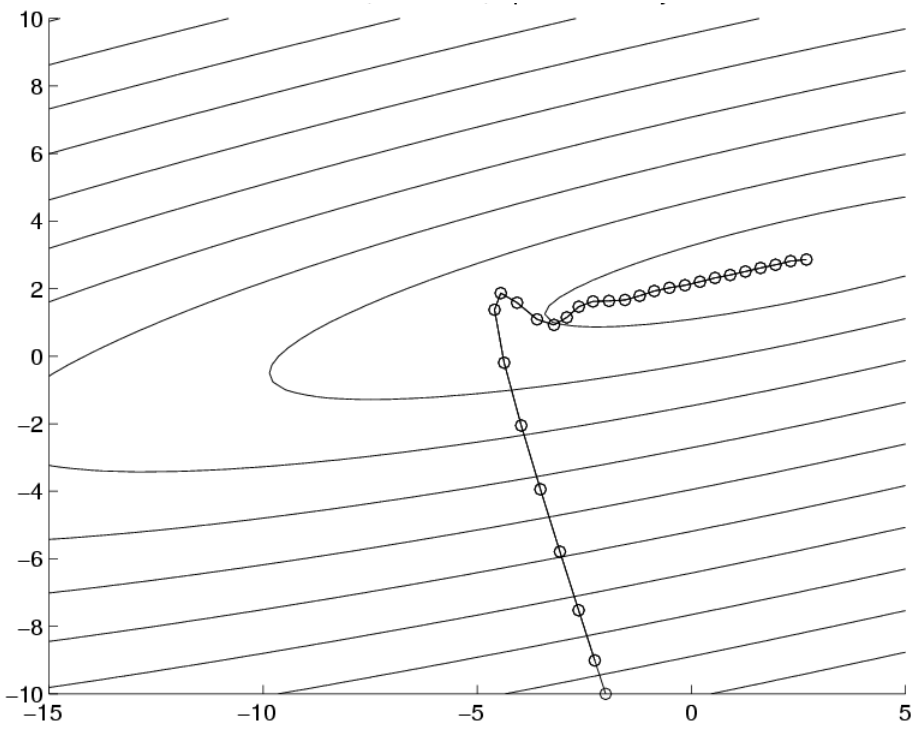
$\rho = 0.1$: 52 iterations

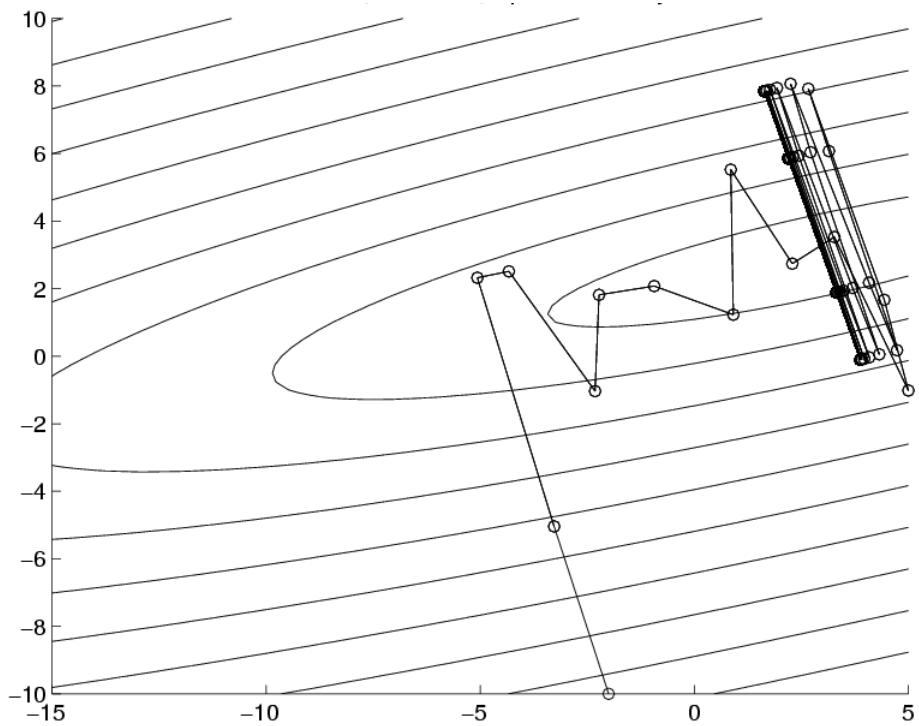
In practice, not easy
(imagine doing this for thousands of weights)

BioSB

Backpropagation training (4)

- Momentum uses a bit of the previous step



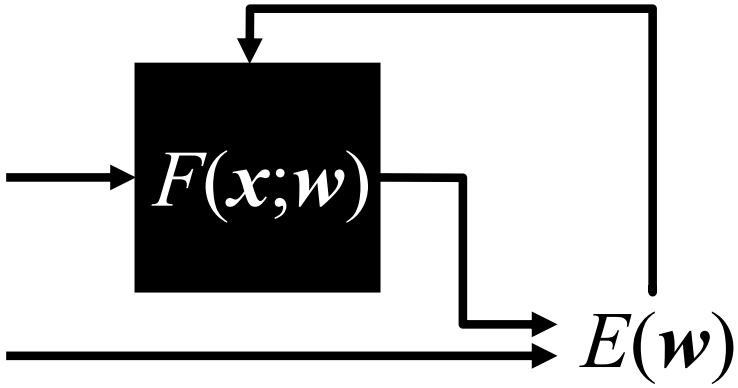


$\rho = 0.1, \alpha = 0.5$: 29 iterations $\rho = 0.5, \alpha = 0.5$: >100 iterations

Right: learning rate to large , so oscillations start occurring again …
Also option to make learning rate dependent on time : $\rho(t)$

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 optimize MSE E w.r.t. weight vector $\boldsymbol{w}$ using any optimization routine, e.g.
  - second order (Newton, pseudo-Newton)
  - conjugate gradient descent
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
  - Levenberg-Marquardt (LM, in **`PRTools`**)

BioSB

Multilayer perceptrons

- 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!

*"Training ANNs is more of an **ART** than a science'*

BioSB

Multilayer perceptrons (2)

- 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

  Per node: #parents+bias node (n_l + 1)

- Danger of overtraining!

- Prevention:
  - use small networks
  - regularize: minimize $E(\boldsymbol{w})+\lambda\|\boldsymbol{w}\|$
  - small w's: low complexity, training slowly increases w's;
    so when stopping in time: automatic regularization!
- Regularization is a form of complexity control (discussed later)

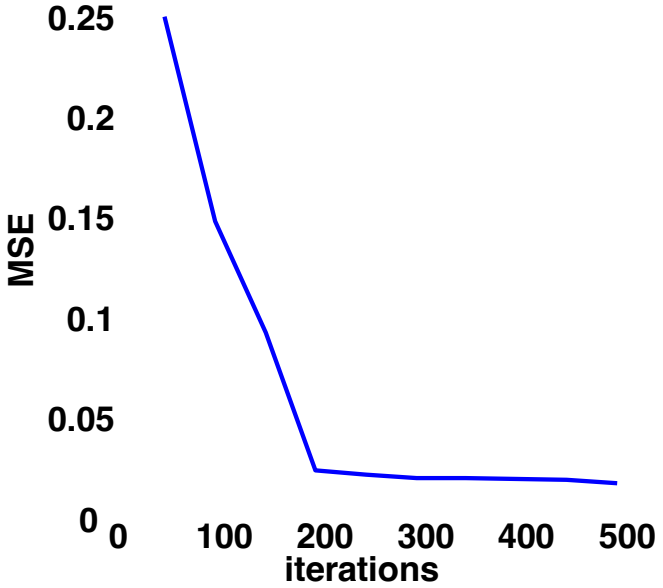
BioSB

Multilayer perceptrons (3)

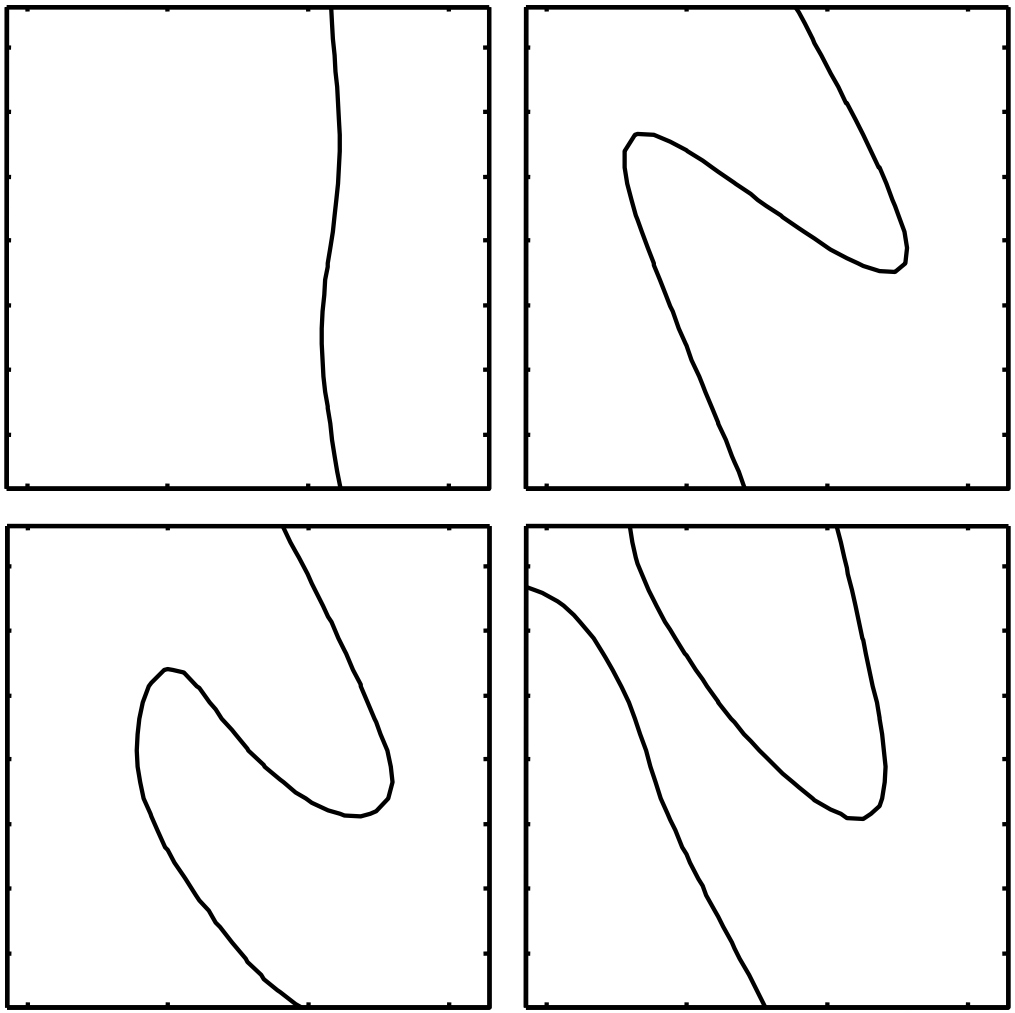
Initialization still important

- Examples:

1 hidden layer
of 3 units,
2 initialisations



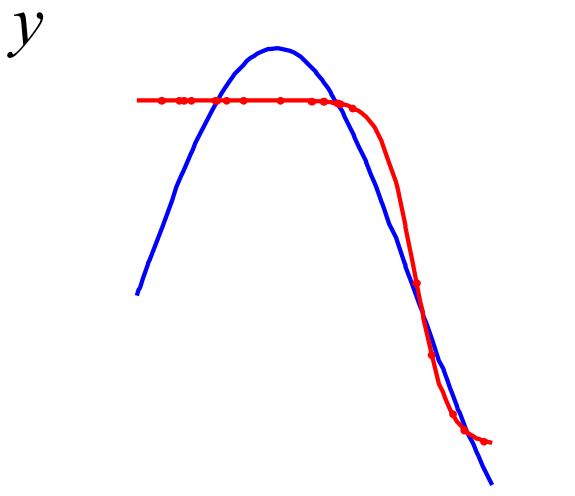
2 hidden layers
of 5 units each,
2 initialisations



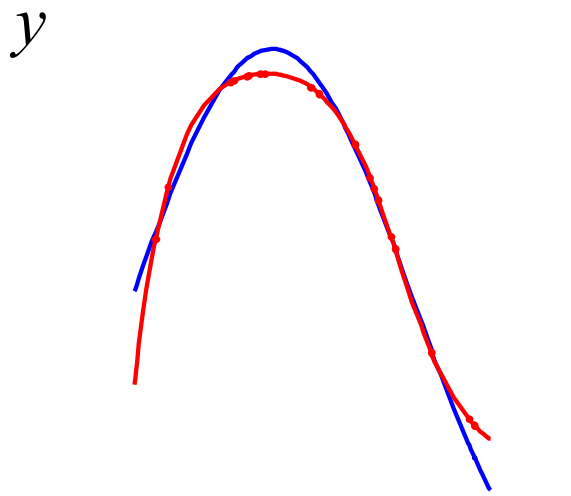
BioSB

ANNs for regression

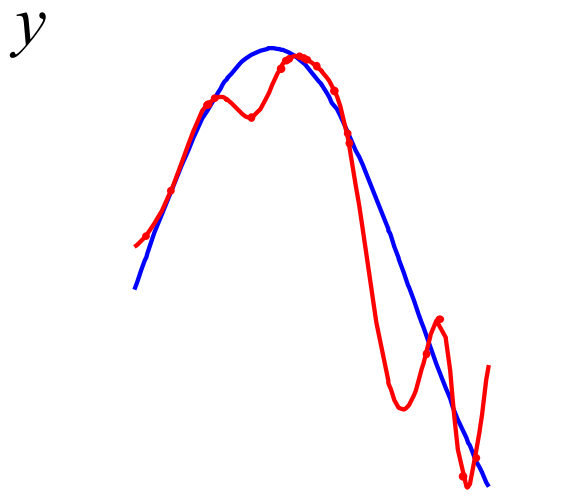
- Feedforward ANNs are *universal approximators*
  - Classification: input x, targets y = 0/1, 0.1/0.9
  - Regression: input x, output y
- Examples:



1 hidden unit (sigmoid)



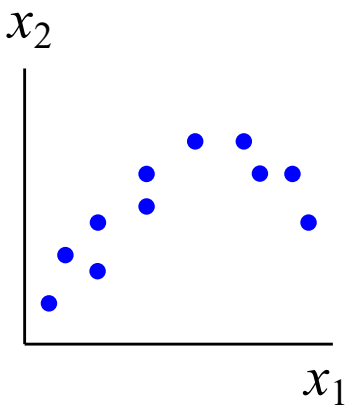
2 hidden units

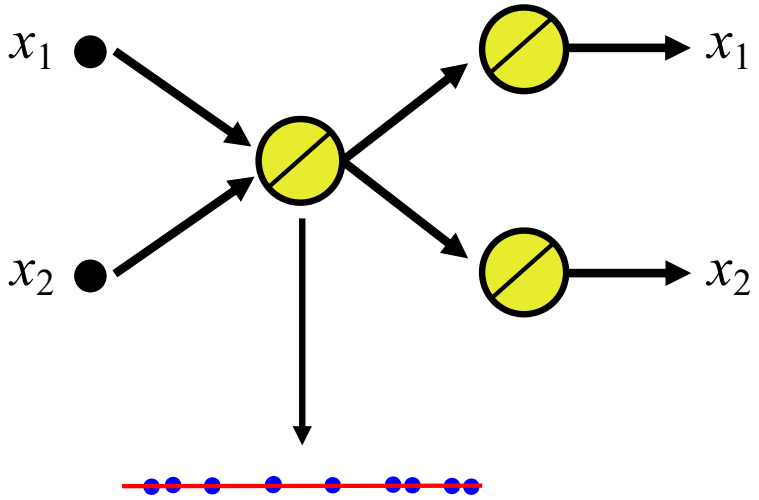


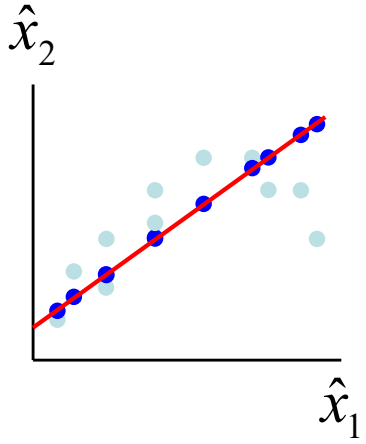
2x5 hidden units

Autoregressive ANNs / Autoencoder

- Feedforward ANNs that predict their input
- Bottleneck layer: feature extraction





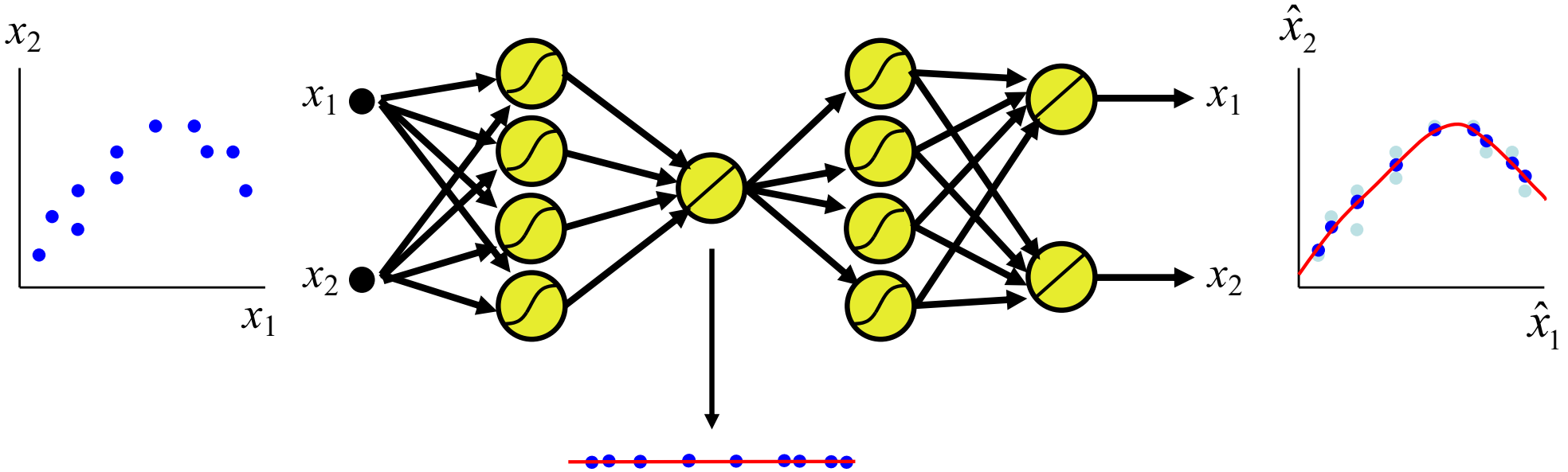


If linear (as in this example) : then we are performing PCA !!!

BioSB

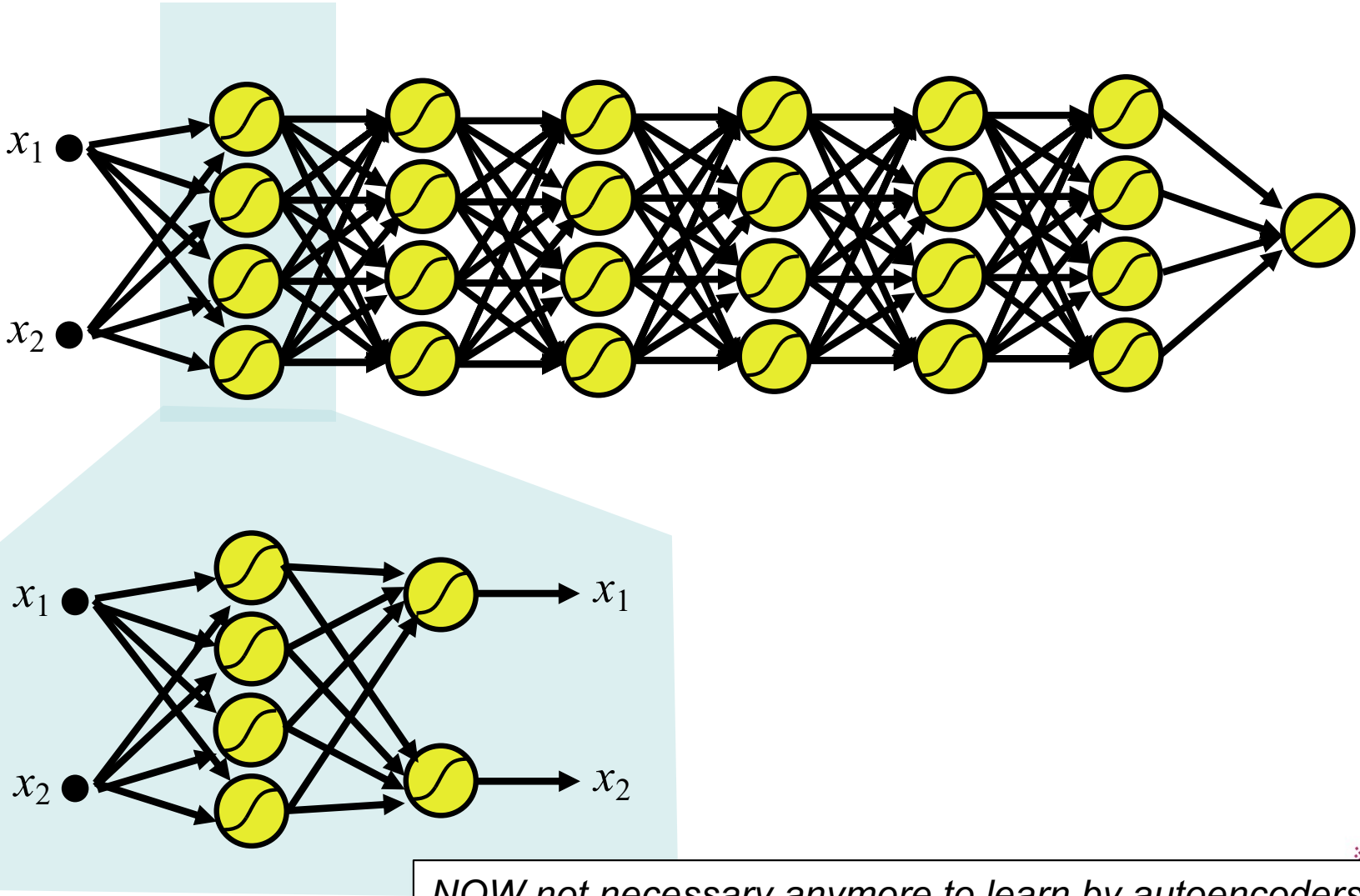
Autoregressive ANNs / Autoencoder (2)

- With multiple hidden layers:
  nonlinear feature extraction



Deep learning

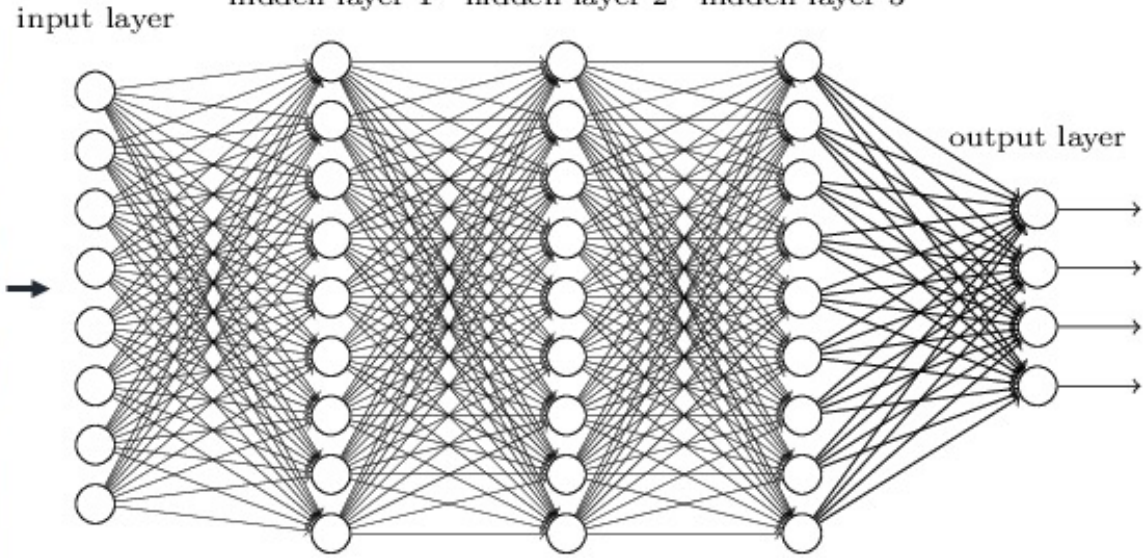
Many hidden layers, learn by auto-encoding



NOW not necessary anymore to learn by autoencoders
With GPUs you can use Backpropagation again (fast enough)

Deep learning
Learning features

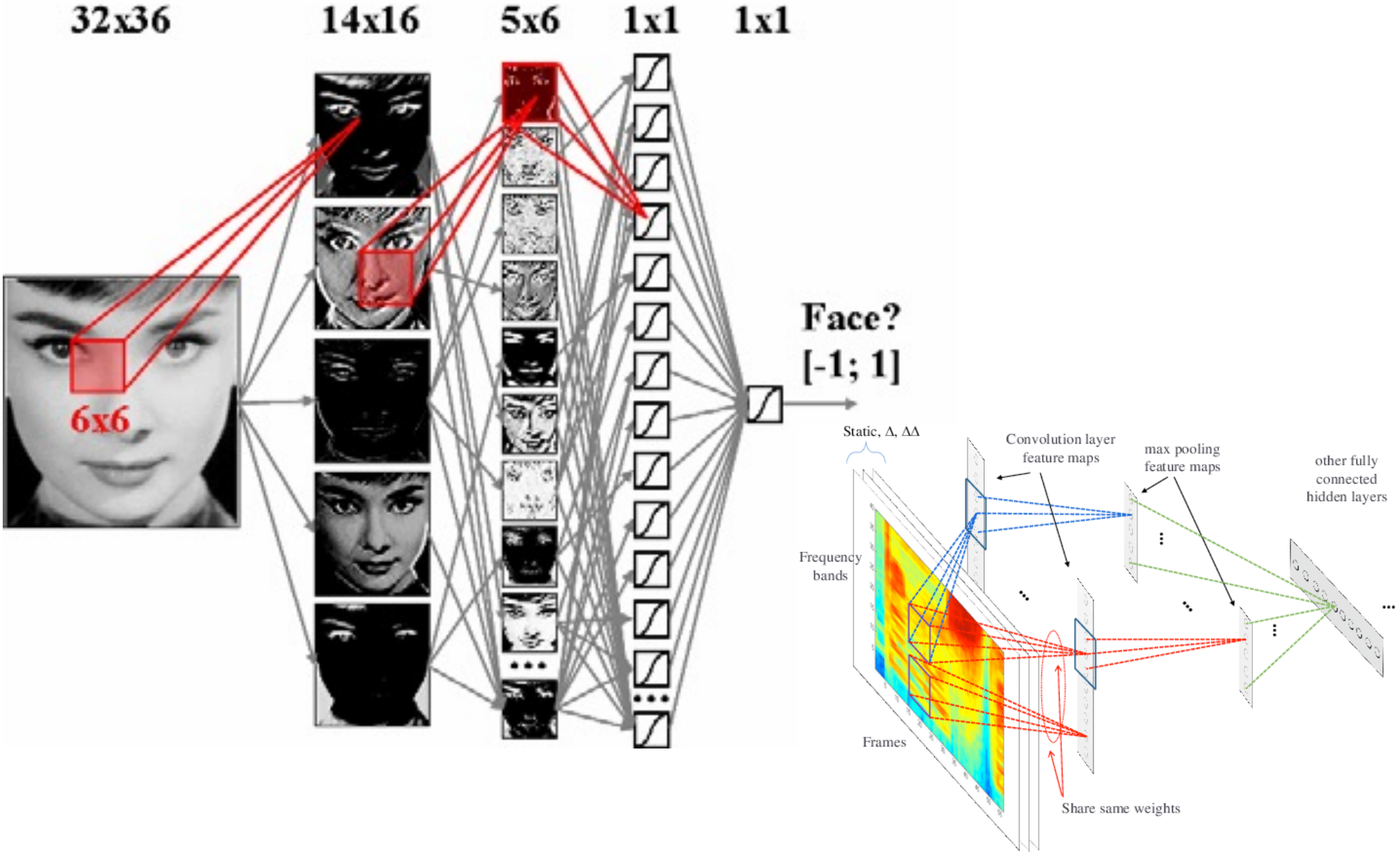
Deep neural networks learn hierarchical feature representations



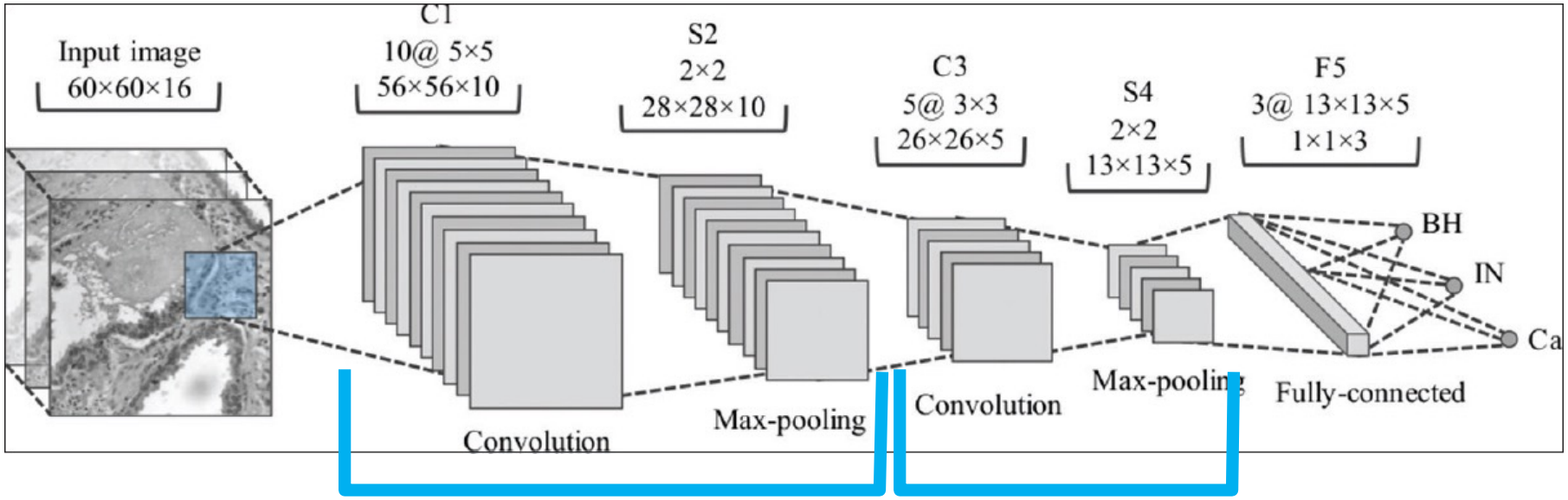
BioSB

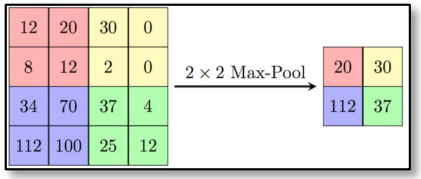
Deep learning

Convolutional Neural Networks (1)



Deep learning
Convolutional Neural Networks (2)





- amount of layers
- use of pre-trained networks (on another problem)

BioSB

Deep learning
Convolutional Neural Networks (3)

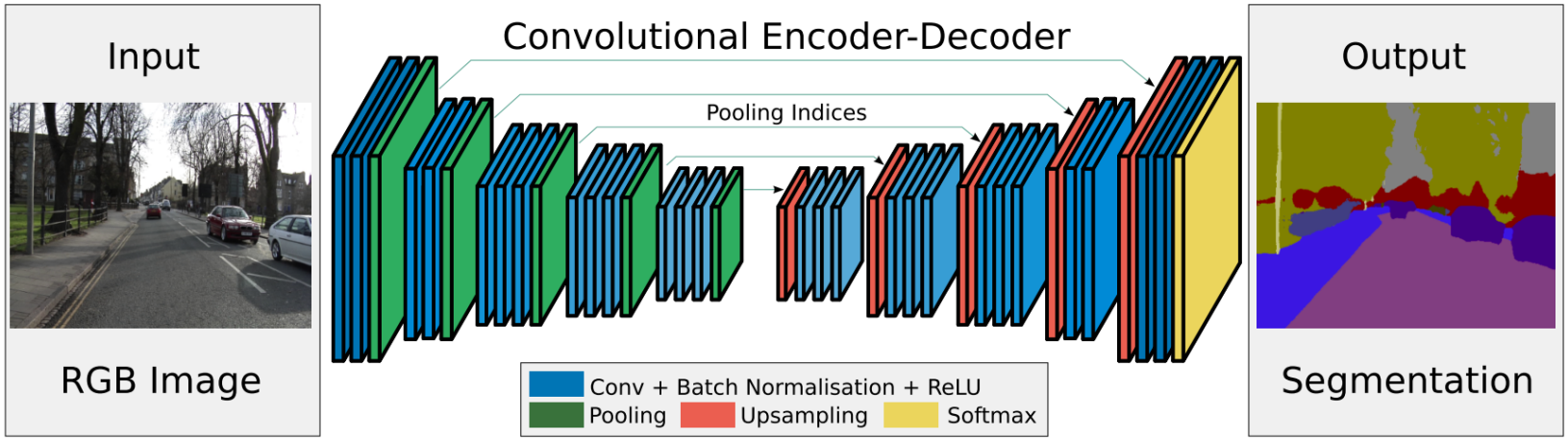
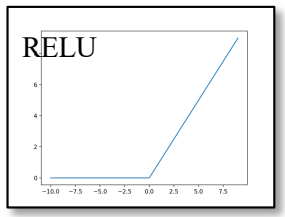
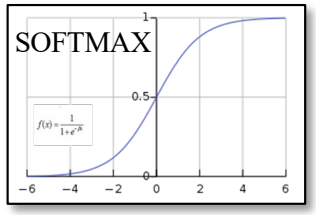


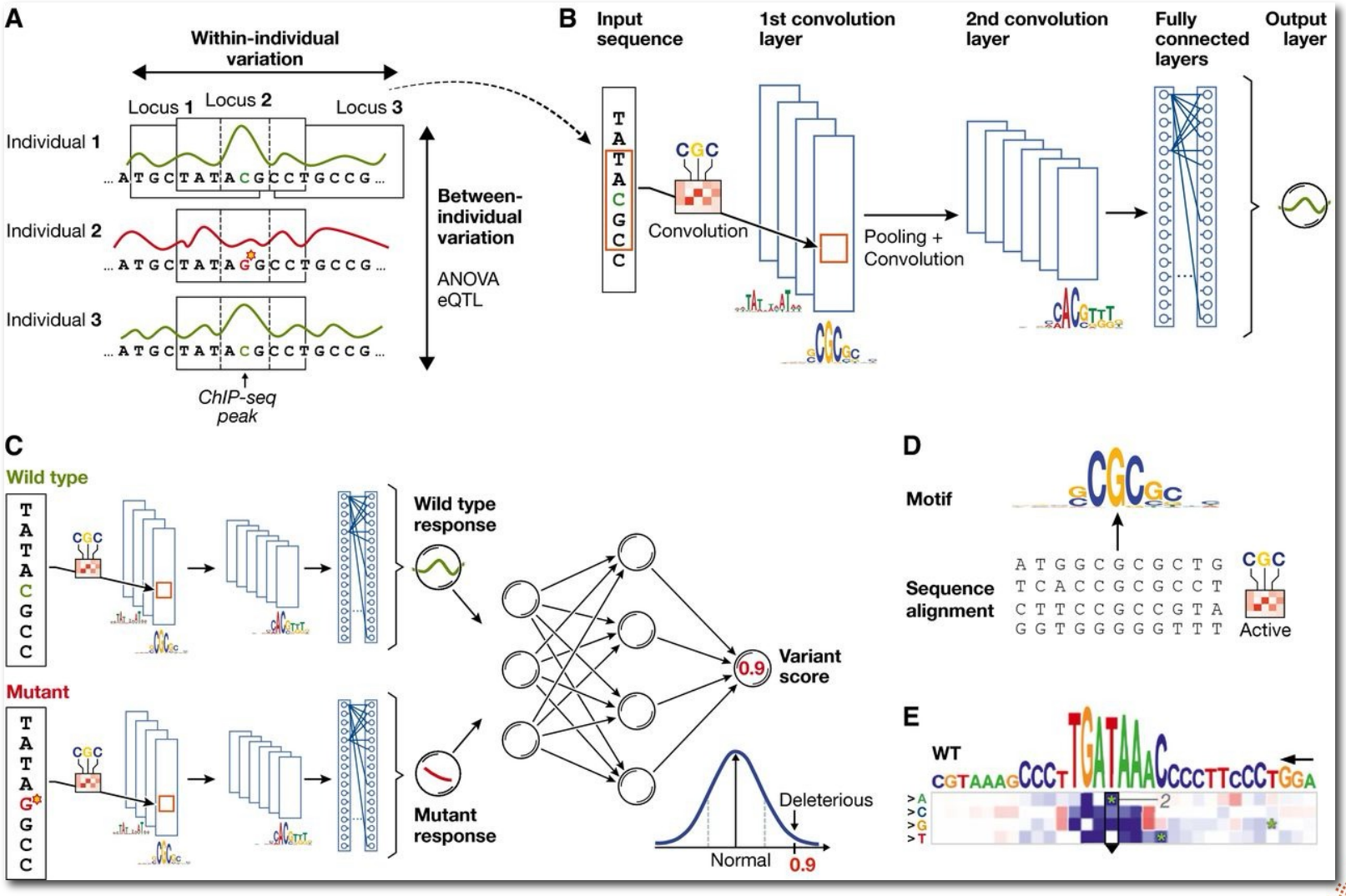
Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.





BioSB

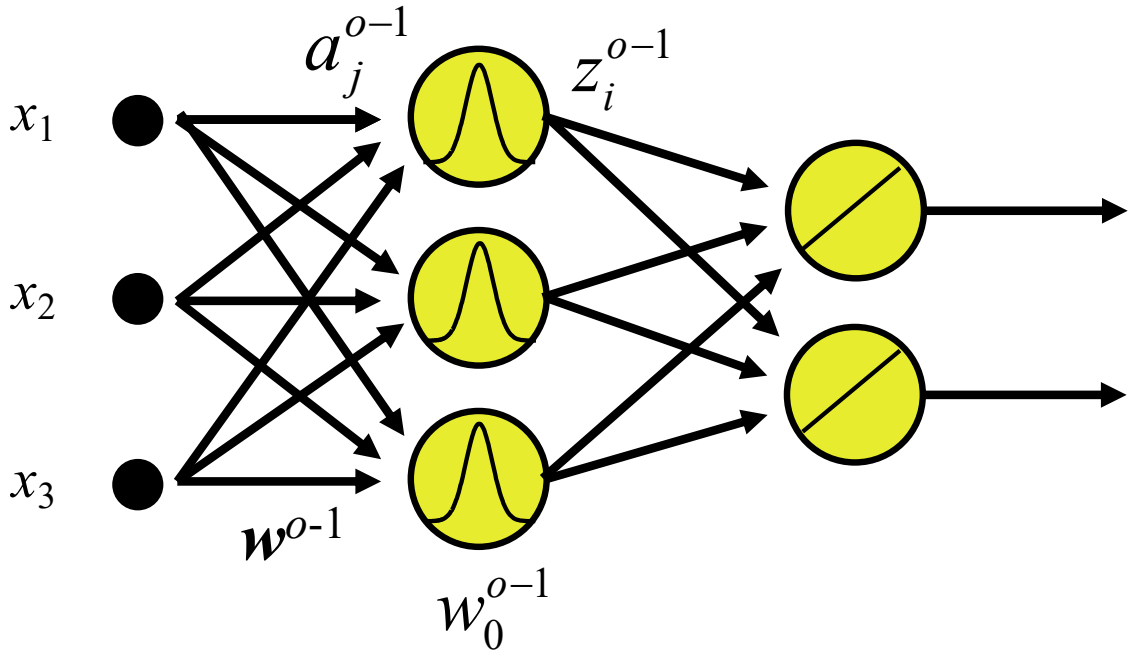
Deep learning
Convolutional Neural Networks (4)



Christof Angermueller et al. Mol Syst Biol 2016;12:878

Radial basis function ANNs

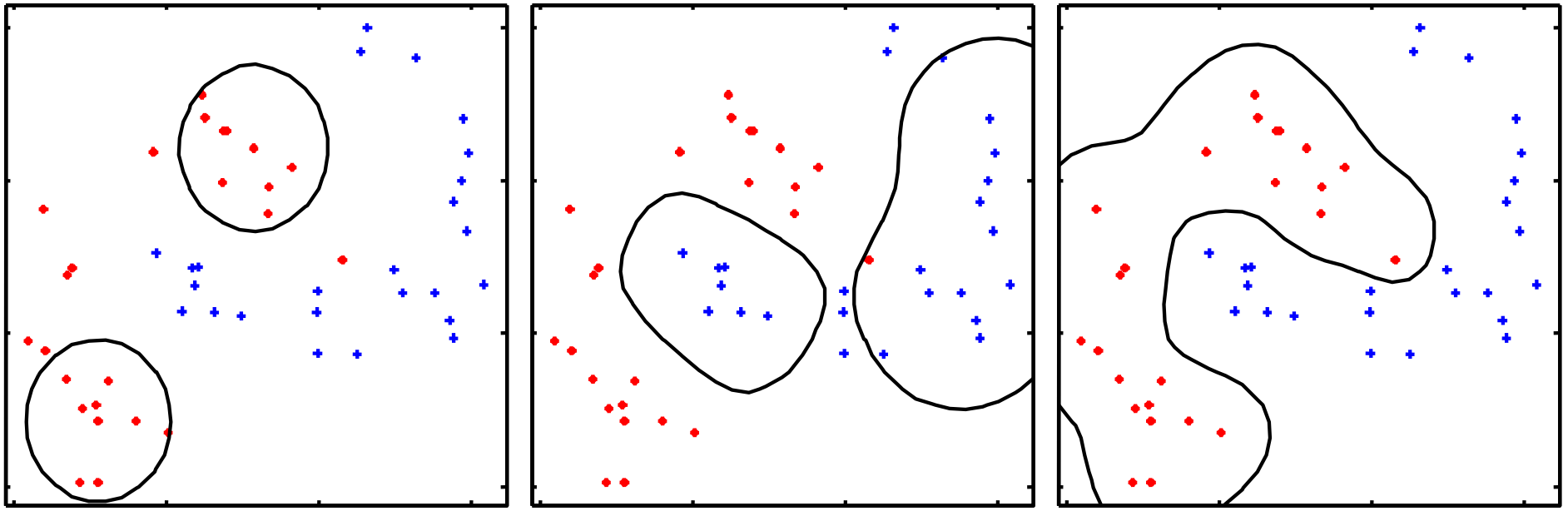
- Feed-forward ANNs with
  - Squared distance activation functions $a_j^{o-1} = \left\| \boldsymbol{x} - \boldsymbol{w}^{o-1} \right\|^2$
  - Gaussian transfer functions $z_j^{o-1} = N(\mu = a_j^{o-1}, \sigma^2 = w_0^{o-1})$



BioSB

Radial basis function ANNs (2)

- Example: classification



2 hidden units | 5 hidden units | 10 hidden units

BioSB

Other types of ANN

- Large number of feedforward variants
  - cascading correlation (self-constructing)
  - Neocognitron (for vision)
  - time-delay (for speech and image analysis)
  - ...
- Self-organising maps and GTMs:
  - feature extraction, clustering
- Hopfield networks:
  - associative memories, optimisation
- Boltzmann machines, Bayesian networks:
  - conditional probability models

BioSB

Recapitulation

- *Perceptrons* are “neuron-inspired” linear discriminants
- *Multilayer perceptrons* and *radial basis function* feedforward ANNs are trainable, nonlinear discriminants
- Feed-forward ANNs in general can be used for classification, regression and feature extraction
- There is a large body of alternative ANNs
- Key problems in the application of ANNs are choosing the right *architecture* and good *training parameters*

BioSB

10 min break

Support vector classifiers

Vapnik

- Performed foundational work in pattern recognition
  with Chervonenkis in Russia from the 1960s

- Motto:

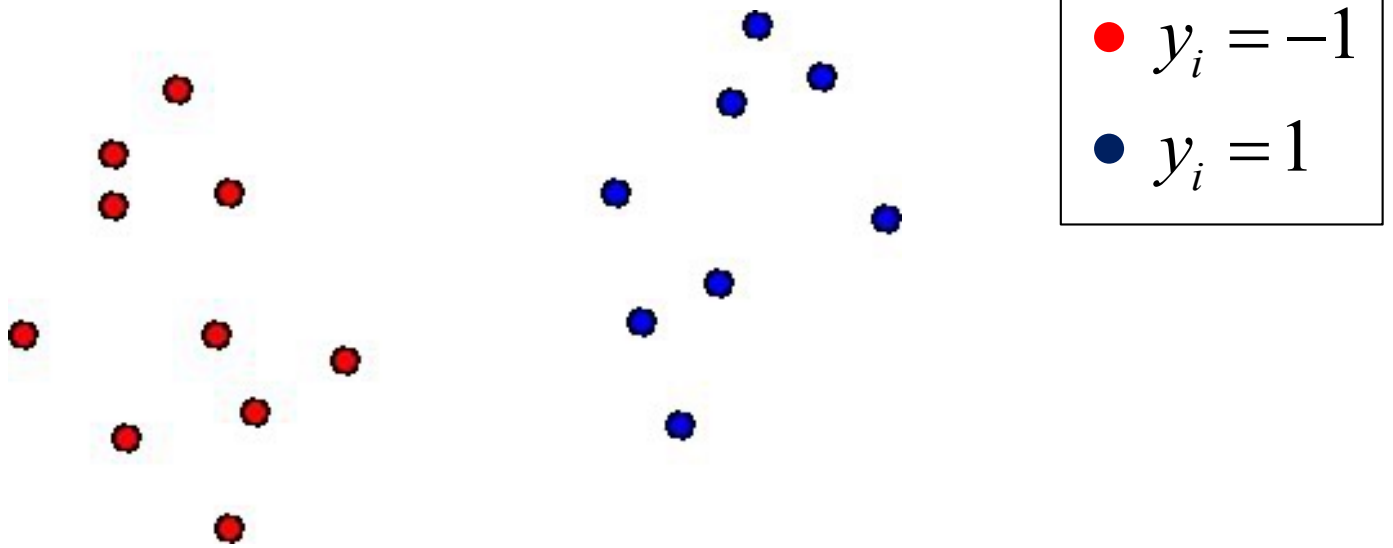
  When you have limited training data,
  and you want to solve a classification problem,
  avoid solving a more complicated intermediate problem

- Translation to classification:
  when you want to find a discriminant, avoid estimating densities

BioSB

Maximum margin classifier

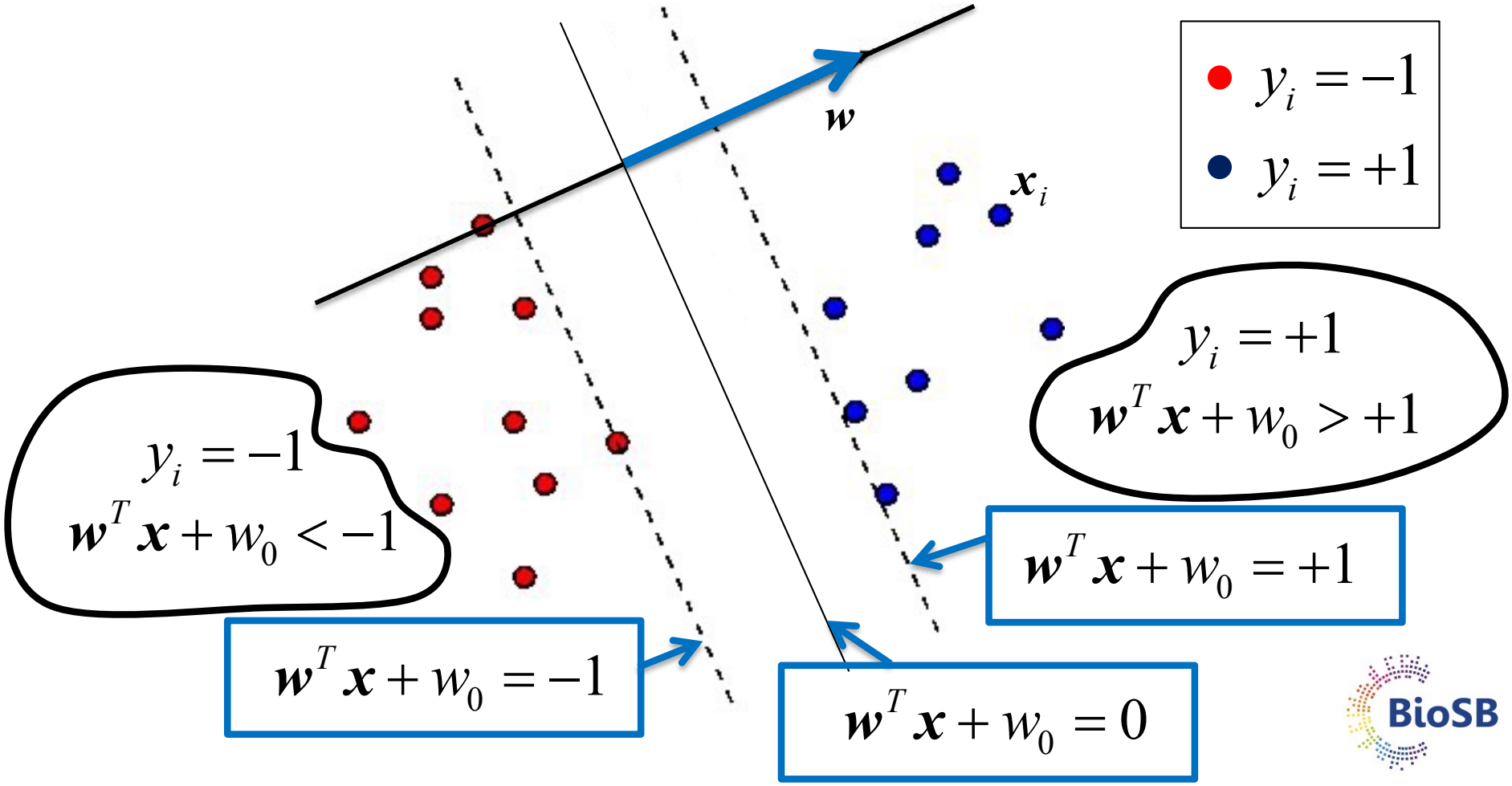
- Simple problem: 2 linearly separable classes
  - What is a good linear classifier?
  - What is the best linear classifier?



BioSB

Maximum margin classifier (2)

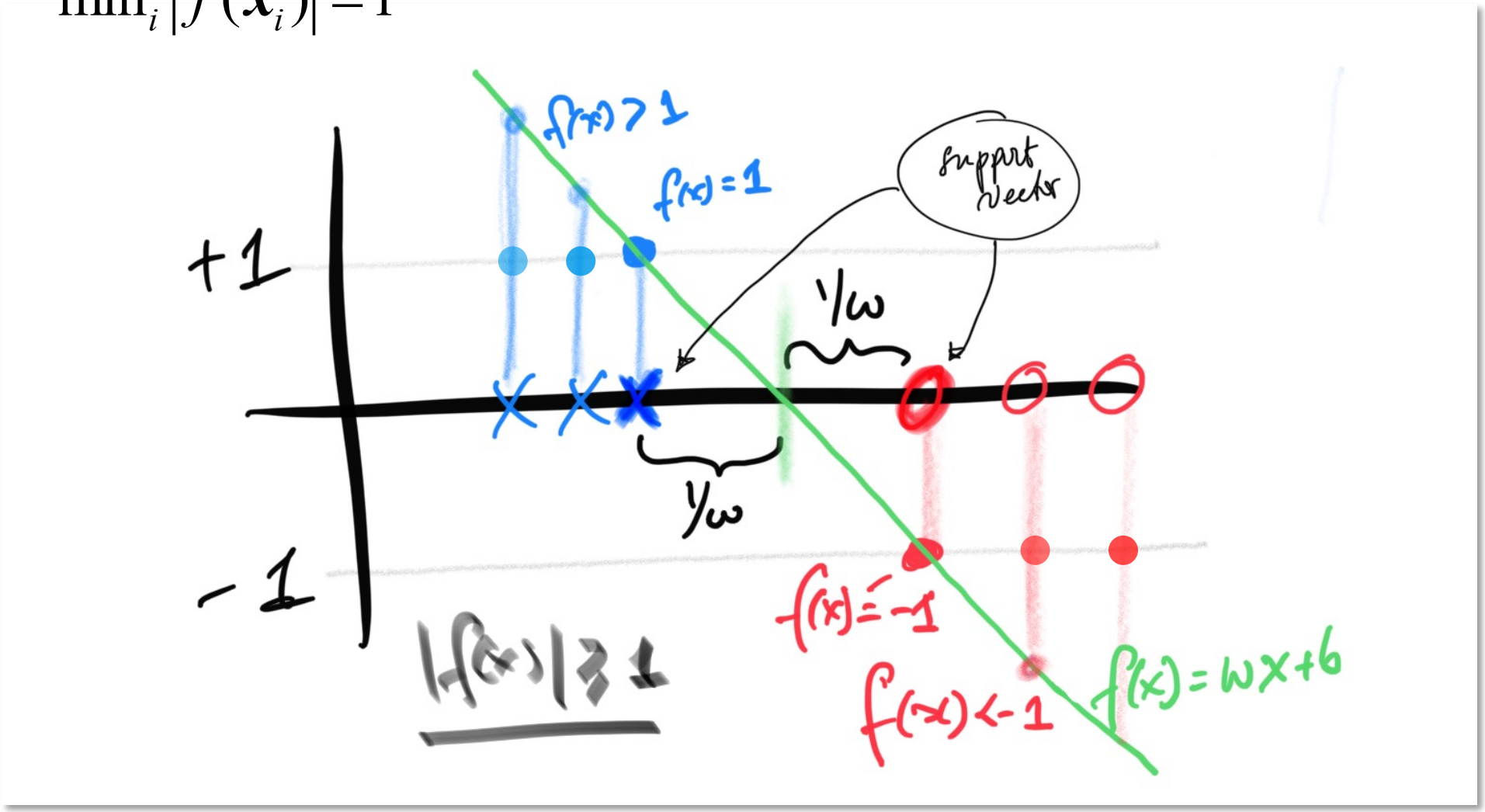
- Canonical hyperplane:
  any plane of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$
  for which $\min_i |f(\boldsymbol{x}_i)| = 1$



BioSB

Maximum margin for 1D data

$$\min_i \left| f(\boldsymbol{x}_i) \right| = 1$$



Maximum margin classifier (3)

- The distance between an object $\boldsymbol{x}_i$ and the hyperplane is

$$d(\boldsymbol{x}_i, \text{decision boundary}) = \frac{\boldsymbol{w}^T \boldsymbol{x}_i + w_0}{\|\boldsymbol{w}\|}$$

- The maximum margin classifier is a canonical hyperplane s.t. the distance between the object closest to the hyperplane on one side,

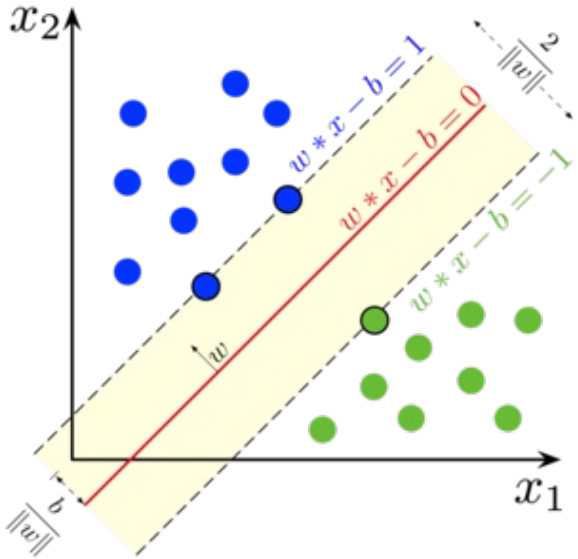
$$\arg\min_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) \quad | \, y_i = +1$$

and the object closest on the other side,

$$\arg\max_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) \quad | \, y_i = -1$$

is maximal

- This distance is called the margin: $\rho = \frac{2}{\|\boldsymbol{w}\|}$



BioSB

Support vector classifier

- Maximizing the margin $\rho = \dfrac{2}{\|\boldsymbol{w}\|}$

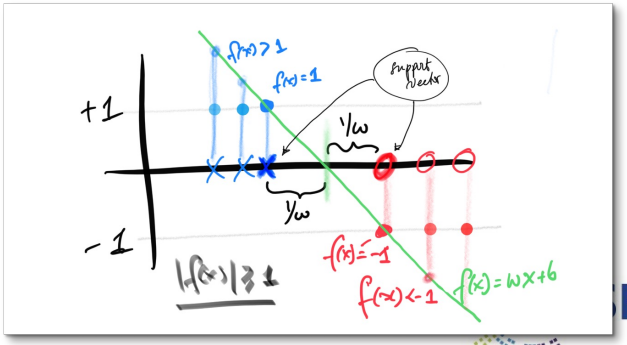
  under the constraint that all training samples are classified correctly, leads to the optimization problem:

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ such that}$$

$$\boldsymbol{w}^T \boldsymbol{x}_i + w_0 \leq -1 \,|\, y_i = -1$$

$$\boldsymbol{w}^T \boldsymbol{x}_i + w_0 \geq +1 \,|\, y_i = +1$$

- The constraints can be written as $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) > 1$
- This is called the *support vector classifier*, or *support vector machine* (SVM)



Support vector classifier (2)

- It is possible to incorporate the constraints into the optimization itself, using Lagrange multipliers (basic calculus):

$$\max_{\alpha} \min_{\mathbf{w},w_0} \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1 \right)$$

$$\text{with } \alpha_i > 0 \quad \forall i$$

  - Each constraint corresponds to a single object $\boldsymbol{x}_i$
  - Each constraint has a Lagrange multiplier α_i
  - So each object corresponds to a Lagrange multiplier

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ such that}$$

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) > 1$$

BioSB

Support vector classifier (3)

- To solve the optimization, take the derivative and set to 0
  - Differentiate with respect to $\boldsymbol{w}, w_0$:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \qquad (w_0)$$

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \qquad (\boldsymbol{w})$$

  - Re-substituting gives:

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$\text{with } \alpha_i > 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Max over $\boldsymbol{\alpha}$, derivatives wrt $\boldsymbol{\alpha}$

$$\max_{\alpha} \min_{\mathbf{w}, w_0} \frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i (\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1 \right), \quad \alpha_i > 0$$

BioSB

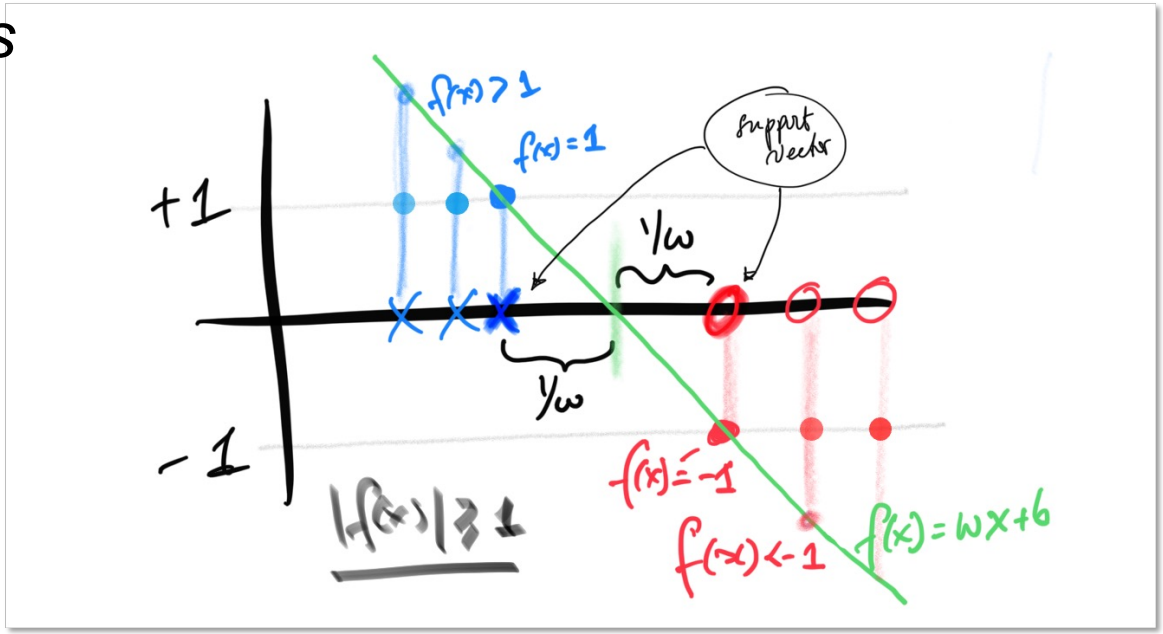
Support vectors

- The classifier is a linear combination of objects:

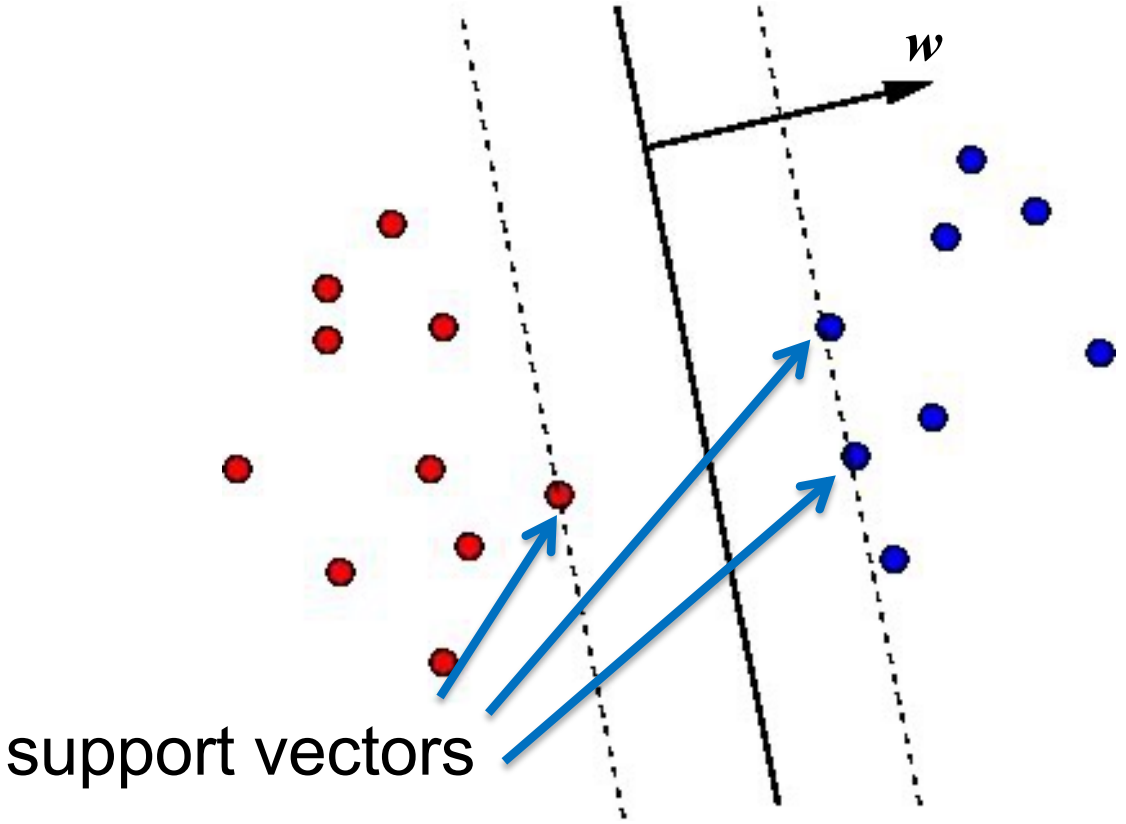
$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$$

- Many Lagrange multipliers become equal to 0, so in fact the classifier is a *sparse* linear combination of objects

- Objects for which the Lagrange multiplier > 0 are called *support vectors*



Support vectors (2)



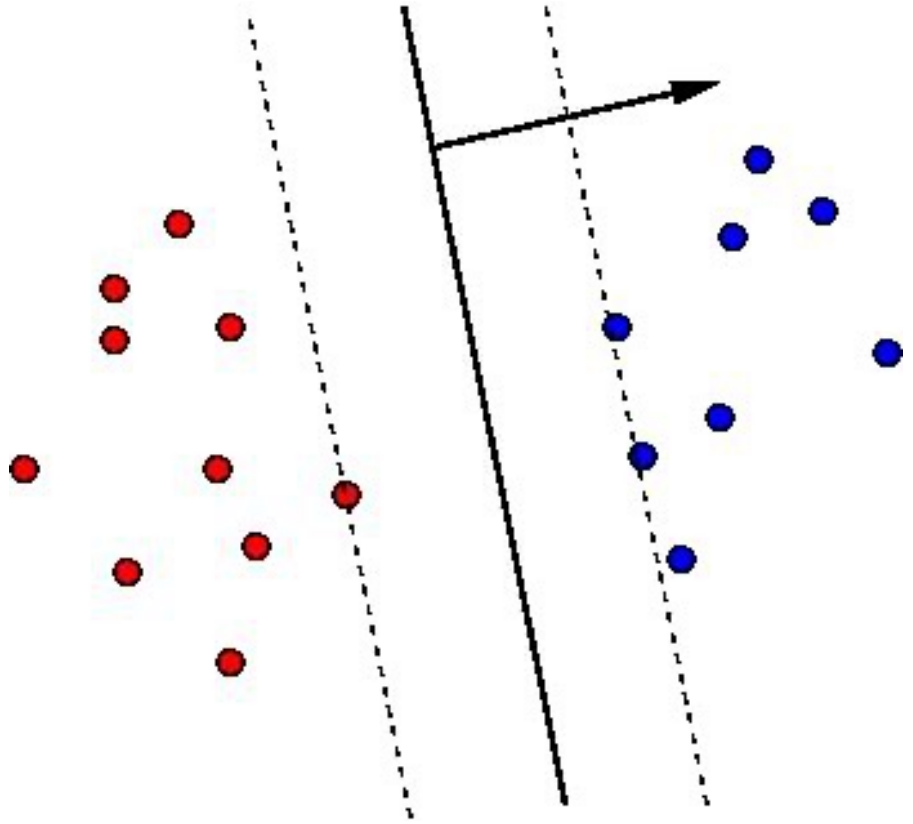
BioSB

Support vectors (3)

- If non-support vectors are left out and training is repeated, the resulting classifier is identical

- The number of support vectors gives a bound on the *leave-one-out error* estimate:

$$\hat{e}_{loo} \leq \frac{\#\ \text{support vectors}}{n}$$



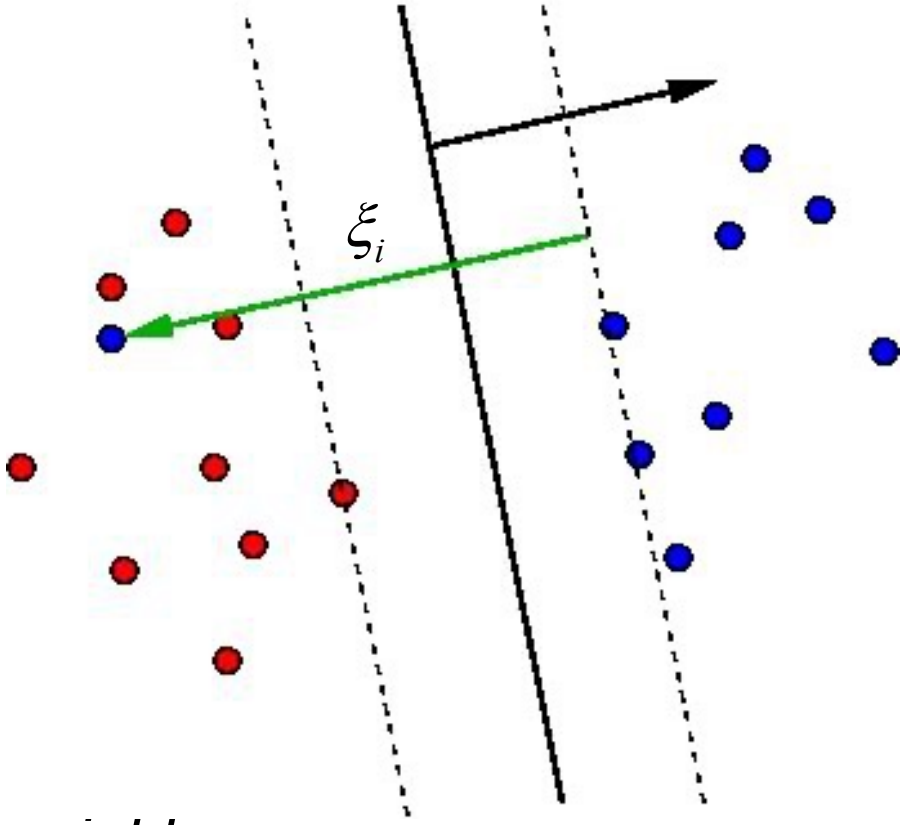
BioSB

Class overlap

- When there is overlap between the classes, the canonical hyperplane is not defined
- To be able to still find a solution, apply a trick:
  soften the constraints that each object is on the correct side of the decision boundary
- For the blue object on the incorrect side of the boundary:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) \geq 1 - \xi_i$$

- The variable ξ_i is called a *slack variable*



BioSB

Class overlap (2)

- In the ideal (non-overlapping) case, all slack variables are 0
- To force slack variables to be small,
  we add them to the margin to be minimized:

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \text{ such that}$$

$$\boldsymbol{w}^T\boldsymbol{x}_i + w_0 \le -(1-\xi_i) \mid y_i = -1$$

$$\boldsymbol{w}^T\boldsymbol{x}_i + w_0 \ge +(1-\xi_i) \mid y_i = +1$$

- We can rewrite that in almost the same way we did before:

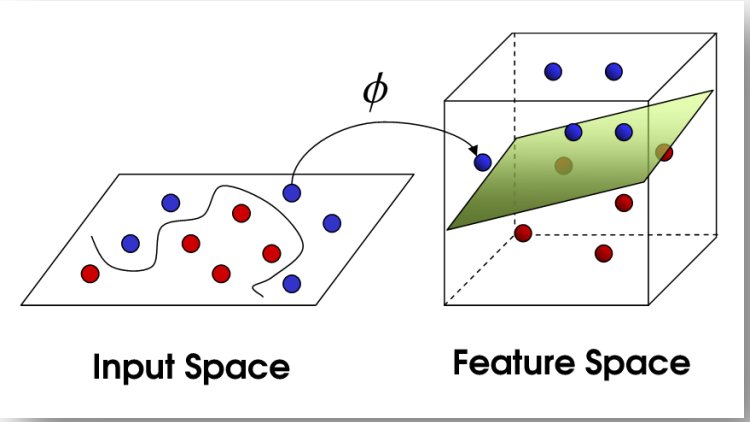
$$\max \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j$$

$$\text{with } 0 \le \alpha_i \le C \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

BioSB

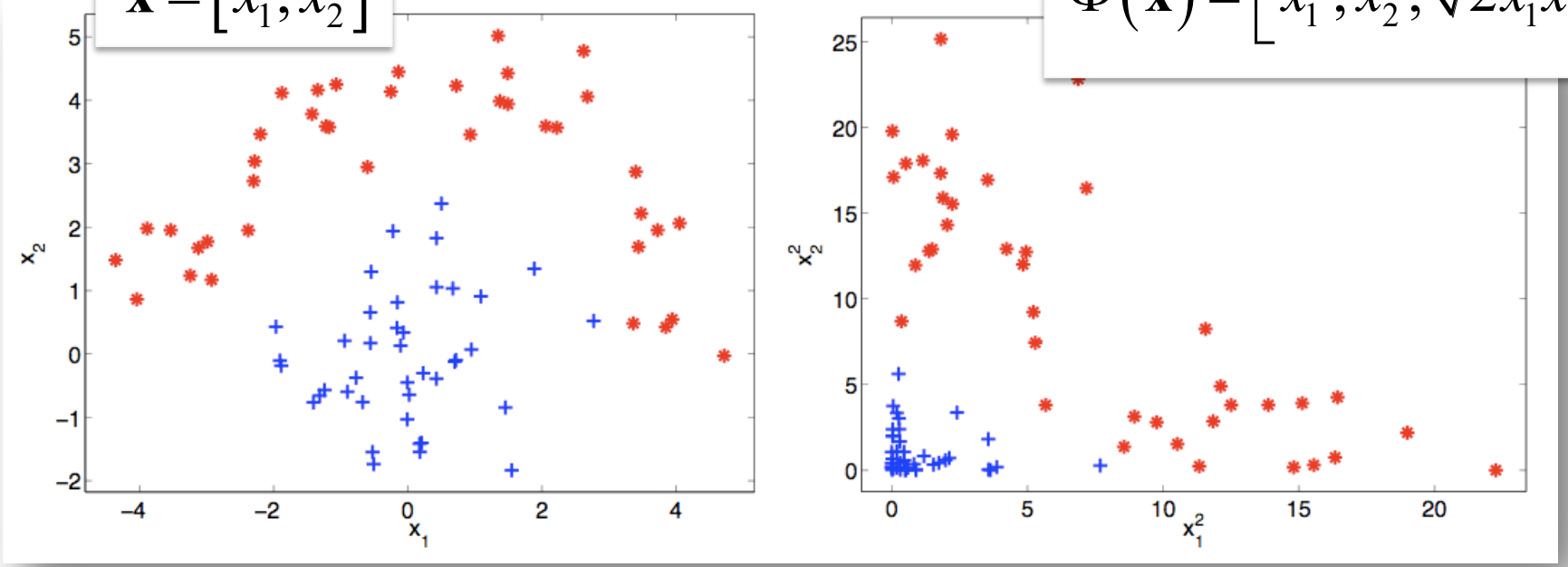
The kernel trick

- Function Φ maps data into a space in which classification may be easier



$$\mathbf{x} = [x_1, x_2]$$

$$\Phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$$



BioSB

The kernel trick (2)

- Classifier:

$$f(\boldsymbol{z}) = \boldsymbol{w}^T \boldsymbol{z} + w_0$$

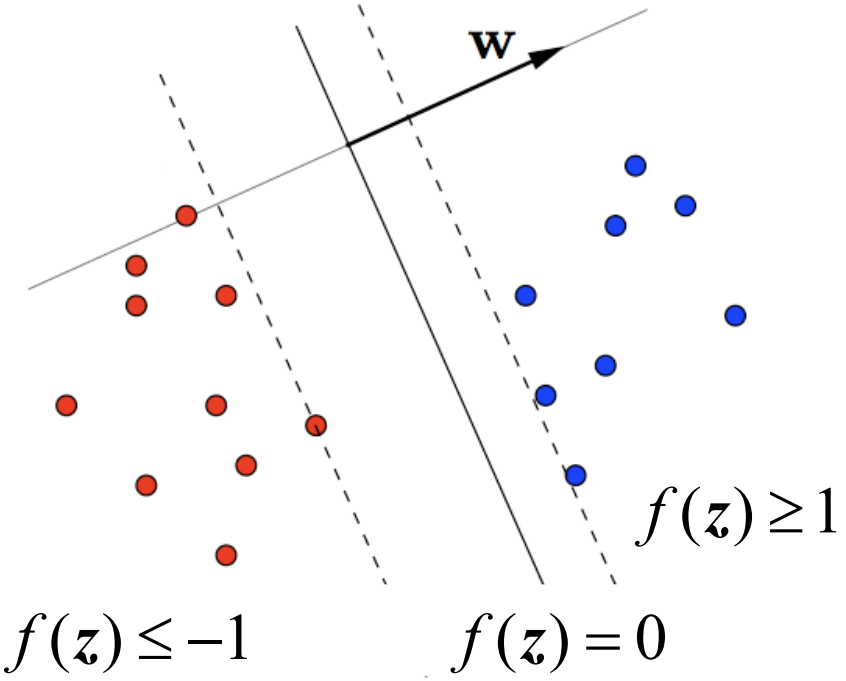
$$= \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{z} + w_0$$

- Optimization problem:

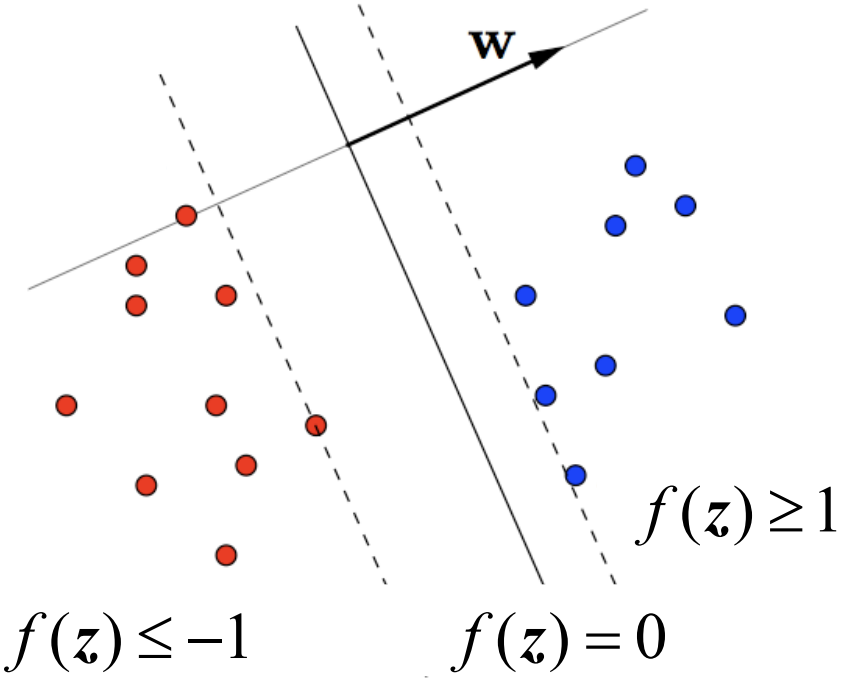
$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$\alpha_i \geq 0, \quad \forall i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$



The kernel trick (3)



- Classifier can be rewritten as:

$$f(z) = w^T \Phi(z) + w_0$$

$$= \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)^T \Phi(z) + w_0$$

- Optimization problem can be rewritten as:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \Phi(x_i)^T \Phi(x_j)$$

$$\alpha_i \geq 0, \quad \forall i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- Only need to specify **kernel** (inner product of transformed points):

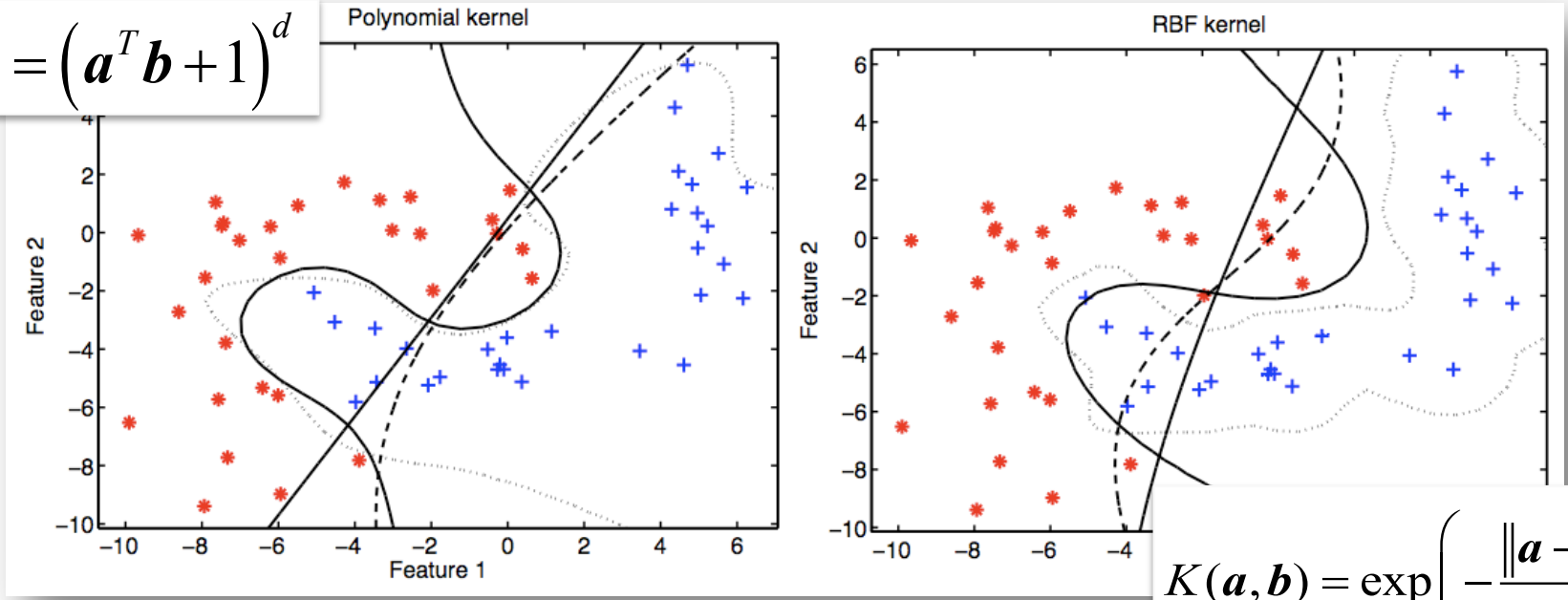
$$K(a,b) = \Phi(a)^T \Phi(b)$$

(inner product = distance)

Kernels

- Kernels $K(\boldsymbol{a},\boldsymbol{b}) = \Phi(\boldsymbol{a})^T \Phi(\boldsymbol{b})$: nonlinear classifier in original space
- Not necessary to actually know $\Phi(.)$,
  as long as $K(\boldsymbol{a},\boldsymbol{b})$ fullfulls some conditions (!) (positive semi-definite)

$$K(\boldsymbol{a},\boldsymbol{b}) = \left(\boldsymbol{a}^T\boldsymbol{b}+1\right)^d$$

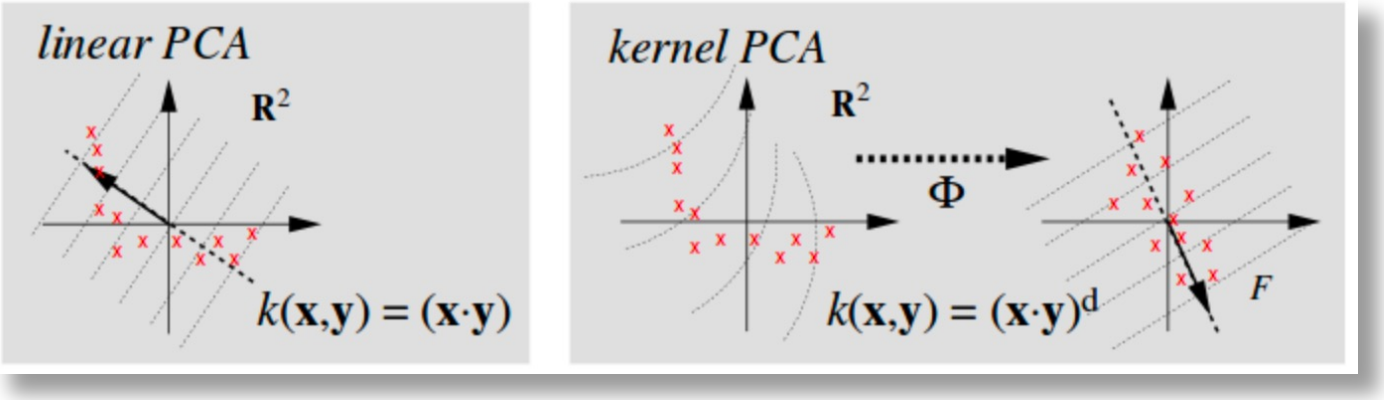


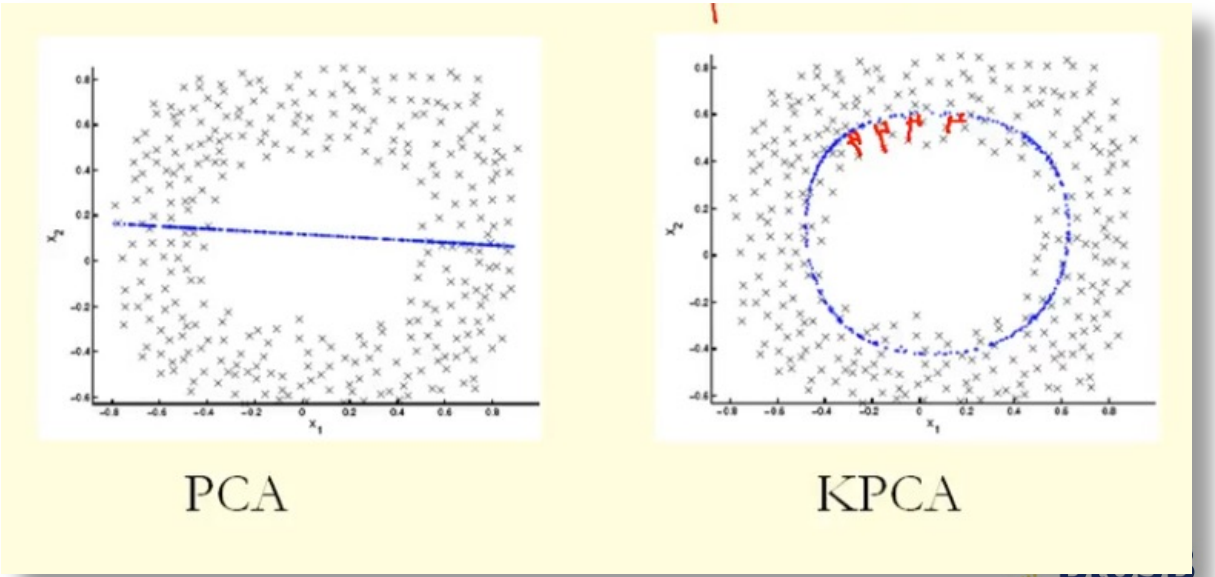
$$K(\boldsymbol{a},\boldsymbol{b}) = \exp\left(-\frac{\|\boldsymbol{a}-\boldsymbol{b}\|^2}{\sigma^2}\right)$$

Positive semi-definite: Hermitian matrix all of whose eigenvalues are nonnegative. One intuitive definition is as follows. Multiply any vector with a positive semi-definite matrix. The angle between the original vector and the resultant vector will always be less than or equal $\pi/2$. The positive definite matrix tries to keep the vector within a certain half space containing the vector.

Kernel not restricted to SVMs

- Also kernel versions of PCA, ICA, LDA, CCA, ...





Kernels

- Vector kernels:
  - Linear $$K(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{a}^T\boldsymbol{b}$$
  - Polynomial $$K(\boldsymbol{a},\boldsymbol{b}) = \left(\boldsymbol{a}^T\boldsymbol{b}+1\right)^d$$
  - Radial basis function $$K(\boldsymbol{a},\boldsymbol{b}) = \exp\left(-\frac{\|\boldsymbol{a}-\boldsymbol{b}\|^2}{\sigma^2}\right)$$

BioSB

Kernels (2)

- For other data types: empirical kernel map
  - If we have some kind of a distance measure (not *per se* positive definite),
    then for each object we can construct a vector with distances to a number of other objects
  - This vector can then be used in a vector kernel

- Example: BLAST kernel
  - BLAST a set of sequences w.r.t. each other
  - Represent each sequence by
    a vector of -log(E)-values
  - Use linear kernels on these vectors

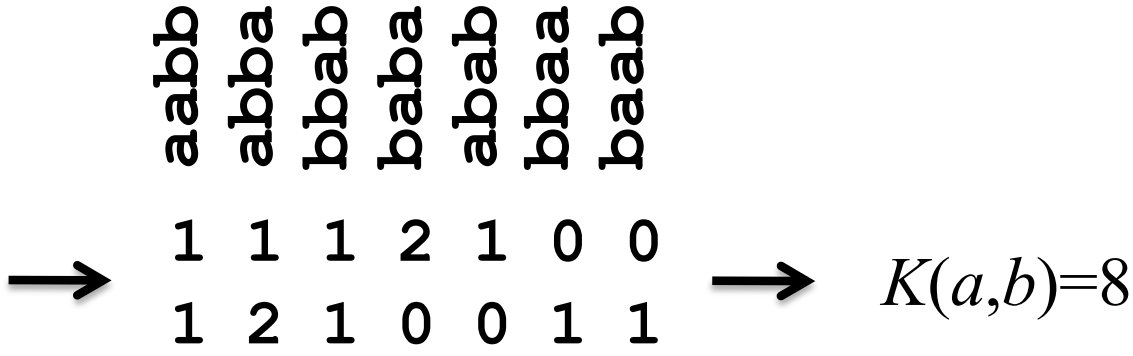
BioSB

Kernels (3)

- Spectrum kernel:
  - Construct a dictionary of all k-mers
  - Construct vector with #occurences of each k-mer
  - Use this in a linear kernel
  - Need for smart data structures (trie)
  - Versions with gaps, substitutions, wildcards...

- Example:

a = **aabbababa**
b = **abbaabbab**



BioSB

Kernels (4)

- Convolution kernel:
  - When kernels operate on subparts,
    but it is not clear which subparts
  - Try all possible decompositions into subparts:

$$K_1 \otimes K_2 \otimes \ldots \otimes K_n(\boldsymbol{a},\boldsymbol{b}) = \sum_{\substack{\boldsymbol{a}=\boldsymbol{a}_1\boldsymbol{a}_2\ldots\boldsymbol{a}_n \\ \boldsymbol{b}=\boldsymbol{b}_1\boldsymbol{b}_2\ldots\boldsymbol{b}_n}} K_1(\boldsymbol{a}_1,\boldsymbol{b}_1)K_2(\boldsymbol{a}_2,\boldsymbol{b}_2)\ldots K_n(\boldsymbol{a}_n,\boldsymbol{b}_n)s$$

BioSB

Kernels (5)

- Local alignment kernel:
  - Trivial kernel: $K_t(\boldsymbol{a},\boldsymbol{b}) = 1$
  - Letter alignment kernel: $K_a(\boldsymbol{a},\boldsymbol{b}) = \begin{cases} 0 & |\boldsymbol{a}| > 1 \vee |\boldsymbol{b}| > 1 \\ \exp\left(\beta S(\boldsymbol{a},\boldsymbol{b})\right) & \text{otherwise} \end{cases}$

    with S the substitution cost
  - Gap kernel: $K_g(\boldsymbol{a},\boldsymbol{b}) = \exp\left(\beta\left(|\boldsymbol{a}| + |\boldsymbol{b}|\right)\right)$
  - Local alignment kernel of length n:

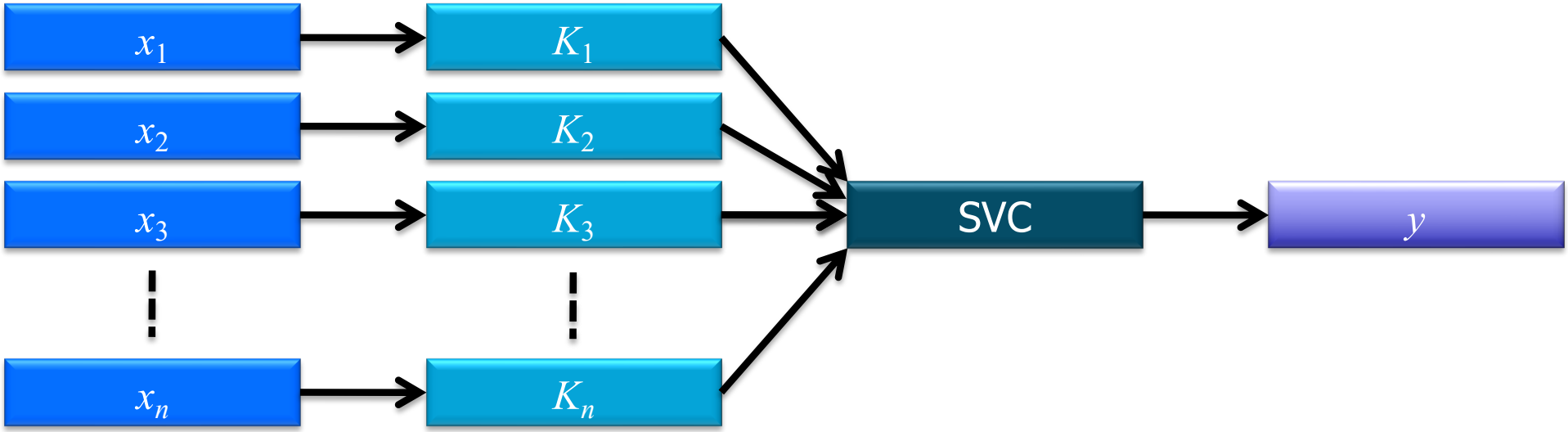
$$K_{la(n)}(\boldsymbol{a},\boldsymbol{b}) = K_t \otimes \left(K_a \otimes K_g\right)^{(n-1)} \otimes K_a \otimes K_t(\boldsymbol{a},\boldsymbol{b})$$

  - Local alignment kernel:

$$K_{la}(\boldsymbol{a},\boldsymbol{b}) = \sum_{n=0}^{\infty} K_{la(n)}(\boldsymbol{a},\boldsymbol{b})$$

BioSB

Kernel combination



- Combination: weighted sum of normalized kernel matrices

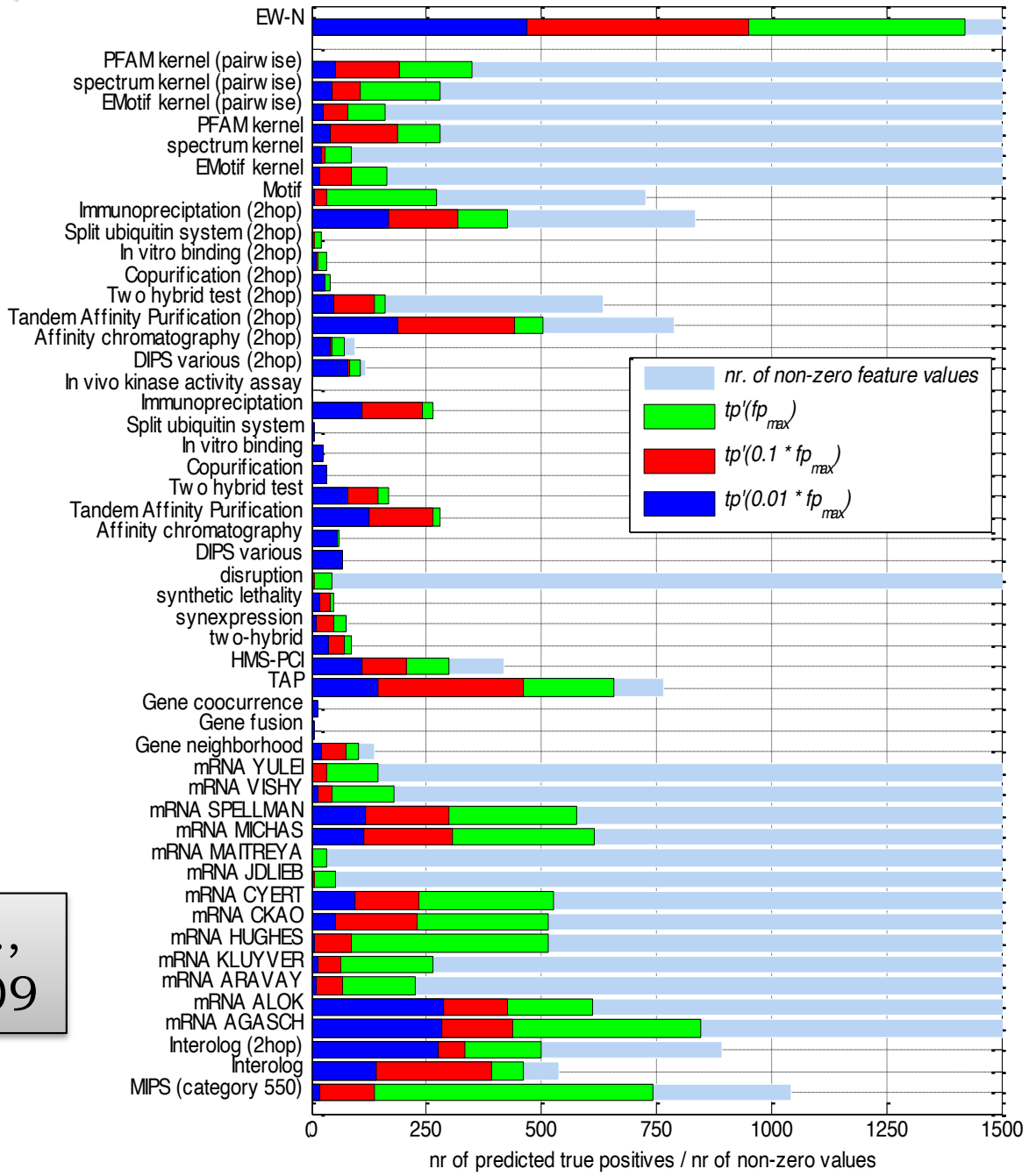
$$K_i'(a,b) = \frac{K_i(a,b)}{\sqrt{K_i(a,a)K_i(b,b)}} \qquad K_{combined}(a,b) = \sum_{i=1}^{n} w_i K_i'(a,b)$$

powerful: can apply optimal kernel to each data type

BioSB

Application

- Protein complex co-member-ship



High accuracy

Medium accuracy

Low accuracy

Hulsman *et al.*, *IEEE TBCB* 2009

BioSB

Recapitulation

- The *support vector classifier* is based on a well-founded theoretical basis (*Vapnik dimension*)
- The original support vector classifier is limited to problems with two non-overlapping classes, but:
  - can be extended to overlapping classes using *slack variables*
  - can be extended to nonlinear decision boundaries using *kernels*
  - can be extended to multiple classes by combining multiple 2-class classifiers
- A large number of specific kernels for biological data are available
- A support vector regressor is available (not discussed)

BioSB

Recapitulation (2)

- Classification performance is often very good
- In particular suited for problems with high-dimensional datasets, for which classes are often separable (and hence estimating densities is extremely difficult)
- The optimization problem is formulated in terms of the training objects, not the features: slow training for large datasets
- The value for the slack variable trade-off C and kernel-specific parameters d, σ etc. have to be set

Kernels need to be chosen, also an ART!

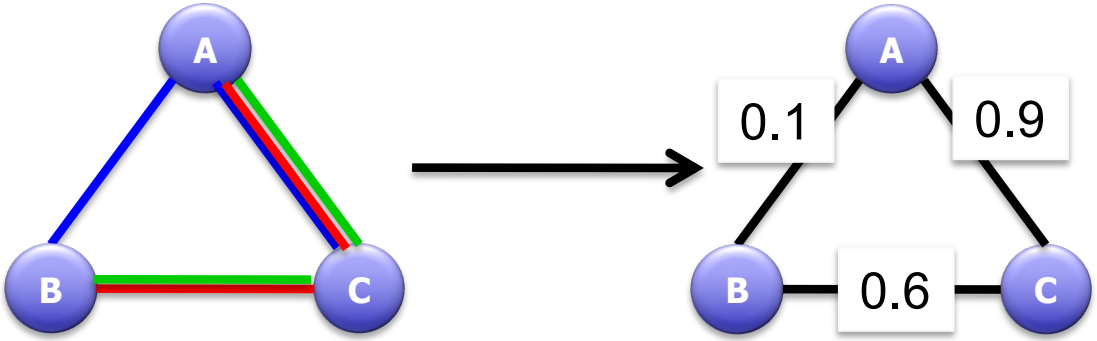
BioSB

10 min break

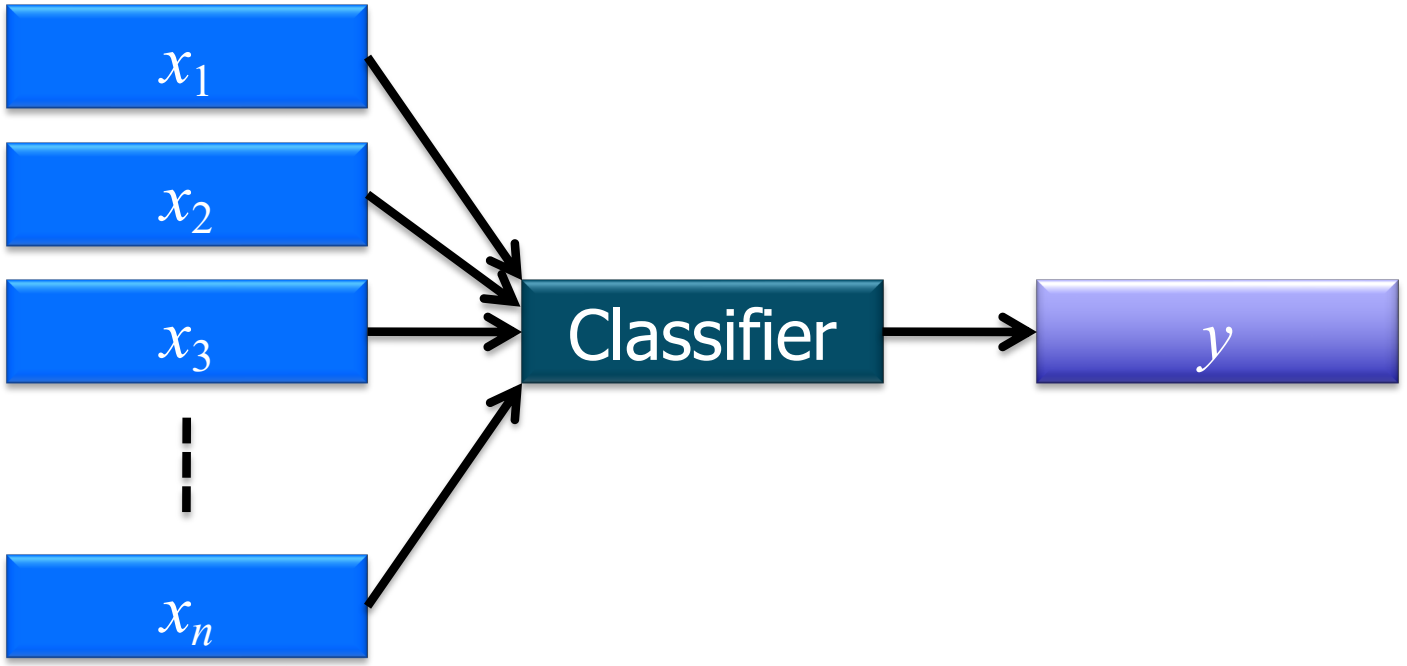
Classifier combination

Data integration

- Often required in bioinformatics, e.g. in interaction prediction



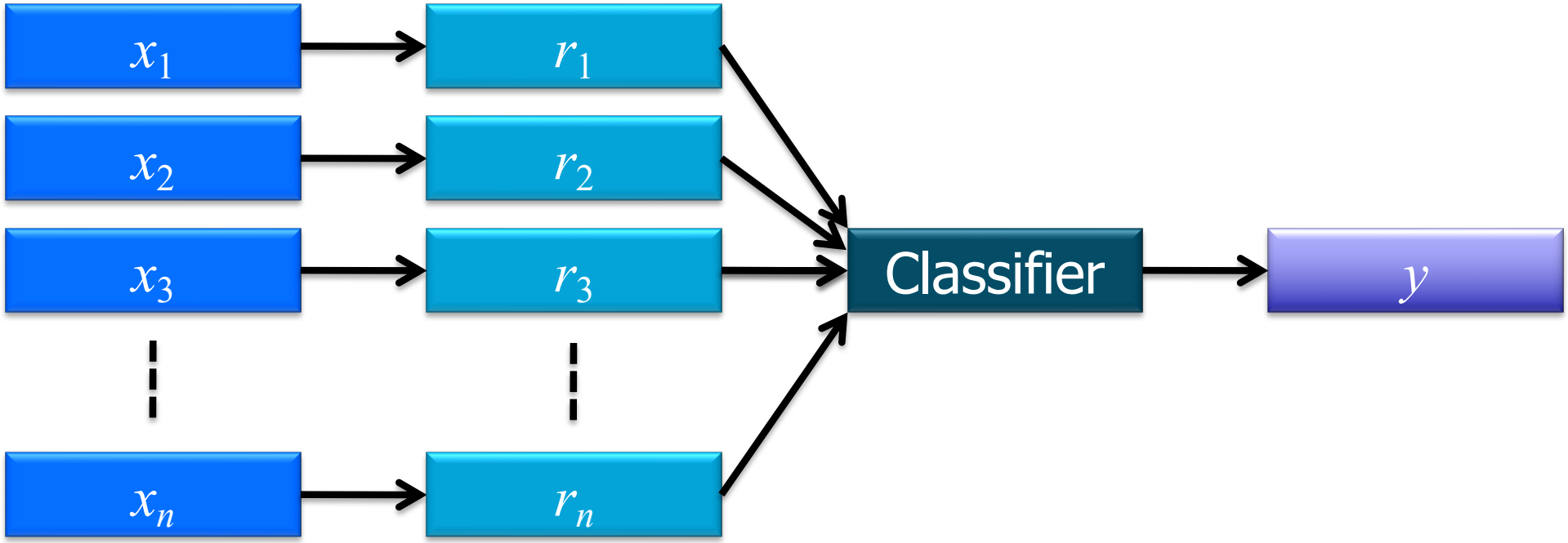
- Early integration: feature fusion



BioSB

Data integration (2)

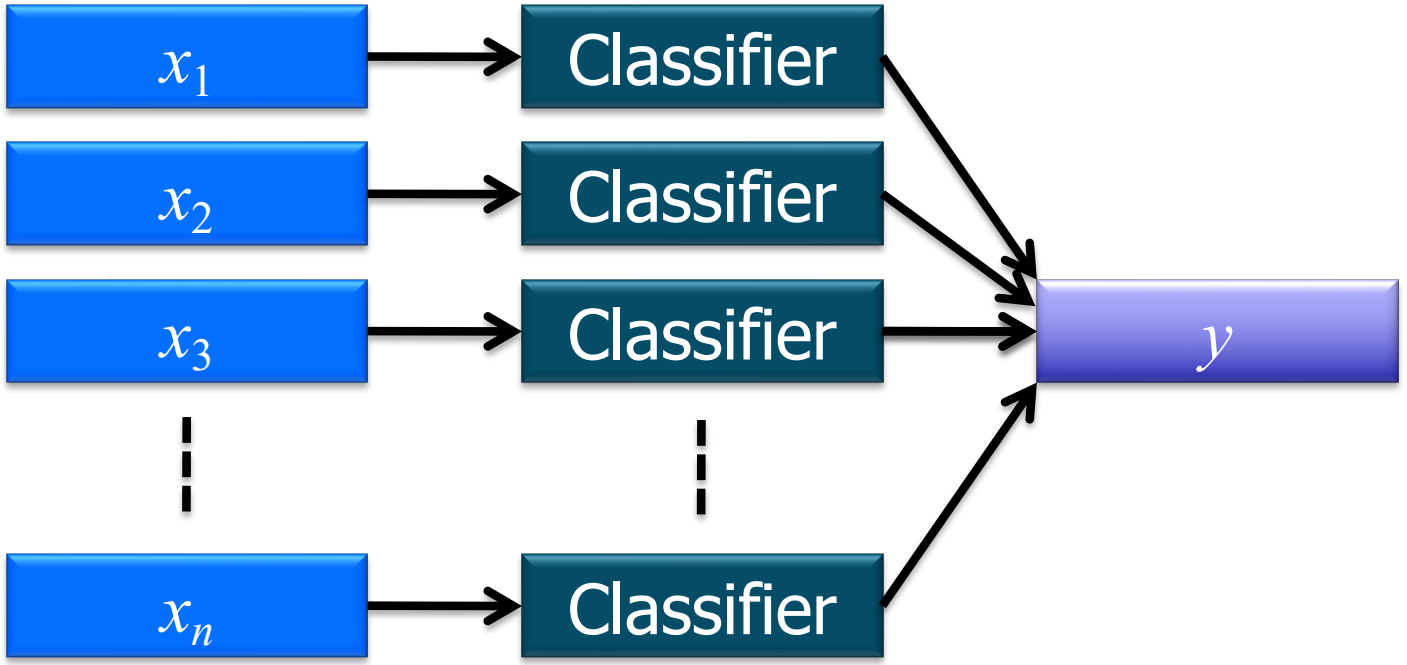
- Intermediate integration: common representation (e.g. kernels or probability distributions)



BioSB

Data integration (3)

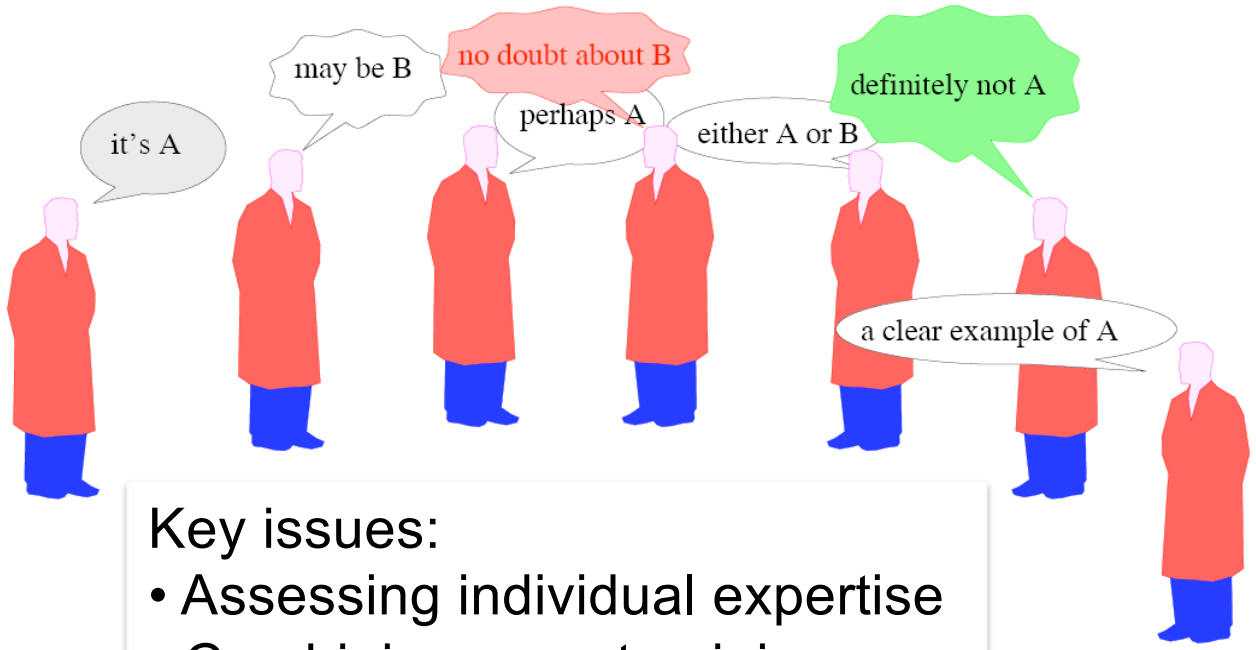
- Late integration: classifier combination



BioSB

Classifier combination (late integration)

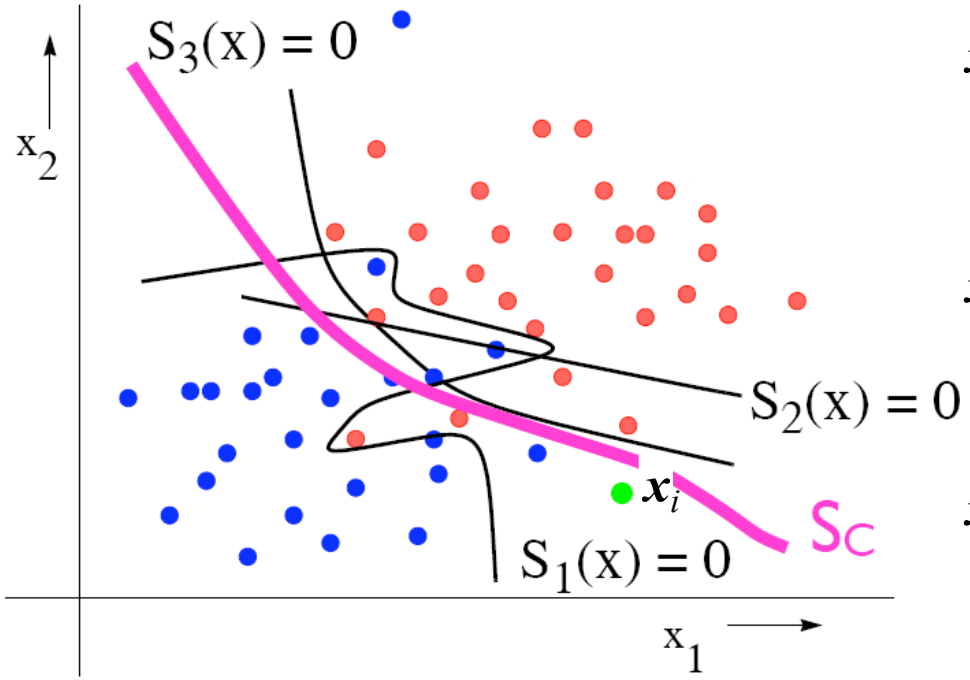
- Design choices:
  - Base classifier: Identical or different?
    Base classifiers, feature spaces, training sets, initialisations, etc.
  - Combination by a fixed rule or by another classifier?
- Related to work on committees-of-experts

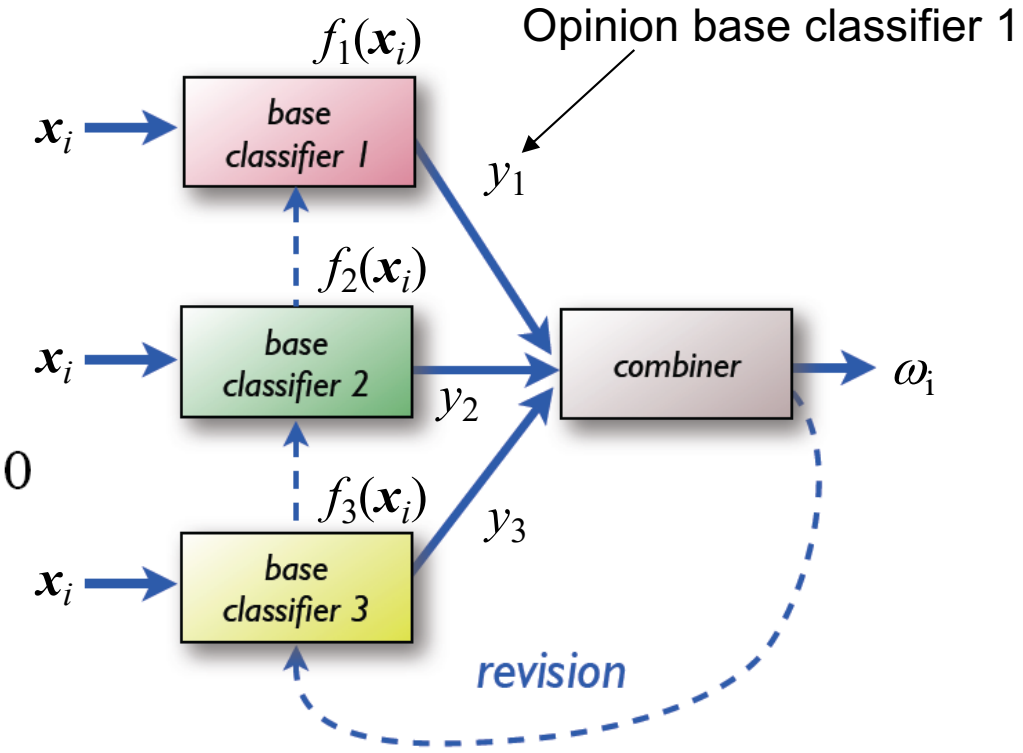


Key issues:
• Assessing individual expertise
• Combining expert opinions

BioSB

Fixed combination





- Classifiers: individual opinion = **posterior probabilities** or **labels**
- Combination by **fixed rule**, e.g.:

$$\omega_i = \arg\max_c (\text{combination-rule}(y_{j,c} = f_{j,c}(\boldsymbol{x}_i)))$$

i.e. assign label $\omega_i = c$ to object $\boldsymbol{x}_i$ if the combination of outputs $y_{j,c}$ for class c over all classifiers $f_j(\boldsymbol{x}_i)$ is maximum

Combination rule might be maximum over all classifiers j, or votes by all classifiers for that class

Fixed combination (2)

- Combination rules on **posterior probabilities** $y_{j,c} = p(\omega_i{=}c|\boldsymbol{x}_i)$:
  - Generally applicable:
    - Maximum, to select “most confident” classifier

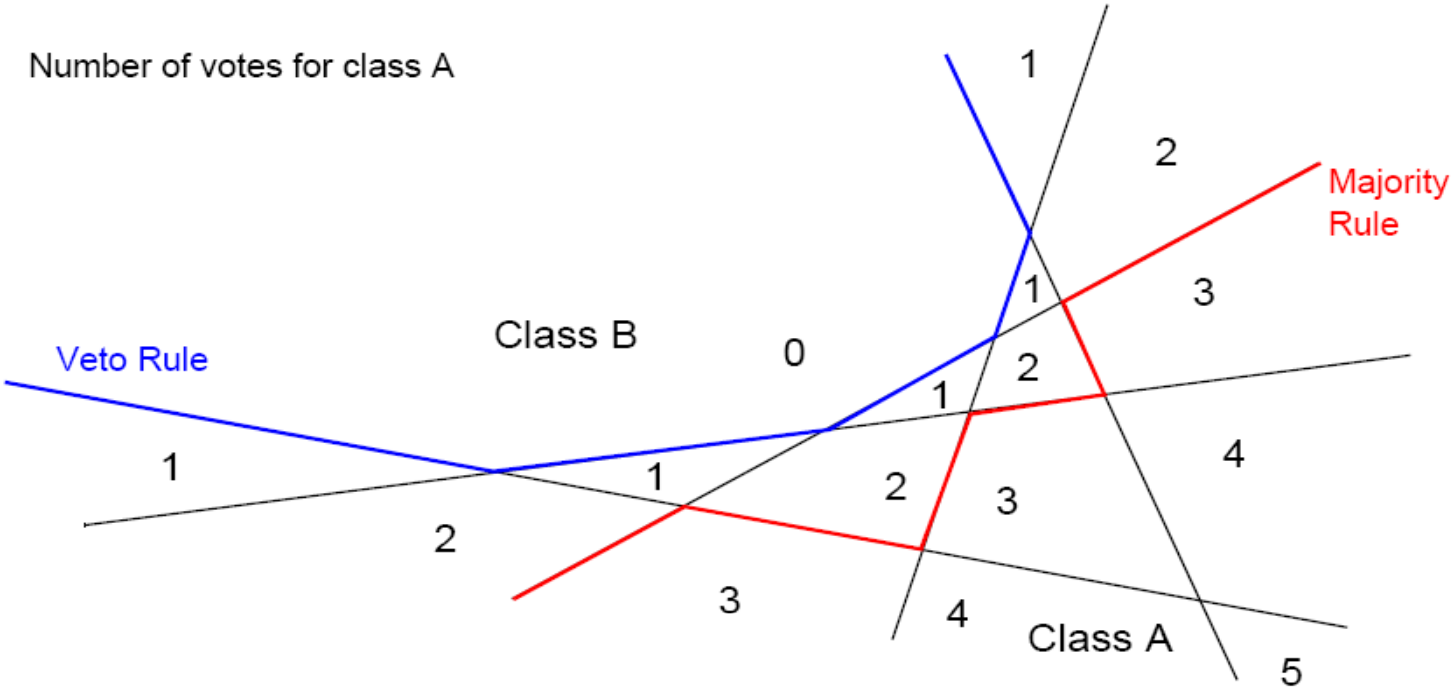
    (assumes good estimates of posteriors)
  - Preferable for classifiers trained in different feature spaces:
    - Product, justified if feature spaces independent
    - Minimum, to select “least objecting” classifier

    (assumes good estimates of posteriors)
  - Preferable for comparable classifiers trained on the same features:
    - Sum/median, to (robustly) improve estimates of posteriors

BioSB

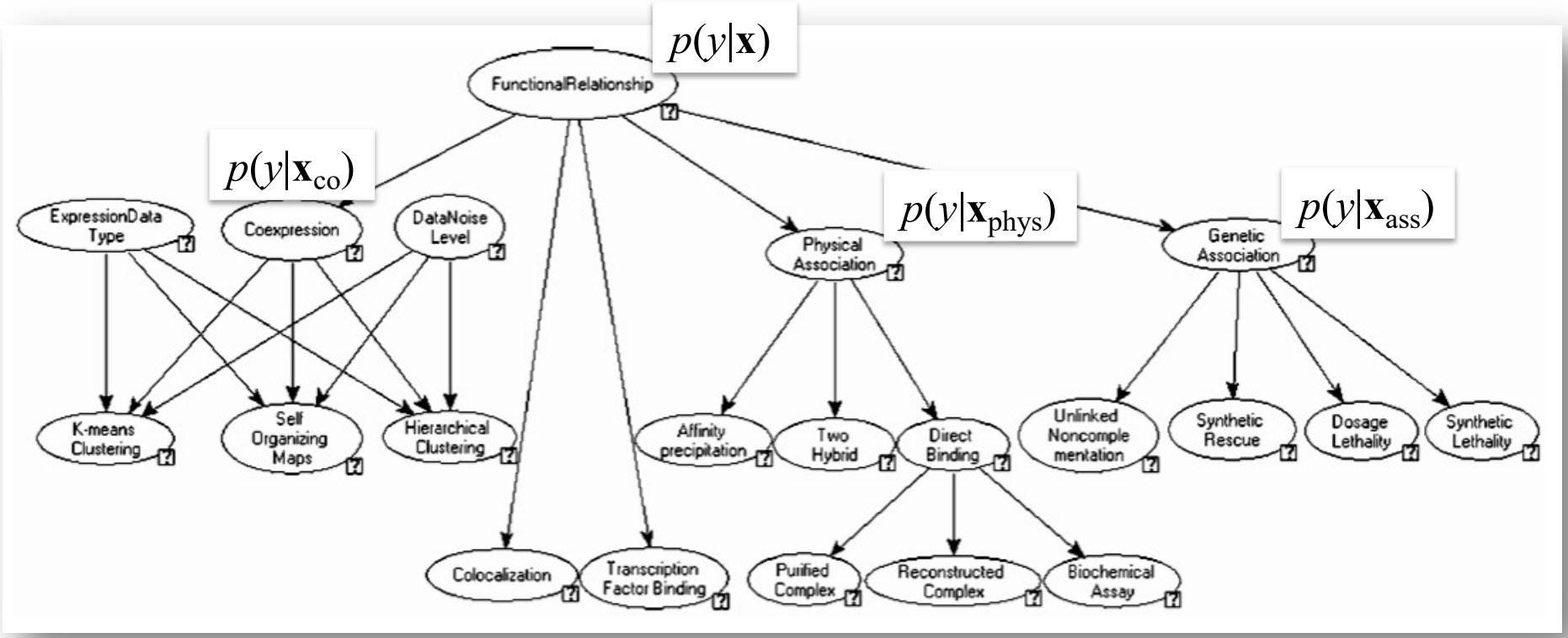
Fixed combination (3)

- Alternatively, **combine labels** assigned by classifiers:
  - Veto (like minimum)
  - Majority vote (like sum/median)



Bayesian network to integrate

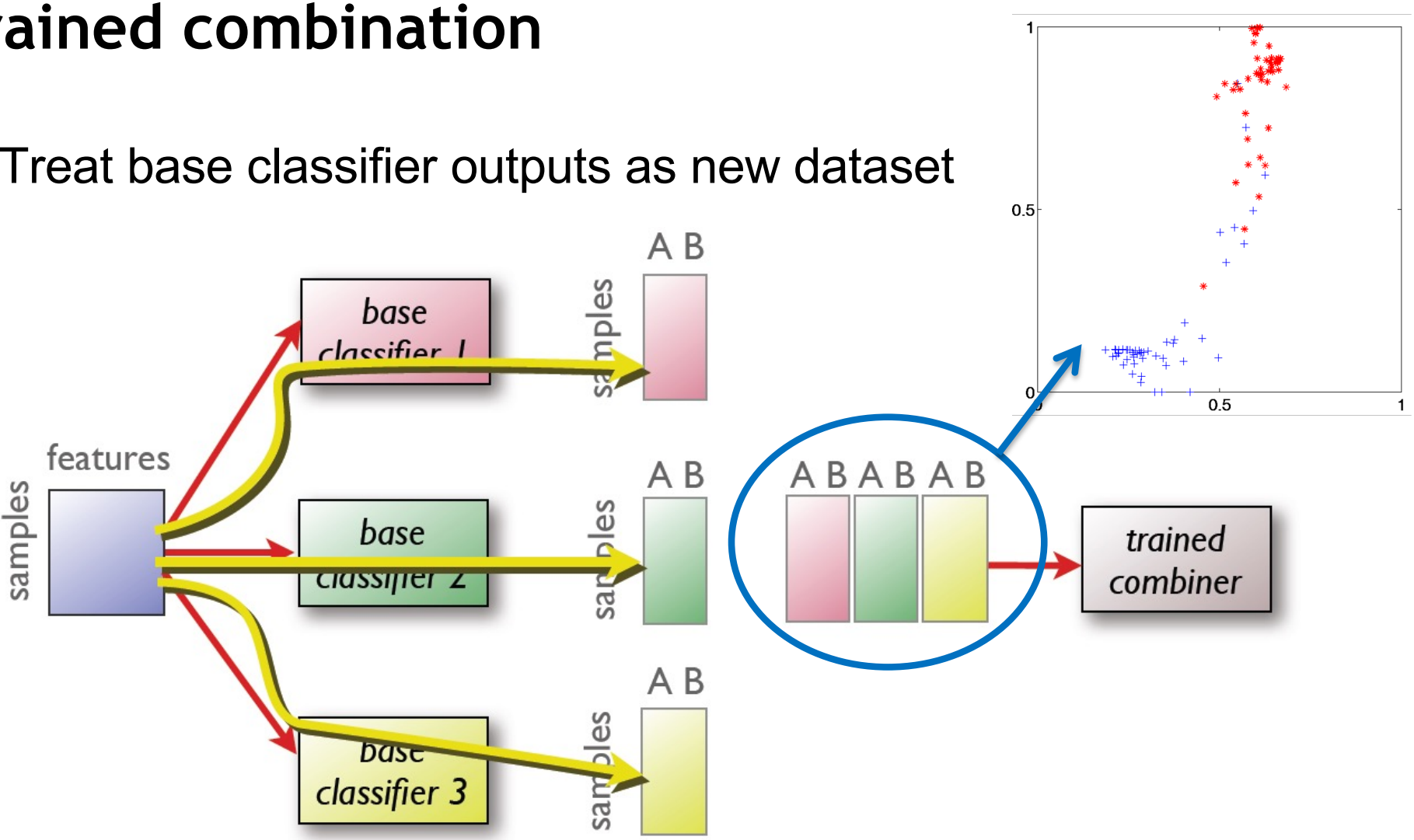
- MAGIC (Troyanska et al., PNAS 2003):
  Bayesian network, integration through “summary nodes”
- All parameters set manually, based on expert knowledge



BioSB

Trained combination

- Treat base classifier outputs as new dataset



- In principle, possible to use any classifier
- Danger of overtraining when using full training set for both stages: use (nested) cross-validation!

BioSB

Base classifier generation

Let's not combine some classifiers, but set out to generate MANY

- Bagging: bootstrapping and aggregating
  - For B repetitions
    - Sample a subset of size $n' < n$ using bootstrapping
    - Train classifier on this subset (e.g. linear or decision tree)
  - Combine B classifier outputs (e.g. sum or vote)

- Boosting:
  - Initialize all objects with equal weight
  - As often as necessary/wanted
    - Sample a subset of size $n' < n$ according to object weights
    - Train a *weak classifier* on this subset
    - Increase weights of incorrectly classified objects
  - Combine classifier outputs

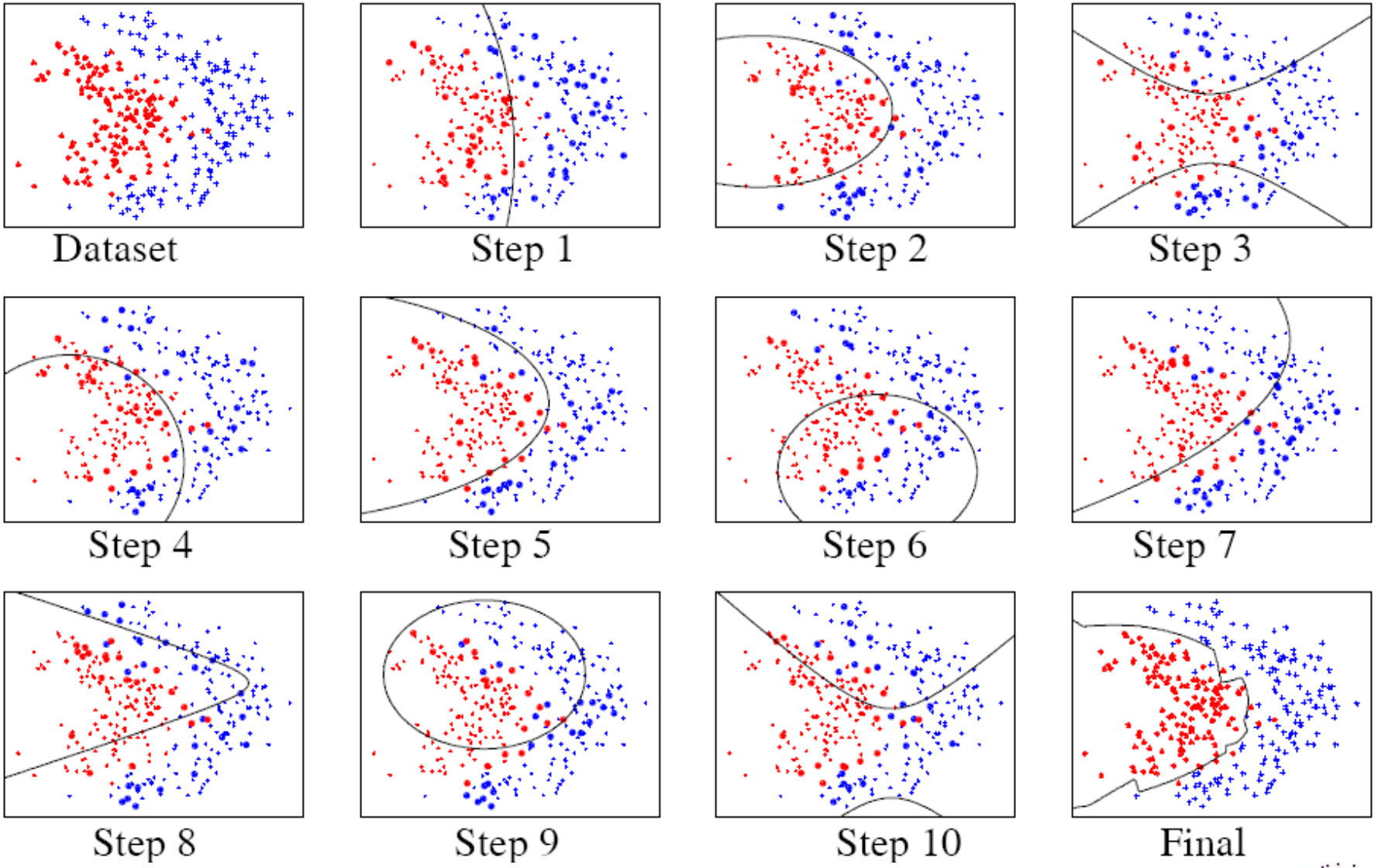
Use weak classifiers: only sensible to average over things that differ

Base classifier generation (2)

- Adaboost:
  - Initialize all objects with equal weight
  - As often as necessary
    - Select a train set size $n' < n$ according to object weights
    - Train a weak classifier j
    - Classify entire data set and calculate classifier error e_j
    - Calculate classifier weight $\alpha_j = 0.5 \log((1-e_j)/e_j)$
    - Multiply weights of incorrectly classified objects with $\exp(\alpha_j)$, multiply weights of correctly classified objects with $\exp(-\alpha_j)$
  - Combine weak classifiers by weighted voting, using α_j

Boosting: weight objects with #errors
Adaboost: weight objects with classifier error

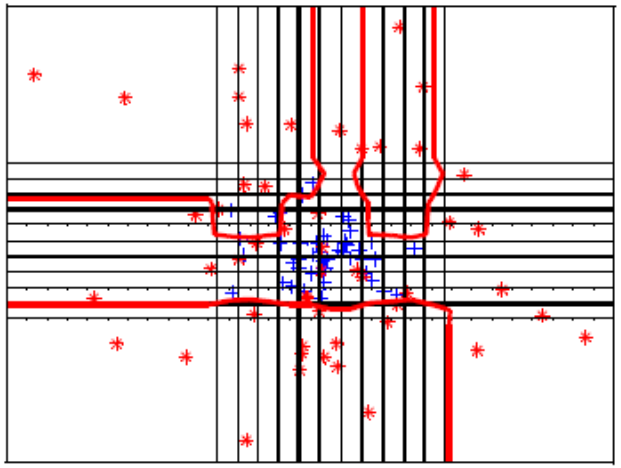
Base classifier generation (3)



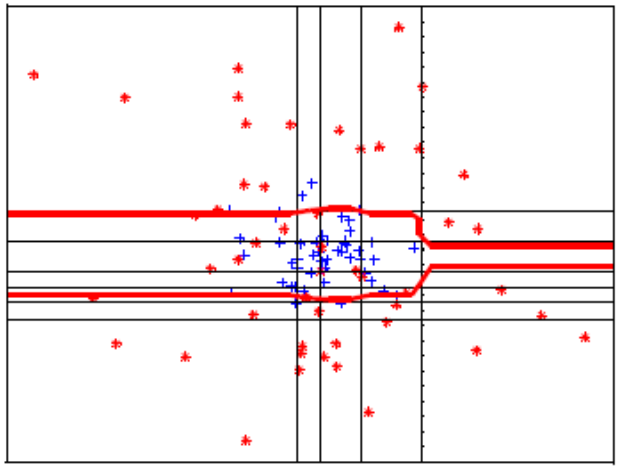
- Adaboost example

Base classifier generation (4)

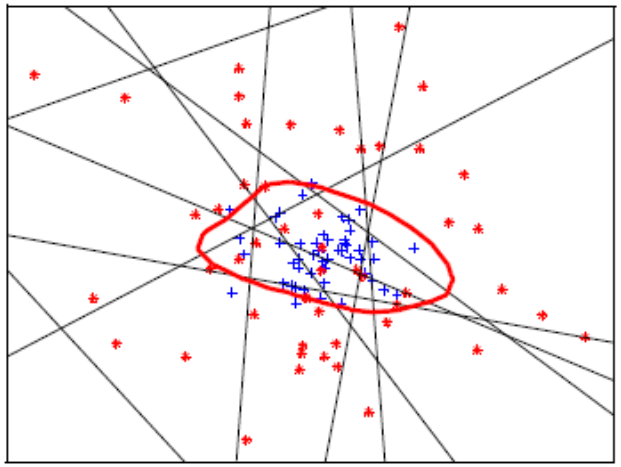
- For all combination methods: base classifier should be fast and weak, i.e. have large bias and small variance
  - Decision stumps: short decision trees
  - Linear classifiers: nearest mean, LDA



100 decision stumps, combined by Adaboost



10 decision stumps, combined by LDA



10 LDAs, combined by LDA

BioSB

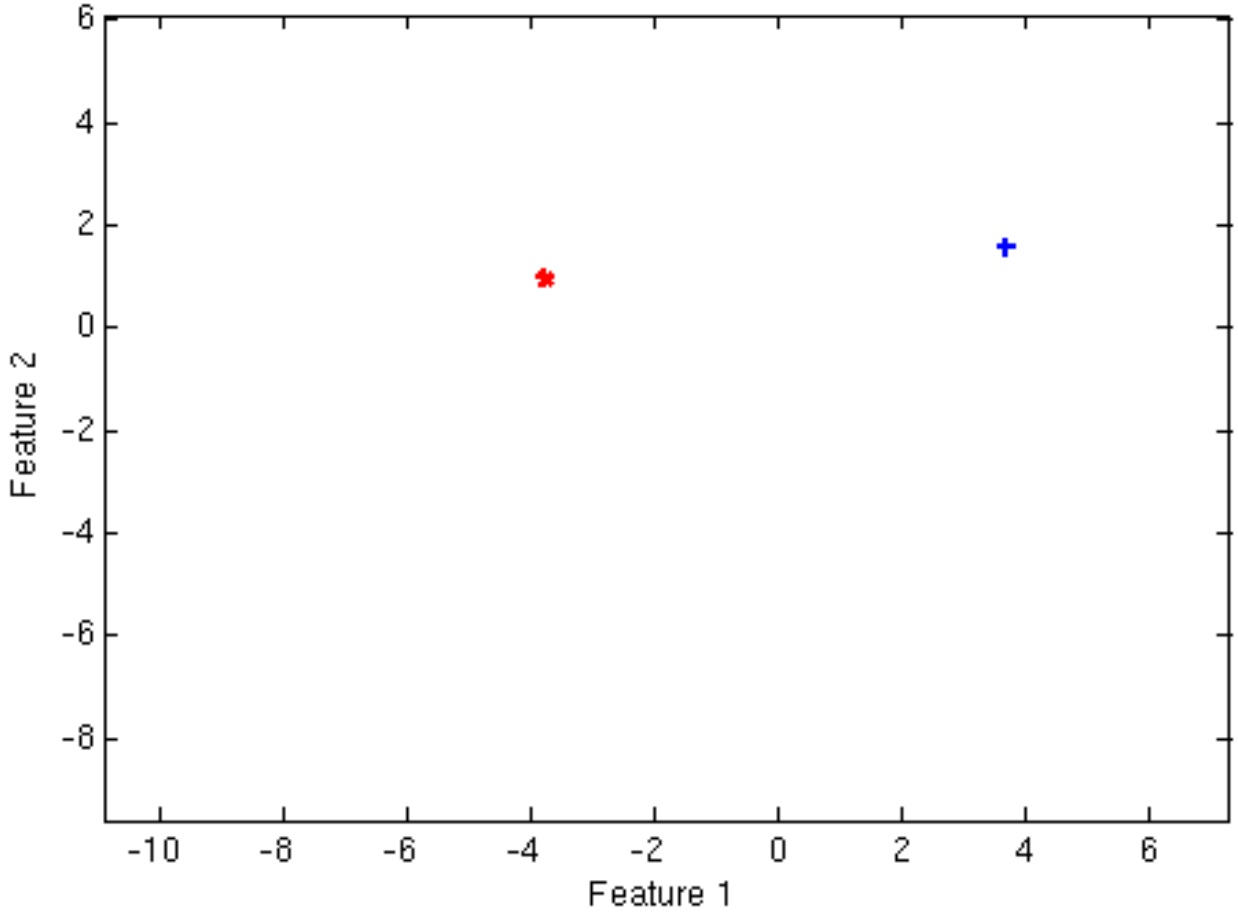
Recapitulation

- Combining classifiers can help, but is no panacea
  - *Fixed* combination:
    - Usually sub-optimal
  - *Trained* combination:
    - Use cross-validation to prevent overtraining
- Use *weak* classifiers: fast, large bias, small variance
- Combination requires *variation* between classifiers:
  - Train different classifiers on the same features
  - Train classifiers on different feature spaces (sample features!)
  - Subsample the train set (*bagging*, *boosting*)

BioSB

Complexity

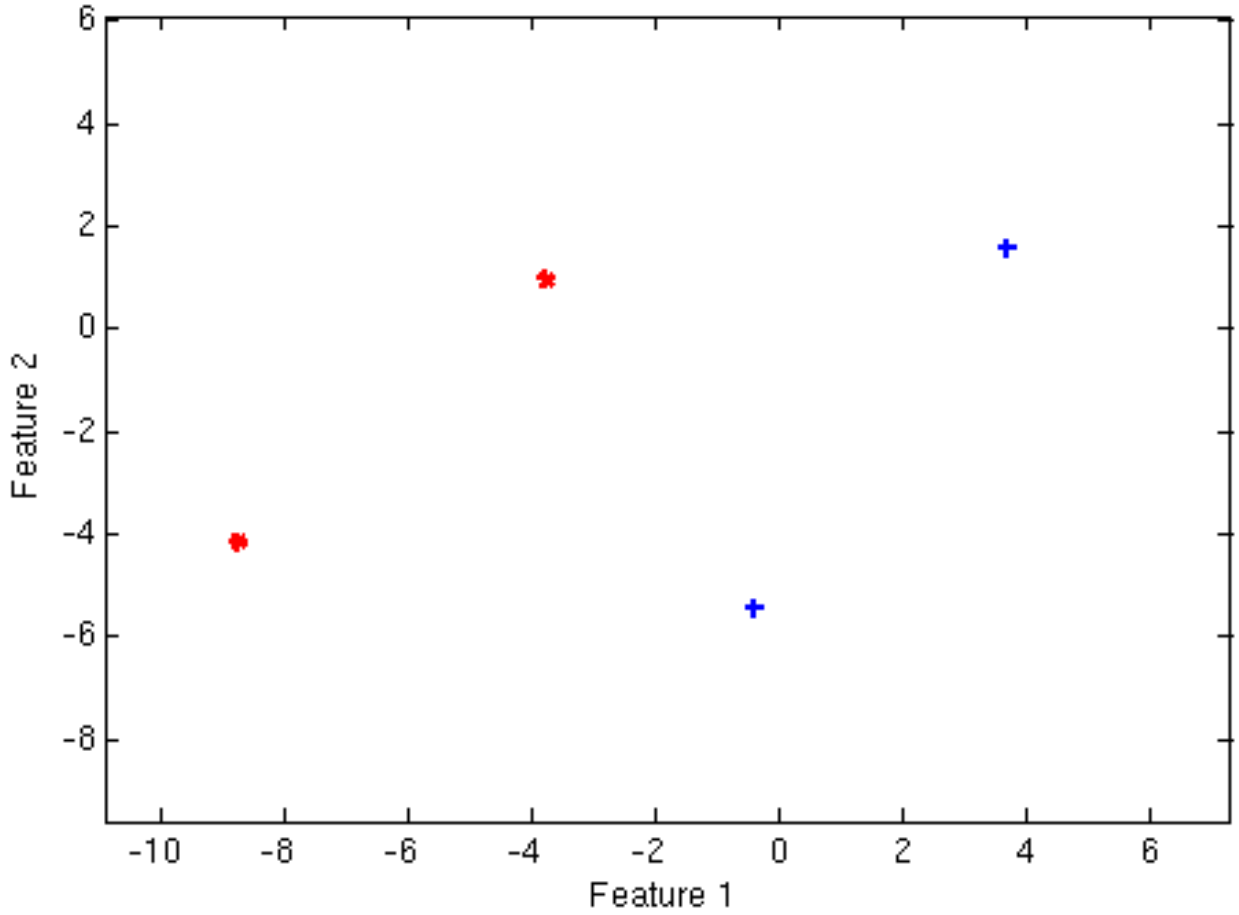
Sample size



What is a good classifier?

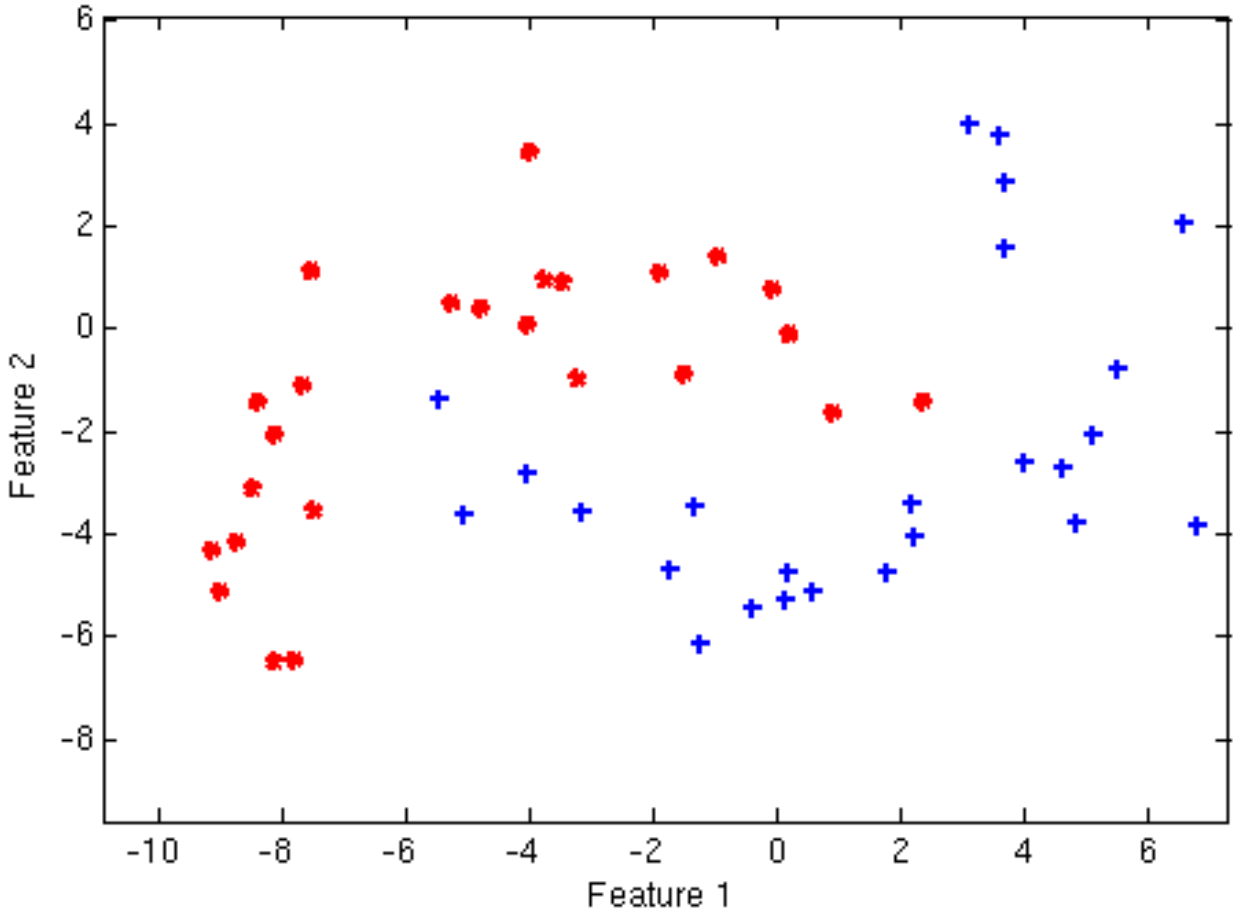
BioSB

Sample size (2)



What is a good classifier?
And now?

Sample size (3)

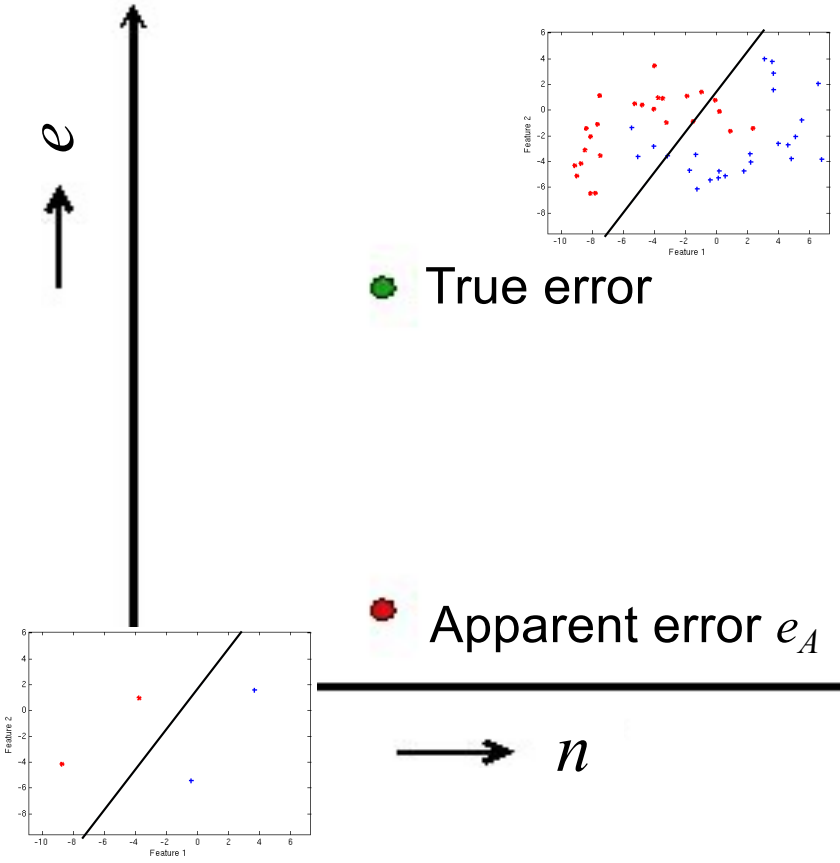


What is a good classifier?
And now? Training size matters! But how?

BioSB

Learning curves

- How does the error change with varying sample size (number of objects in the train set)?



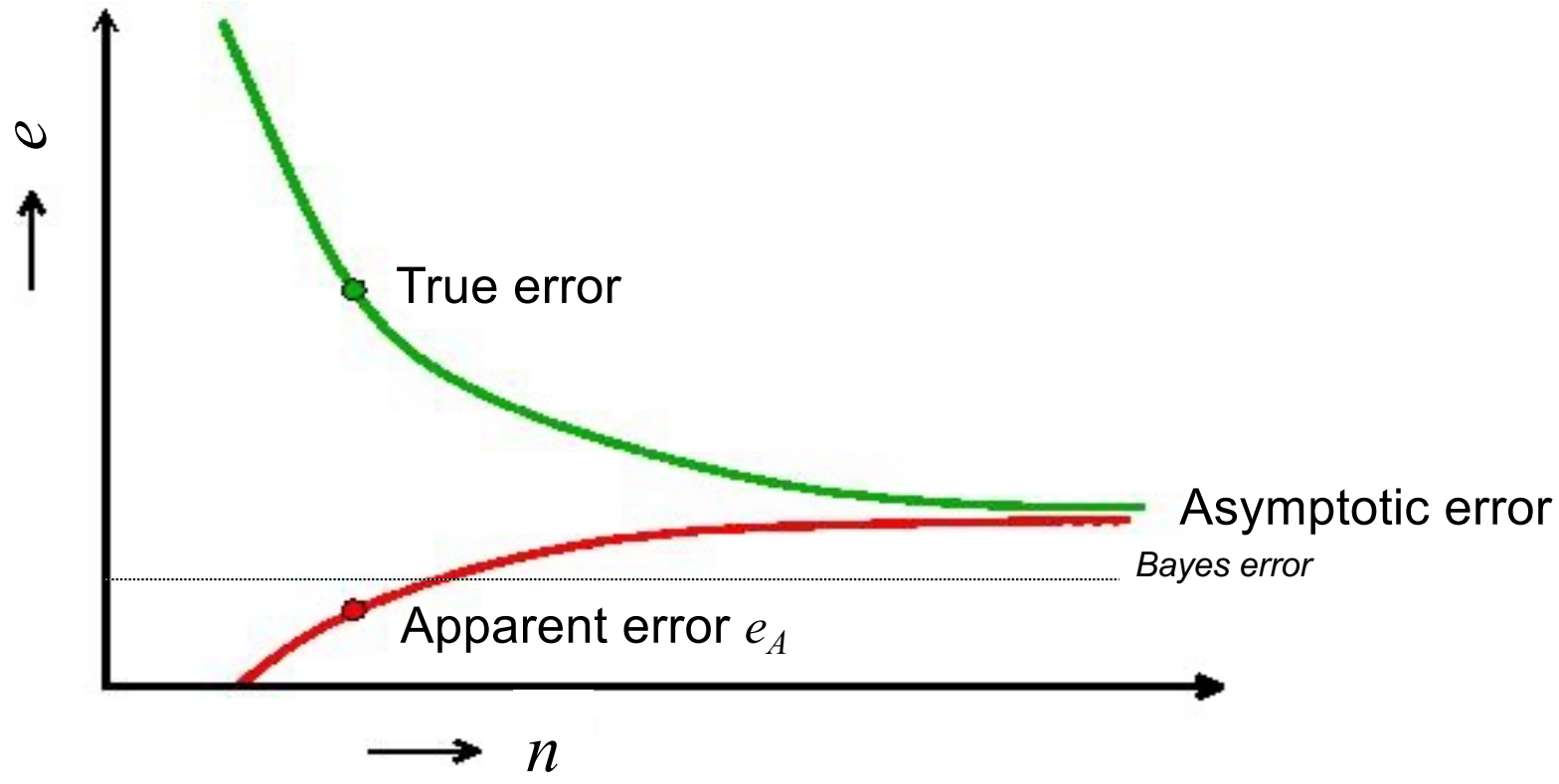
True error not small because of linear classifier and data is not linearly separable

True error: error on infinite test data
Apparent error: error on training data

BioSB

Learning curves (2)

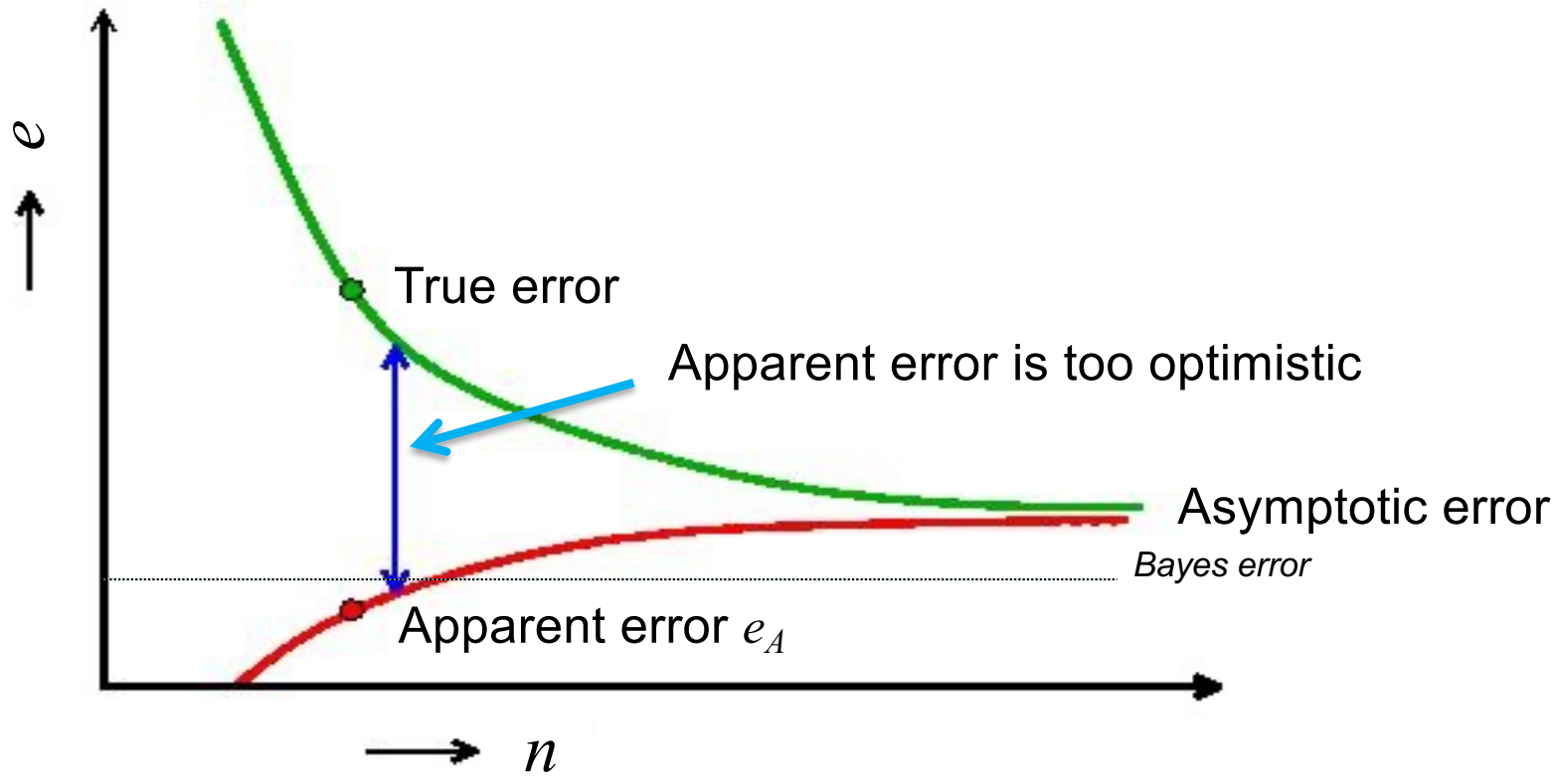
- How does the error change with varying sample size (number of objects in the train set)?



Bayes error: overall minimal error (can be smaller than true error for given classifier)

Learning curves (3)

- How does the error change with varying sample size (number of objects in the train set)?



Learning curves (4)

- What happens when you take another classifier?
  (say, use a **`qdc`** instead of an **`ldc`**)

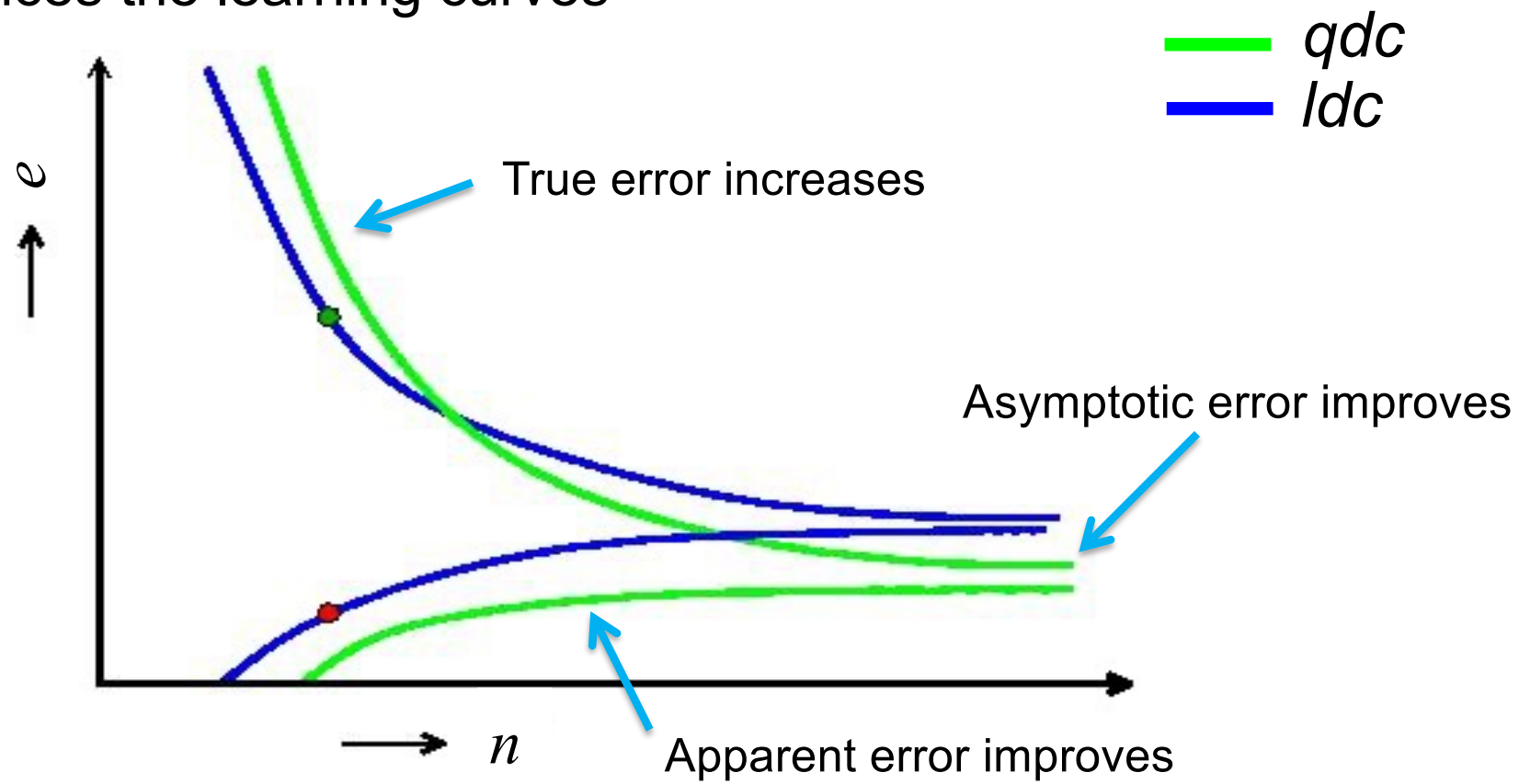
- More flexible:
  - Better performance on the training set
  - Worse performance on the test set
  - Will perform best in the limit of many training objects

- Less flexible:
  - Less adapted to the training set
  - Better performance on the test set
  - Will not perform best in the limit of many training objects

BioSB

Learning curves (5)

- Switching to a more complex classifier influences the learning curves

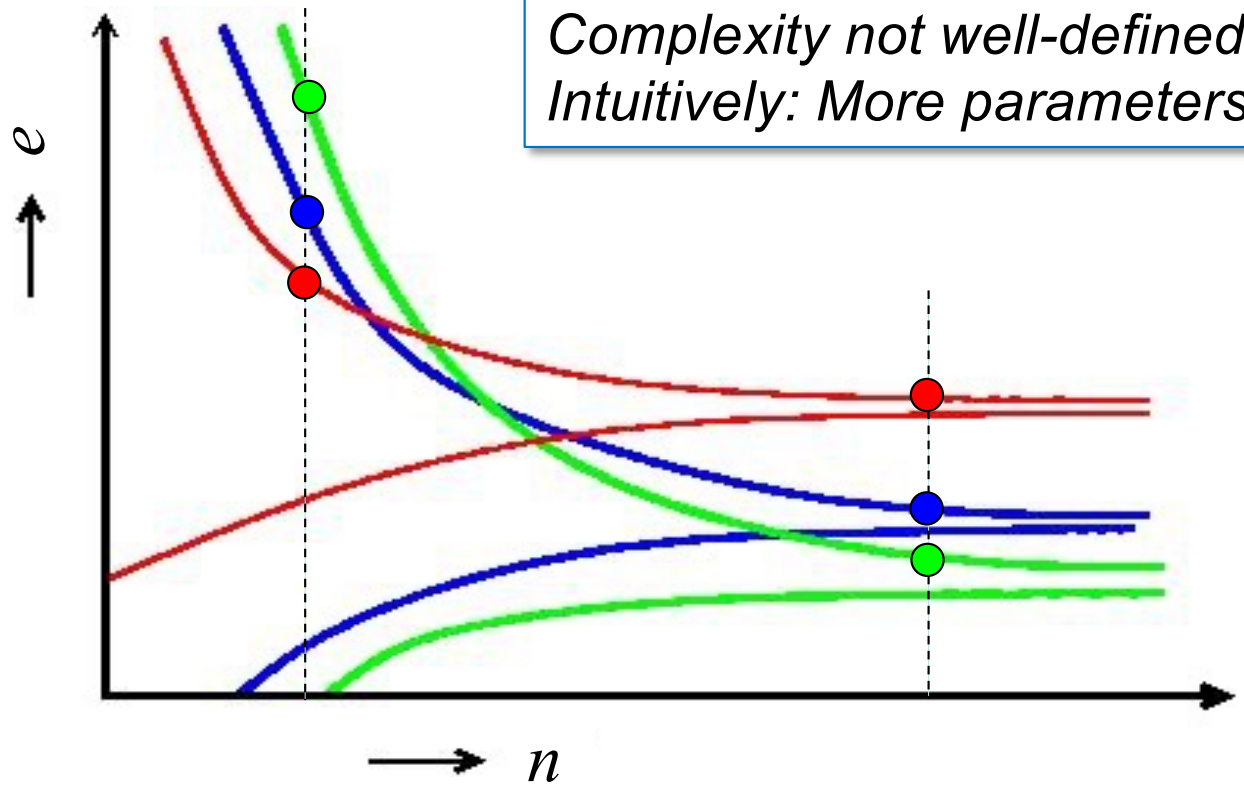


- So why not always use complex classifiers?

BioSB

Classifier complexity

- Optimal complexity depends on sample size



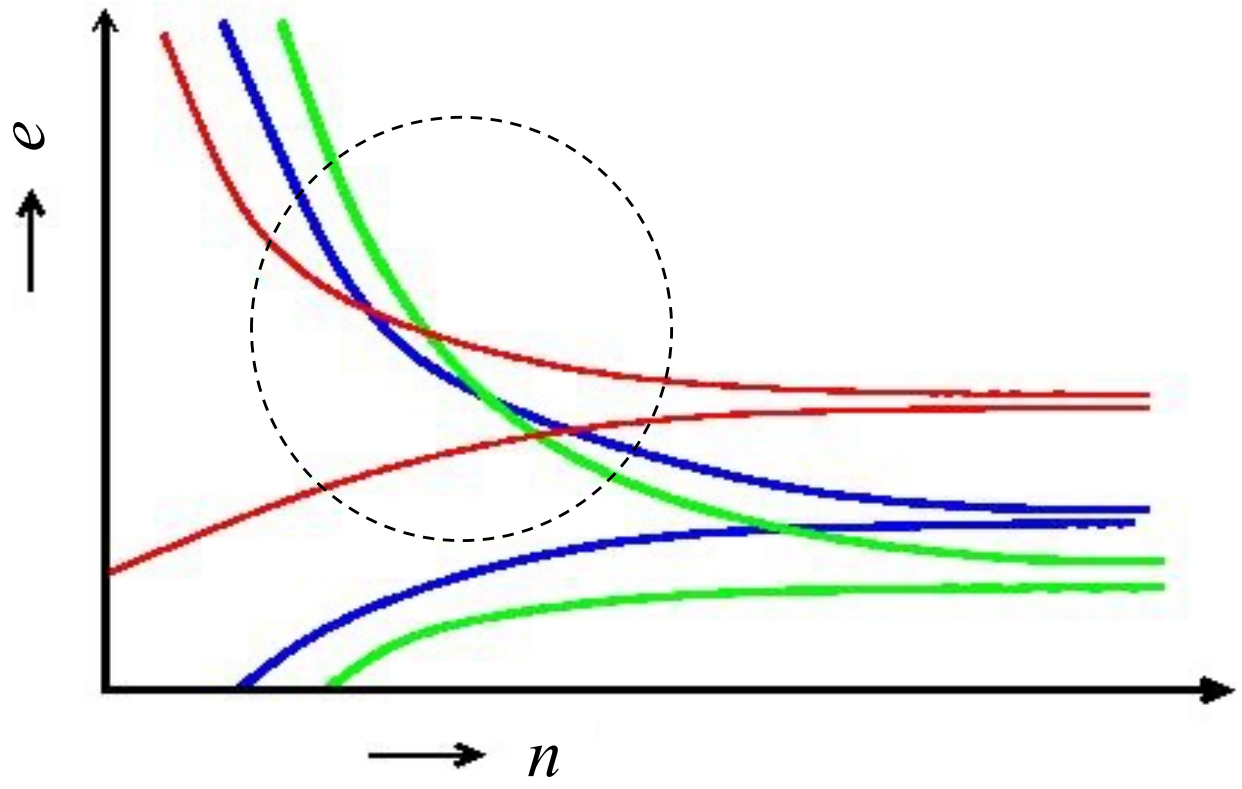
Complexity not well-defined.
Intuitively: More parameters, more complex

- Small: use a simple classifier
- Large: *can* use a complex classifier

BioSB

Classifier complexity (2)

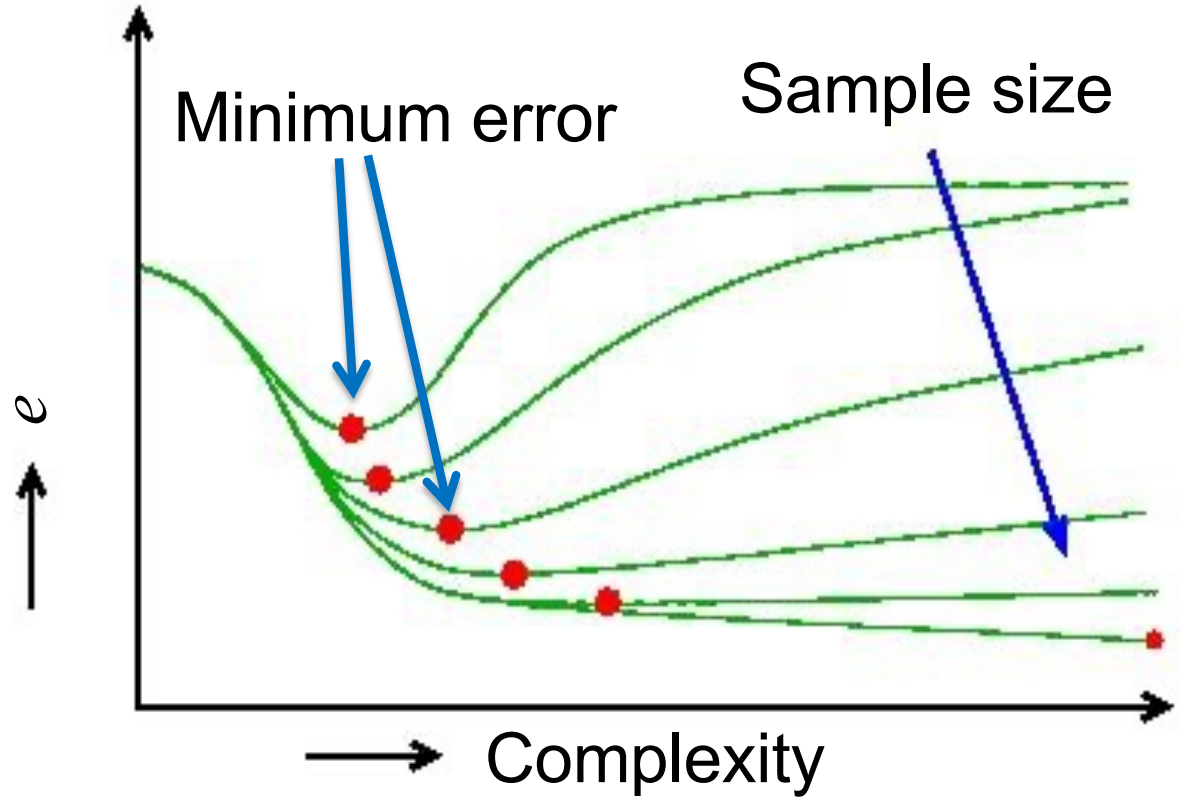
- There is a tradeoff between complexity and training size



BioSB

Classifier complexity (3)

- Remember the curse of dimensionality: for fixed sample size, error increases if classifier complexity increases



BioSB

Classifier complexity (4)

- How to find the best complexity for a given problem?

- Standard approach:
  - Define a large set of classifiers
  - Use cross-validation, and repeatedly
    - Train all the classifiers on the training set
    - Test all the classifiers on the test set
  - Find the best classifier

- This is a lot of work….

BioSB

Regularization

- For many classifiers, it is possible to reduce the complexity of a classifier by adding constraints on the parameters θ
- Often a term is added to the cost function:

$$E = e_A + \lambda f_{reg}(\theta)$$

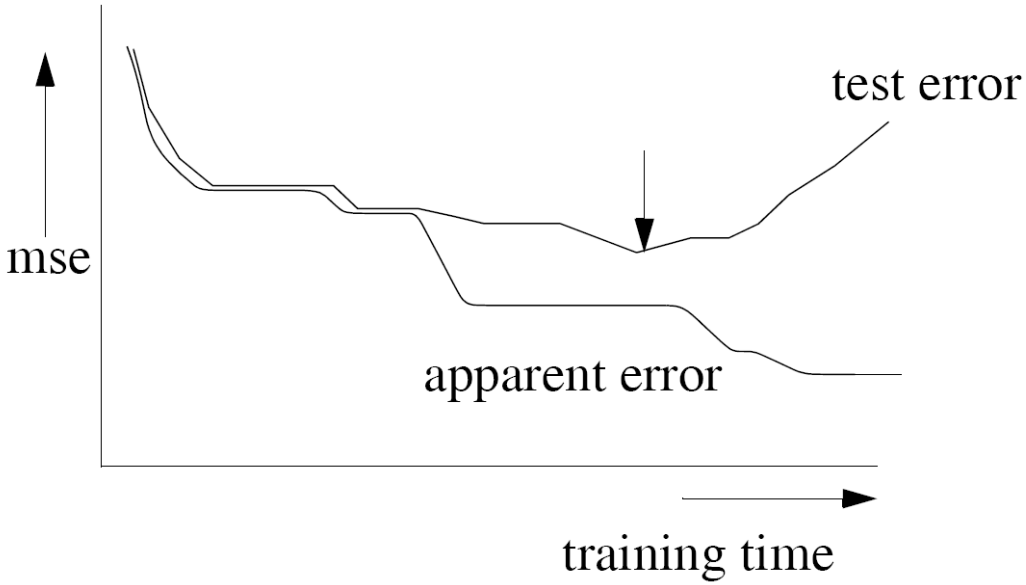
- For example:
  - Multilayer perceptron: $E = \sum_{k=1}^{n} \left| \boldsymbol{t}_k - g(\boldsymbol{x}_k) \right|^2 + \lambda \sum_i w_i^2$
  - Support vector classifier: $E = \frac{1}{2} \left\| \boldsymbol{w} \right\|^2 + C \sum_{i=1}^{n} \xi_i$

BioSB

Regularization (2)

- Another form of regularization: starting with small initial weights in training multilayer perceptrons

- Effective complexity of MLPs increases during training

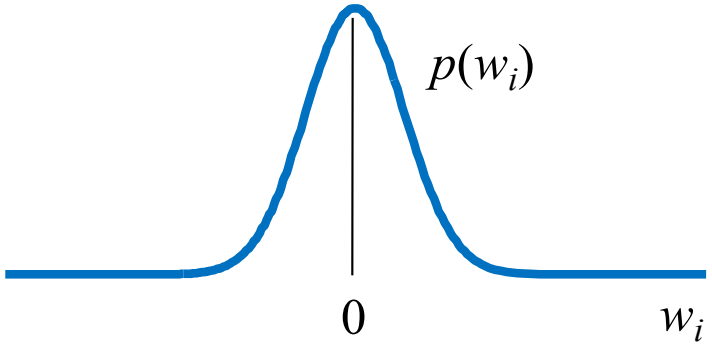


BioSB

Regularization (3)

- Intuitively:
  - Regularization is often a quadratic penalty on weight values
  - Small weights correspond to simple classifier, large weights to complex classifiers
  - This boils down to a *prior* on weights
  - For example:

$$E = \sum_{k=1}^{n} \left| \boldsymbol{t}_k - g(\boldsymbol{x}_k) \right|^2 + \lambda \sum_i w_i^2$$



- Regularization is like Bayesian estimation *on parameters*
- Bayesian model selection: apply Bayesian estimation to entire *models* (classifiers/regressors)

BioSB

Bayesian model selection

- The *evidence* for model M is the probability of data $\boldsymbol{X} = \{\boldsymbol{x}\}$ given model M
- Found by integrating over *all possible values* of parameters $\boldsymbol{\theta}$:

$$p(\boldsymbol{X} \mid M) = \int p(\boldsymbol{X} \mid M, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid M) d\boldsymbol{\theta}$$

- If multiple alternative models are available, use the Bayes factor:

$$\frac{p(\boldsymbol{X} \mid M_1)}{p(\boldsymbol{X} \mid M_2)} > 1 \quad \Rightarrow \quad M_1$$

- We can even take priors on models into account:

$$\frac{p(\boldsymbol{X} \mid M_1)}{p(\boldsymbol{X} \mid M_2)} \frac{p(M_1)}{p(M_2)} > 1 \quad \Rightarrow \quad M_1$$

BioSB

Bayesian model selection (2)

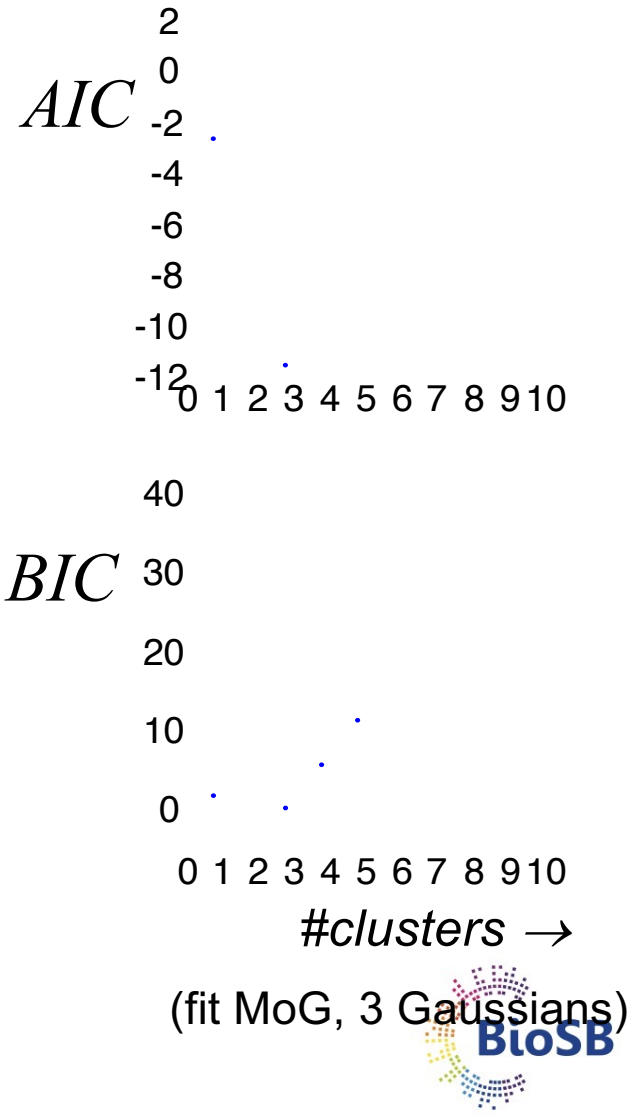
- Integrating over all possible values of θ is very hard in practice
  - Use Monte Carlo methods
  - Use approximations:
    - Akaike Information Criterion:

$$AIC = 2k - 2\log\left[p(\boldsymbol{X} \mid M, \boldsymbol{\theta}_{opt}) \right]$$

    - Bayesian Information Criterion:

$$BIC = k\log(n) - 2\log\left[p(\boldsymbol{X} \mid M, \boldsymbol{\theta}_{opt}) \right]$$

    - k = number of parameters
    - n = number of training objects
    - $\boldsymbol{\theta}_{opt}$ = parameters optimizing likelihood



Recapitulation

- A fundamental trade-off in pattern recognition is between *model descriptiveness* (e.g. classification error) and *model complexity*
- Optimal complexity depends on the problem and sample size, and can be assessed/controlled through:
  - *Cross-validation and learning curves*
  - *Regularization*
  - *Bayesian information criteria*
- More fundamental approaches are:
  - *Bayesian model selection*
  - *Minimum description length*
  - *VC dimension*

  Only the latter leads to a practical solution, the support vector classifier

BioSB

END