Statistical Learning Part 2 Nonparametric Learning: The Main Ideas

R. Marrone, R. Moeller
Hamburg University of Technology

Instance-Based Learning

- So far: Statistical learning as parameter learning
- Given a specific parameter-dependent family of probability models fit it to the data by tweaking parameters
  - Often simple and effective
  - Fixed complexity
  - Maybe good for some problem classes
- Adapting the structure of the hypothesis proved to be very difficult

Instance-Based Learning

- Nonparametric learning methods allow hypothesis complexity to grow with the data
  - "The more data we have, the 'wigglier' the hypothesis can be"

Characteristics

- An instance-based learner is a *lazy-learner* and does all the work when the test example is presented. This is opposed to so-called *eager-learners*, which build a parameterised compact model of the target.
- It produces *local* approximation to the target function (*different* with each test instance)

Nearest Neighbor Classifier

- Basic idea
  - Store all labelled instances (i.e., the training set) and compare new unlabeled instances (i.e., the test set) to the stored ones to assign them an appropriate label.
  - Comparison is performed, for instance, by means of the Euclidean distance, and the labels of the k nearest neighbors of a new instance determine the assigned label
  - Other distance measures: Mahalanobis distance (for multidimensional space), ...
- Parameter: k
  (the number of nearest neighbors)

Nearest Neighbor Classifier

- 1-Nearest neighbor:

  Given a query instance x_q,

  - first locate the nearest training example x_n
  - then $f(x_q) := f(x_n)$

- K-Nearest neighbor:

  Given a query instance x_q,

  - First locate the k nearest training examples
  - If discrete values target function then take vote among its k nearest nbrs
    else if real valued target fct then take the mean of the f values of the k nearest nbrs

$$f(x_q) := \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance Between Examples

- We need a measure of distance in order to know who are the neighbours
- Assume that we have T attributes for the learning problem. Then one example point $\boldsymbol{x}$ has elements $x_t \in \Re,\ t=1,\ldots T.$
- The distance between two points $\boldsymbol{x_i}\,\boldsymbol{x_j}$ is often defined as the Euclidean distance:

$$d(\mathbf{x}_i,\mathbf{x}_j) = \sqrt{\sum_{t=1}^{T}[x_{it} - x_{jt}]^2}$$

Difficulties

- For higher dimensionality, neighborhoods must be large in the average case – *curse of dimensionality*
- There may be irrelevant attributes amongst the attributes
- Have to calculate the distance of the test case from *all* training cases

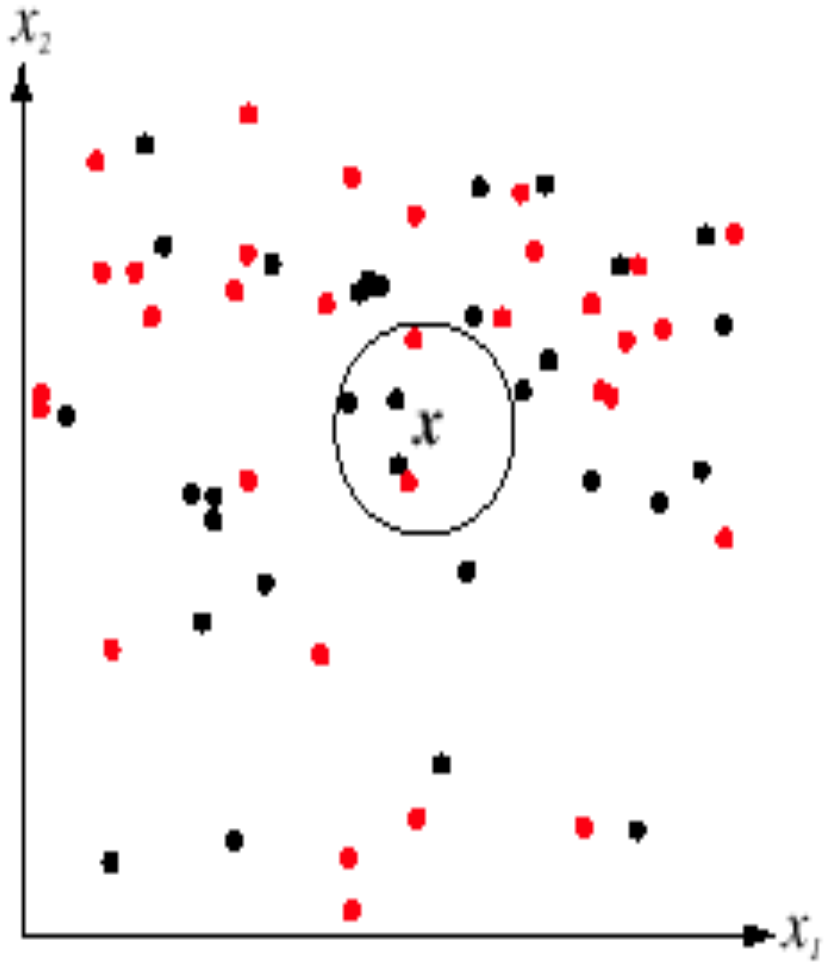
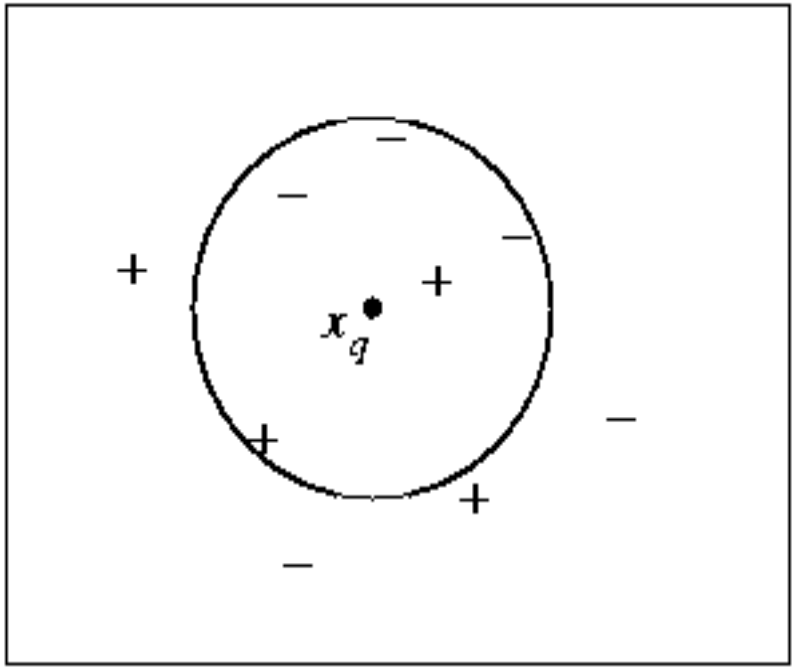
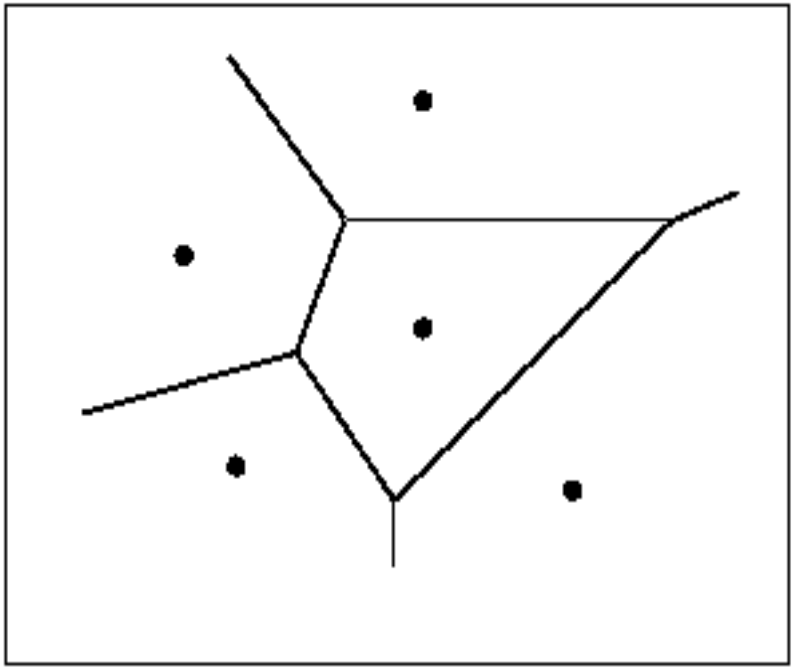


FIGURE 4.15. The k-nearest-neighbor query starts at the test point **x** and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point **x** would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

kNN vs 1NN: Voronoi Diagram





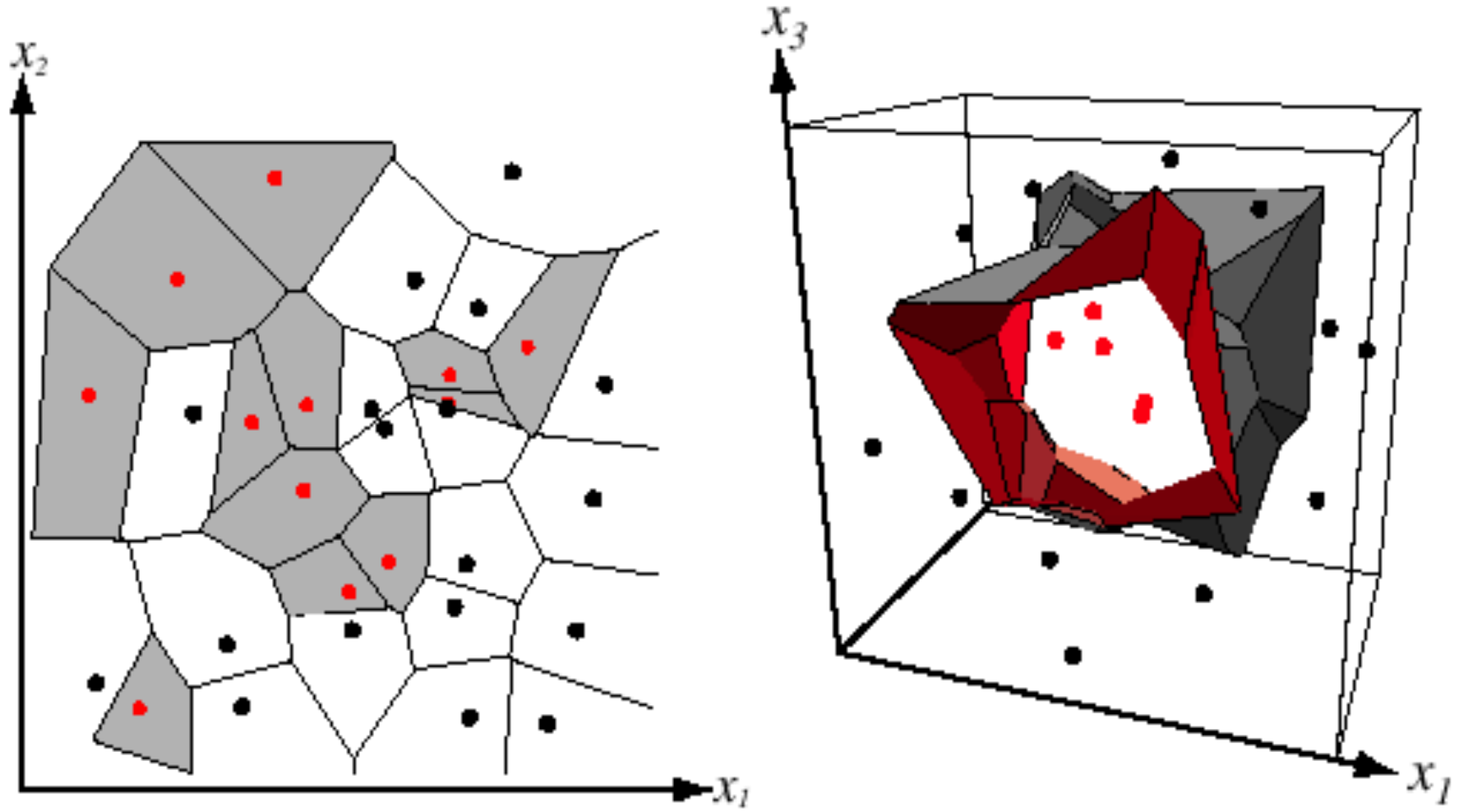
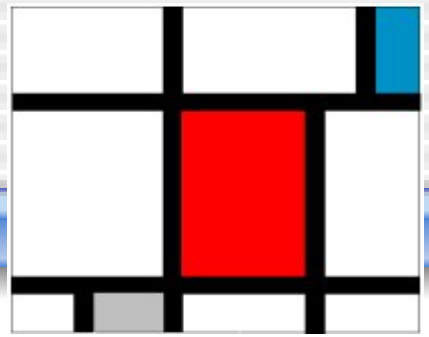


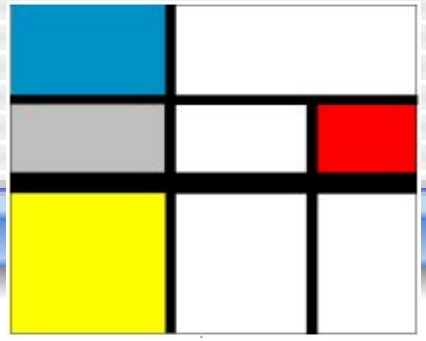
FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

When to Consider kNN Algorithms?

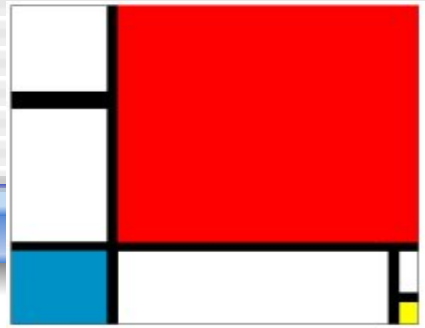
- Instances map to points in $\Re^n$
- Not more then say 20 attributes per instance
- Lots of training data
- Advantages:
  - Training is very fast
  - Can learn complex target functions
  - Don't lose information
- Disadvantages:
  - ? (will see them shortly…)



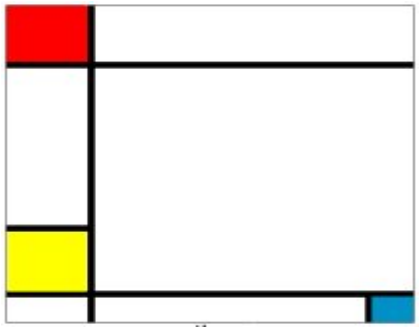
one



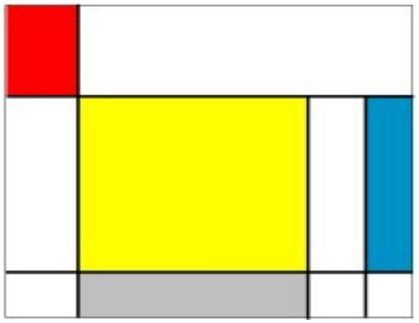
two



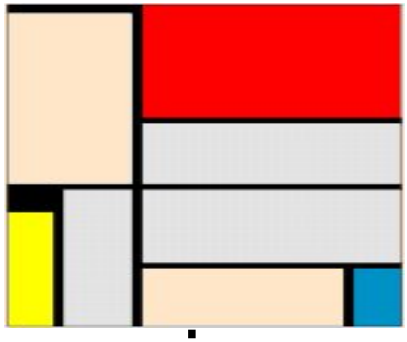
three



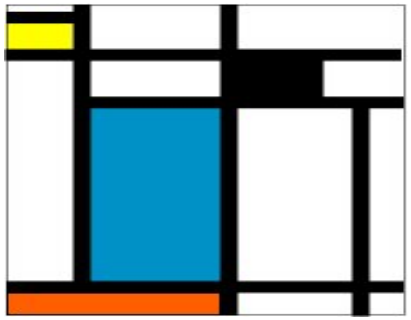
four



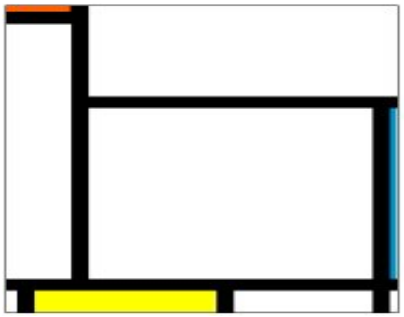
five



six



seven



eight ?

Training data

Number	Lines	Line types	Rectangles	Colours	Mondrian?
1	6	1	10	4	*No*
2	4	2	8	5	*No*
3	5	2	7	4	Yes
4	5	1	8	4	Yes
5	5	1	10	5	*No*
6	6	1	8	6	Yes
7	7	1	14	5	*No*

Test instance

Number	Lines	Line types	Rectangles	Colours	Mondrian?
8	7	2	9	4	

Keep Data in Normalized Form

One way to normalize the data $a_r(x)$ to $a'_r(x)$ is

$$x_t' \equiv \frac{x_t - \bar{x}_t}{\sigma_t}$$

$\bar{x}_t \equiv$ mean of t^{th} attribute

$\sigma_t \equiv$ standard deviation of t^{th} attribute

average distance of the data values from their mean

Mean and standard deviation

If the random variable *X* takes on the values $x_1, \cdots, x_N$ (which are real numbers) with equal probability, then its standard deviation can be computed as follows. First, the mean of *X*, $\overline{x}$, is defined as a summation:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{x_1 + x_2 + \cdots + x_N}{N}$$

where *N* is the number of samples taken. Next, the standard deviation simplifies to

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

Source: Wikipedia

Normalized Training Data

Number	Lines	Line types	Rectangles	Colours	Mondrian?
1	0.632	-0.632	0.327	-1.021	*No*
2	-1.581	1.581	-0.588	0.408	*No*
3	-0.474	1.581	-1.046	-1.021	Yes
4	-0.474	-0.632	-0.588	-1.021	Yes
5	-0.474	-0.632	0.327	0.408	*No*
6	0.632	-0.632	-0.588	1.837	Yes
7	1.739	-0.632	2.157	0.408	*No*

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{T} [x_{it} - x_{jt}]^2}$$

Test instance

Number	Lines	Line types	Rectangles	Colours	Mondrian?
8	1.739	1.581	-0.131	-1.021	

Normalized Training Data

Number	Lines	Line types	Rectangles	Colours	Mondrian?
1	0.632	-0.632	0.327	-1.021	*No*
2	-1.581	1.581	-0.588	0.408	*No*
3	-0.474	1.581	-1.046	-1.021	Yes
4	-0.474	-0.632	-0.588	-1.021	Yes
5	-0.474	-0.632	0.327	0.408	*No*
6	0.632	-0.632	-0.588	1.837	Yes
7	1.739	-0.632	2.157	0.408	*No*

$$\sqrt{(0 + 4{,}89 + 5{,}23 + 2{,}04)} = 3{,}489$$

Test instance

Number	Lines	Line types	Rectangles	Colours	Mondrian?
8	1.739	1.581	-0.131	-1.021	

Distances of Test Instance From Training Data

Example	Distance of test from example	Mondrian?
1	2.517	No
2	3.644	No
3	2.395	Yes
4	3.164	Yes
5	3.472	No
6	3.808	Yes
7	3.490	No

Classification	
1-NN	Yes
3-NN	Yes
5-NN	No
7-NN	No

What if the target function is real valued?

- The k-nearest neighbor algorithm would just calculate the mean of the k nearest neighbours

Variant of kNN: Distance-Weighted kNN

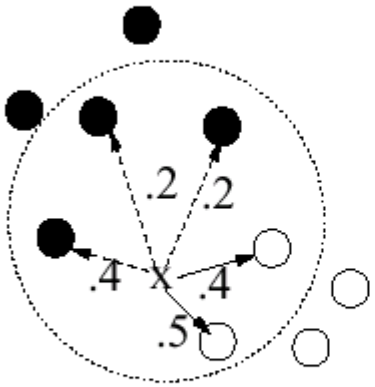
- We might want to weight nearer neighbors more heavily

$$f(\mathbf{x}_q) := \frac{\sum_{i=1}^{k} w_i f(\mathbf{x}_i)}{\sum_{i=1}^{k} w_i} \text{ where } w_i = \frac{1}{d(\mathbf{x}_q, \mathbf{x}_i)^2}$$

- Then it makes sense to use *all training examples* instead of just k

Variant of kNN: Distance-Weighted kNN

k-NN using a weighted-sum voting scheme



kNN (k = 5)

Assign "white" to x because the weighted sum of "whites" is larger then the sum of "blacks".

Each neighbor is given a weight according to its nearness.

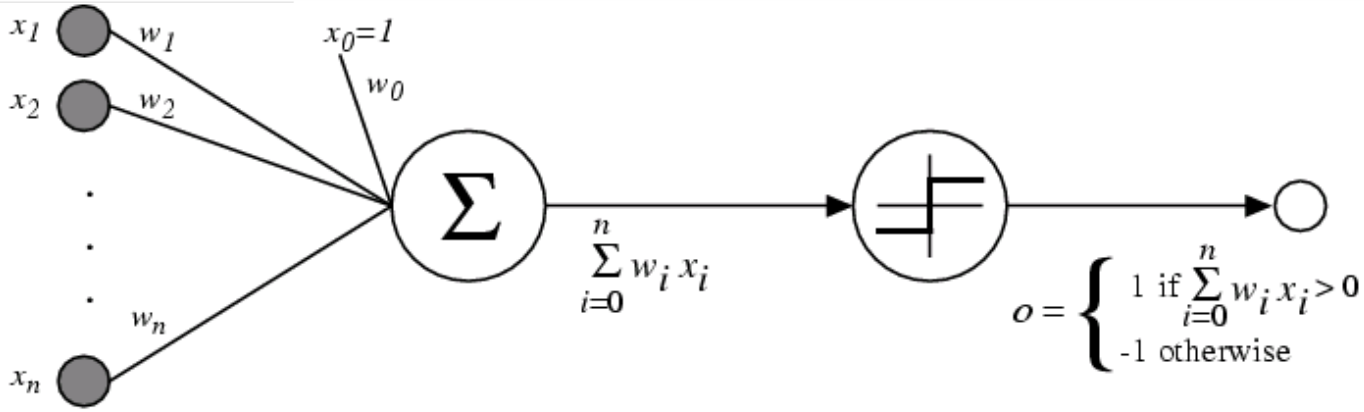
Remarks

- Very simple approach
- Behaves well if data cannot be easily separated
- Rank 7 of top 10 data mining algorithms
- In 1993 outperformed all others in handwritten digit recognition
- In 1994 outperformed all in land usage recognition

Neural Networks

- Feed-forward networks
- Single-layer networks (Perceptrons)
  - Perceptron learning rule
  - Easy to train
    - Fast convergence, few data required
  - Cannot learn „complex“ functions
- Multi-Layer networks
  - Backpropagation learning
  - Hard to train
    - Slow convergence, many data required

Perceptron

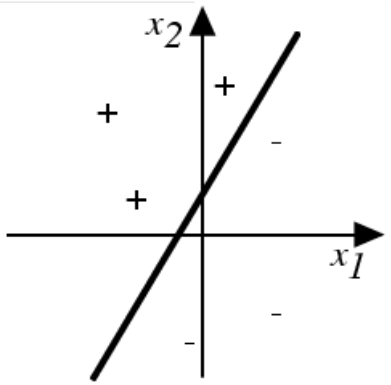


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

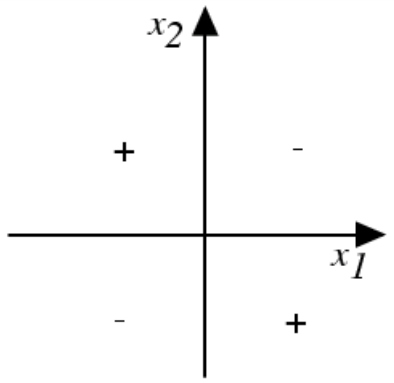
Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Decision Surface of a Perceptron



(a)



(b)

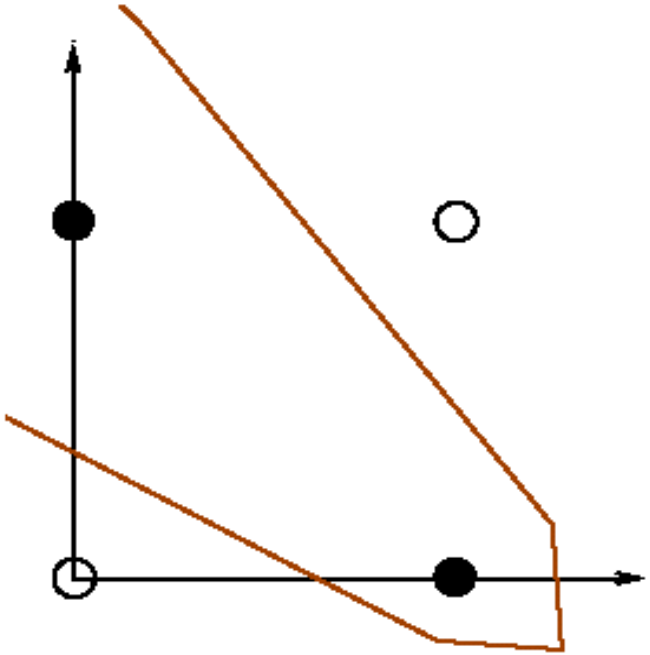
Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable

XOR problem



Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

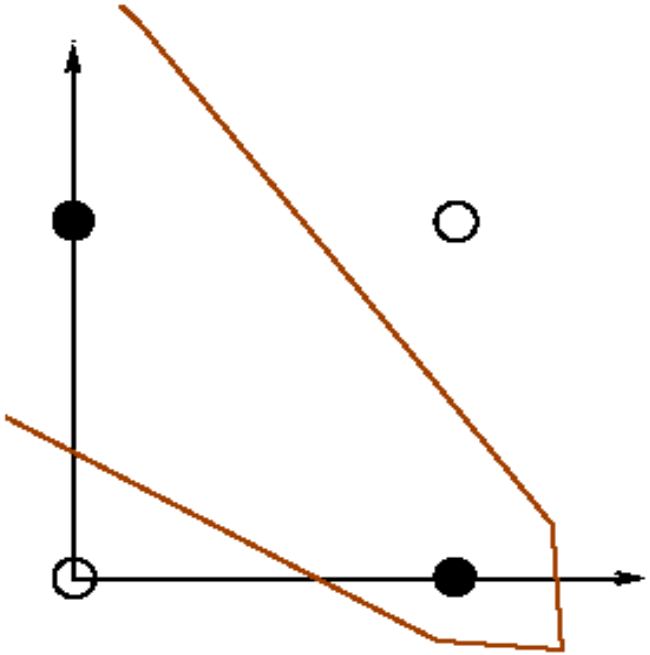
- $y = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Perceptron training rule

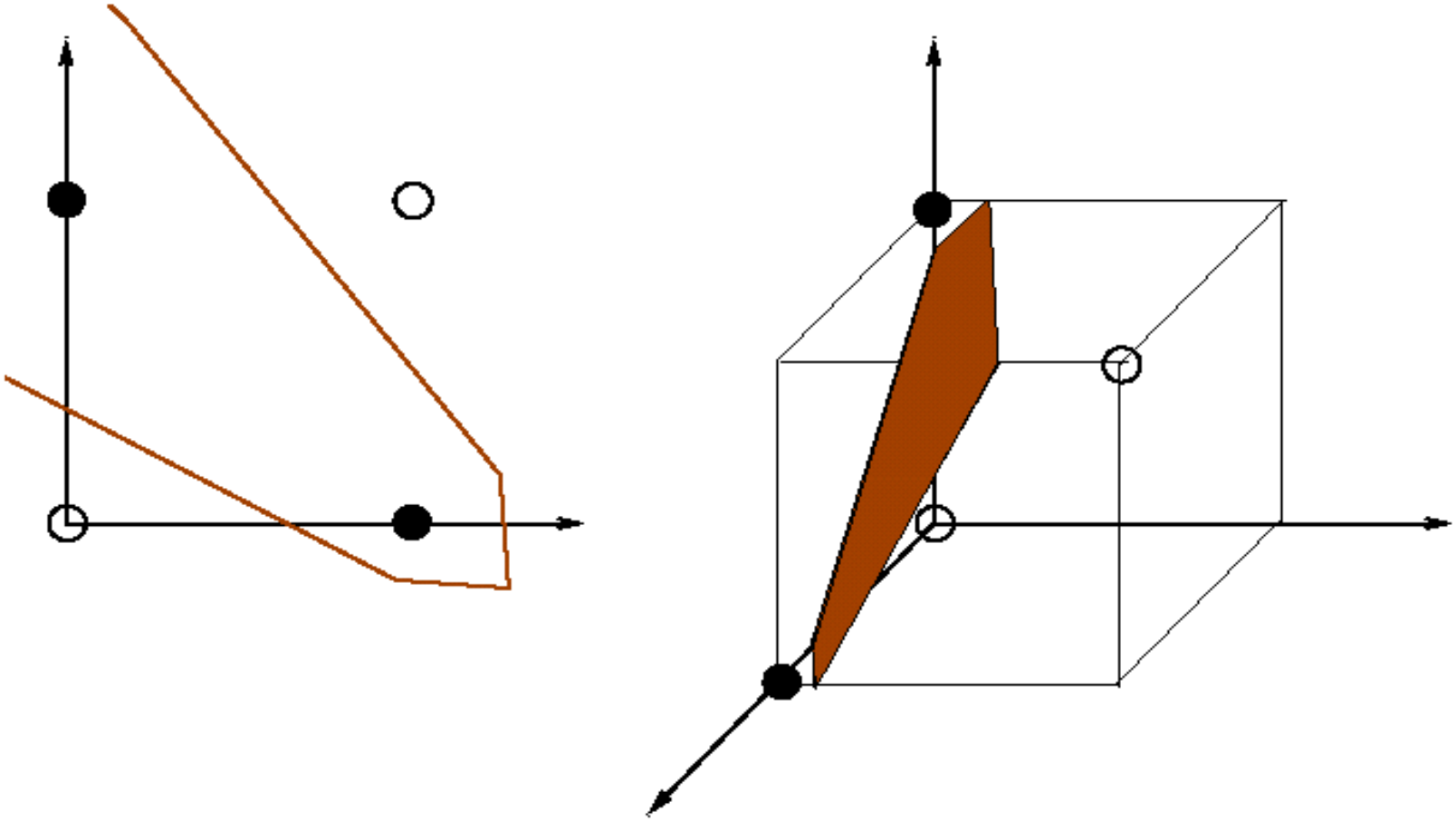
Can prove it will converge

- If training data is linearly separable
- and η sufficiently small *(learning rate)*

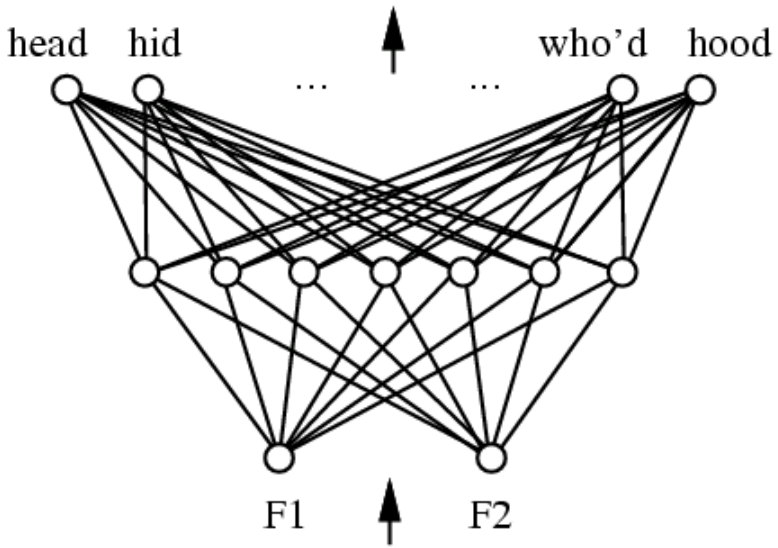
XOR problem

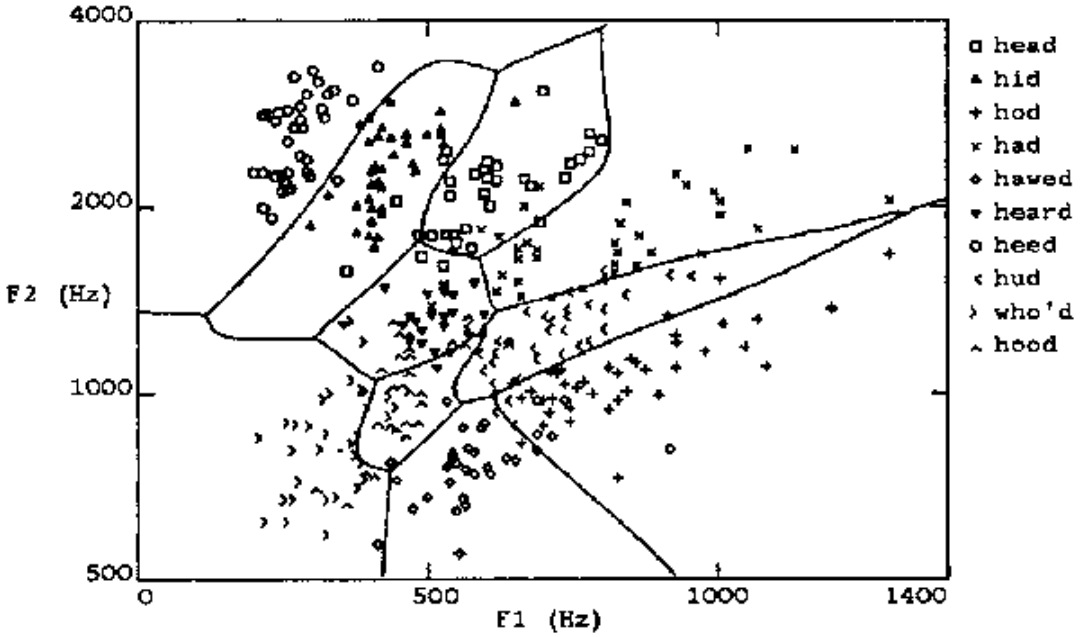


XOR problem

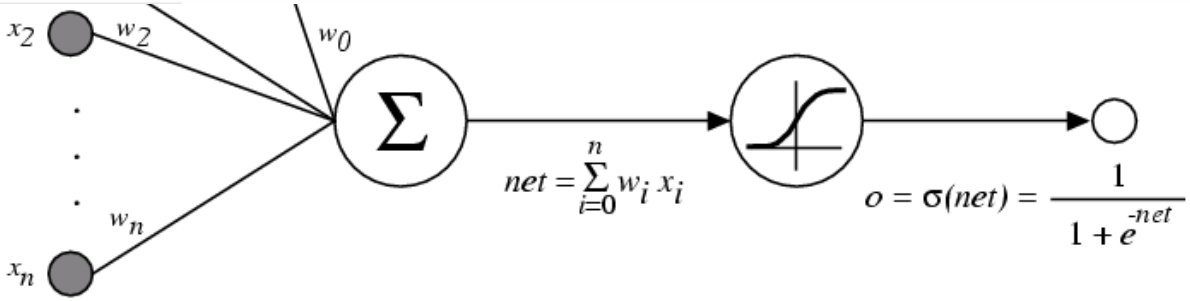


Multilayer Networks of Sigmoid Units





Sigmoid Unit



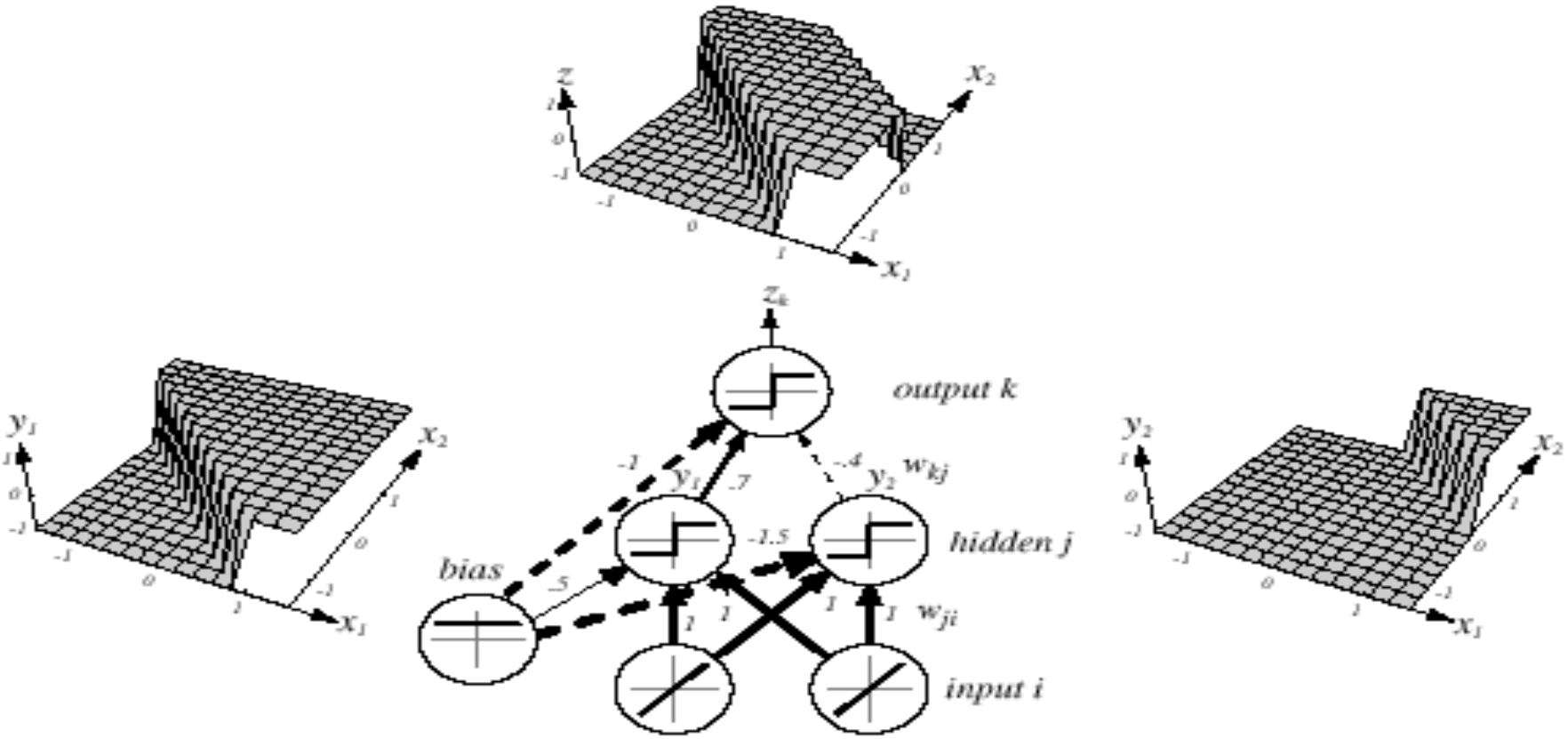
$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

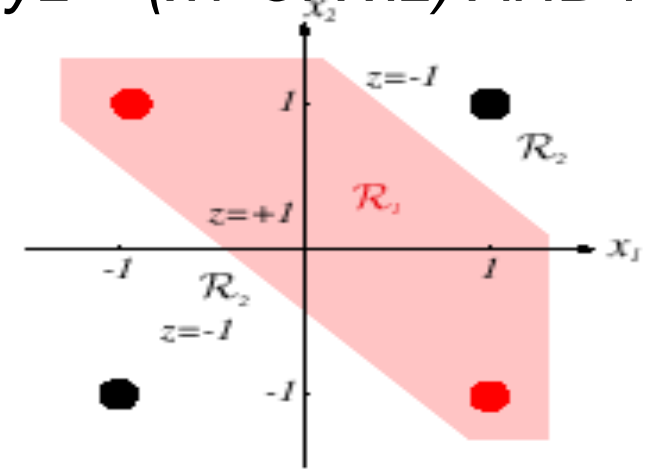
Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation



Z = y1 AND NOT y2 = (x1 OR x2) AND NOT(x1 AND x2)



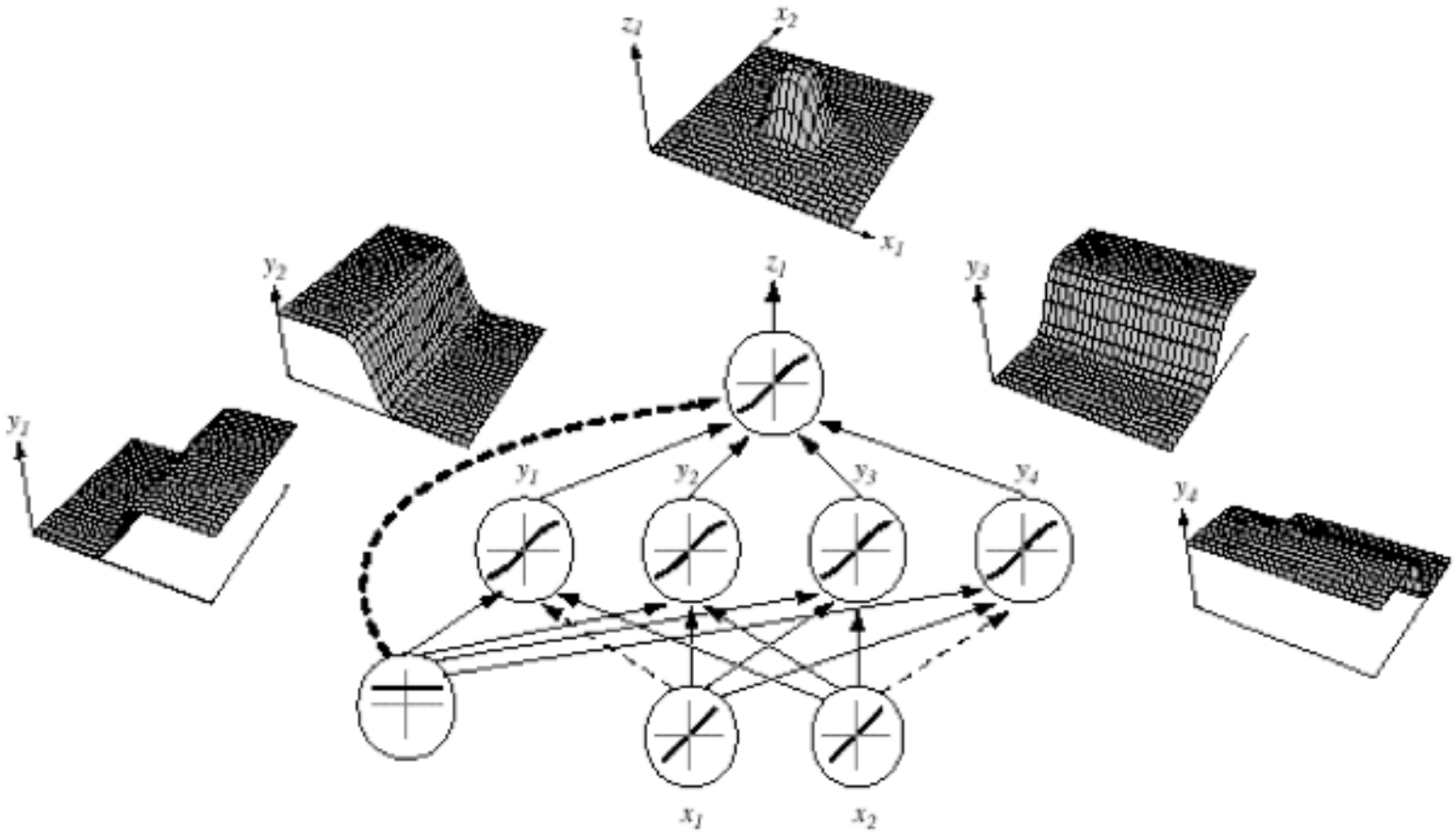


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

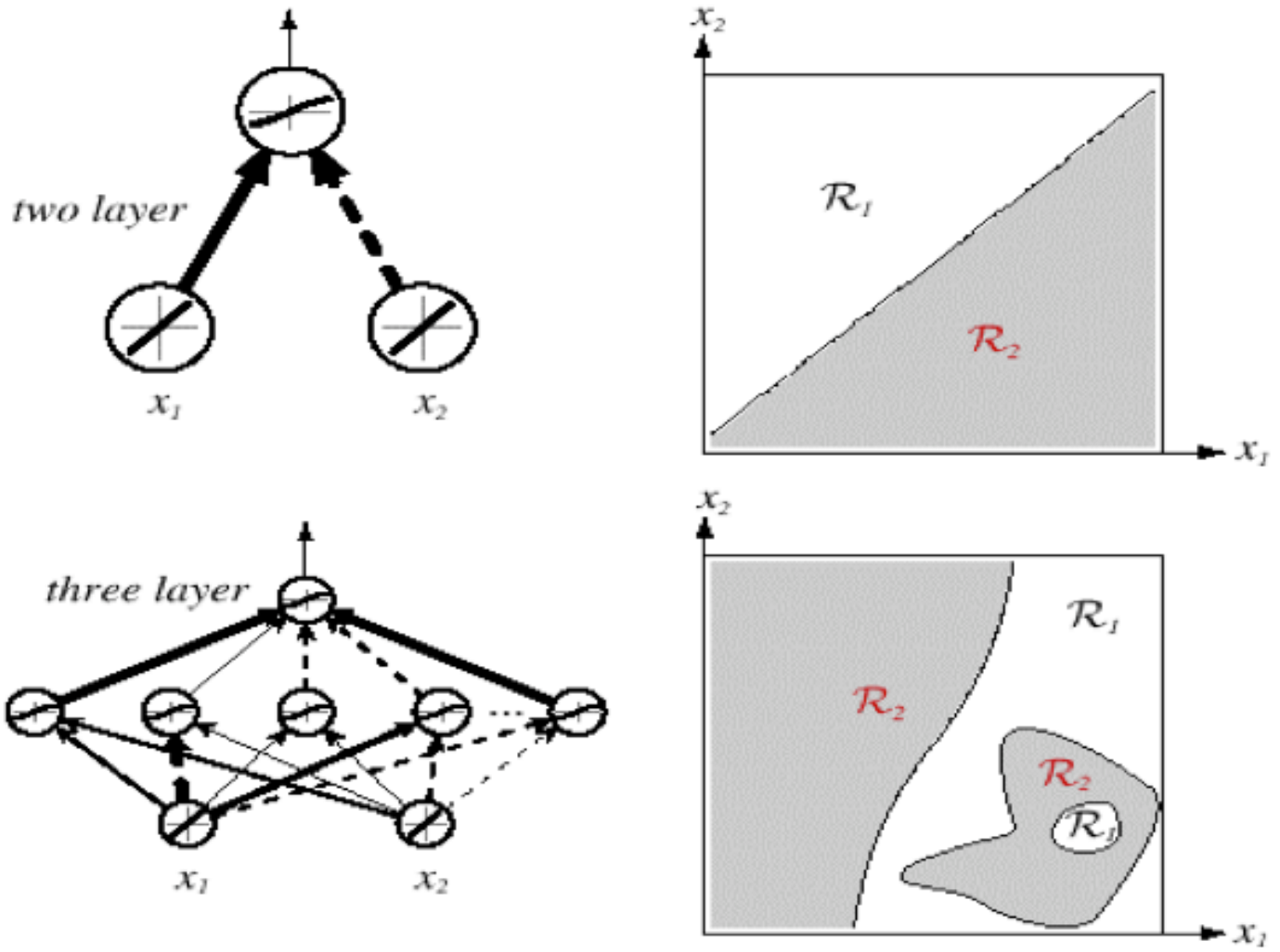
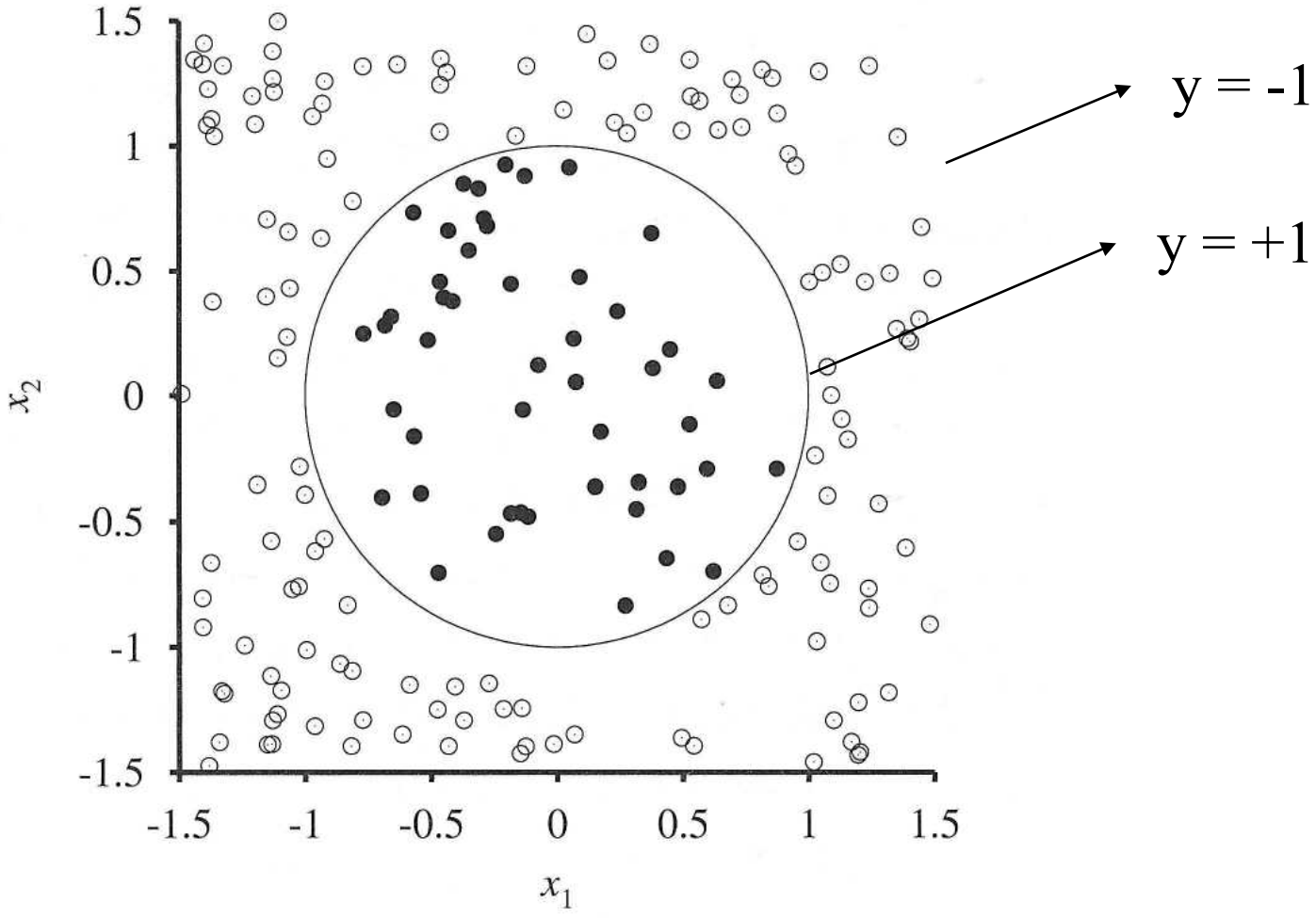


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

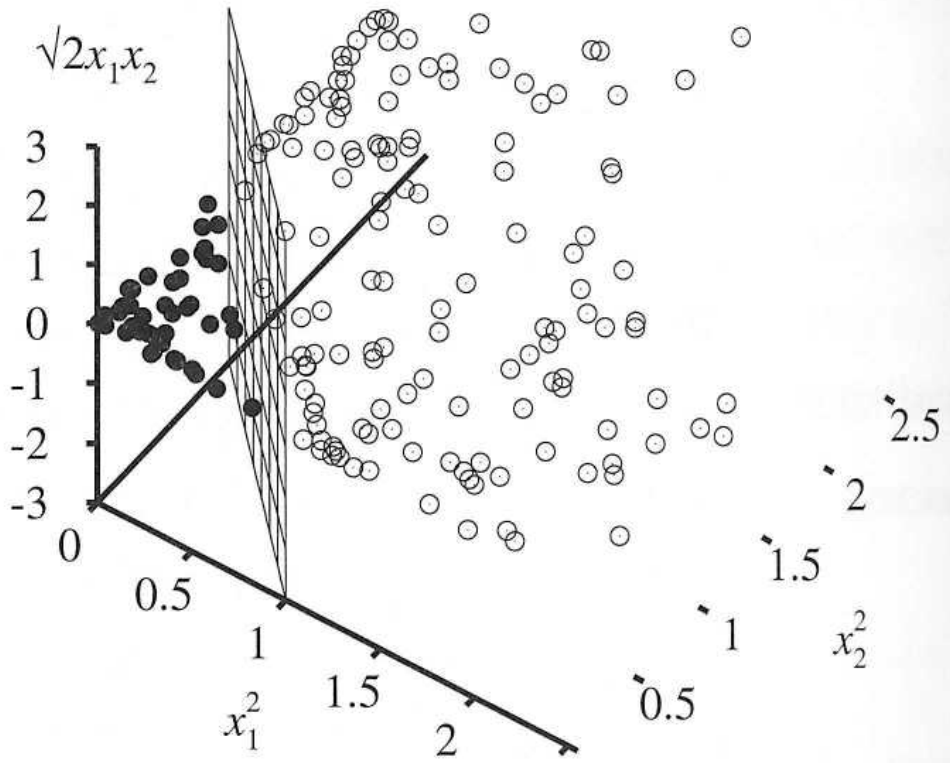
Support Vector Machine Classifier

- Basic idea
  - Mapping the instances from the two classes into a space where they become linearly separable. The mapping is achieved using a kernel function that operates on the instances near to the margin of separation.
- Parameter: kernel type

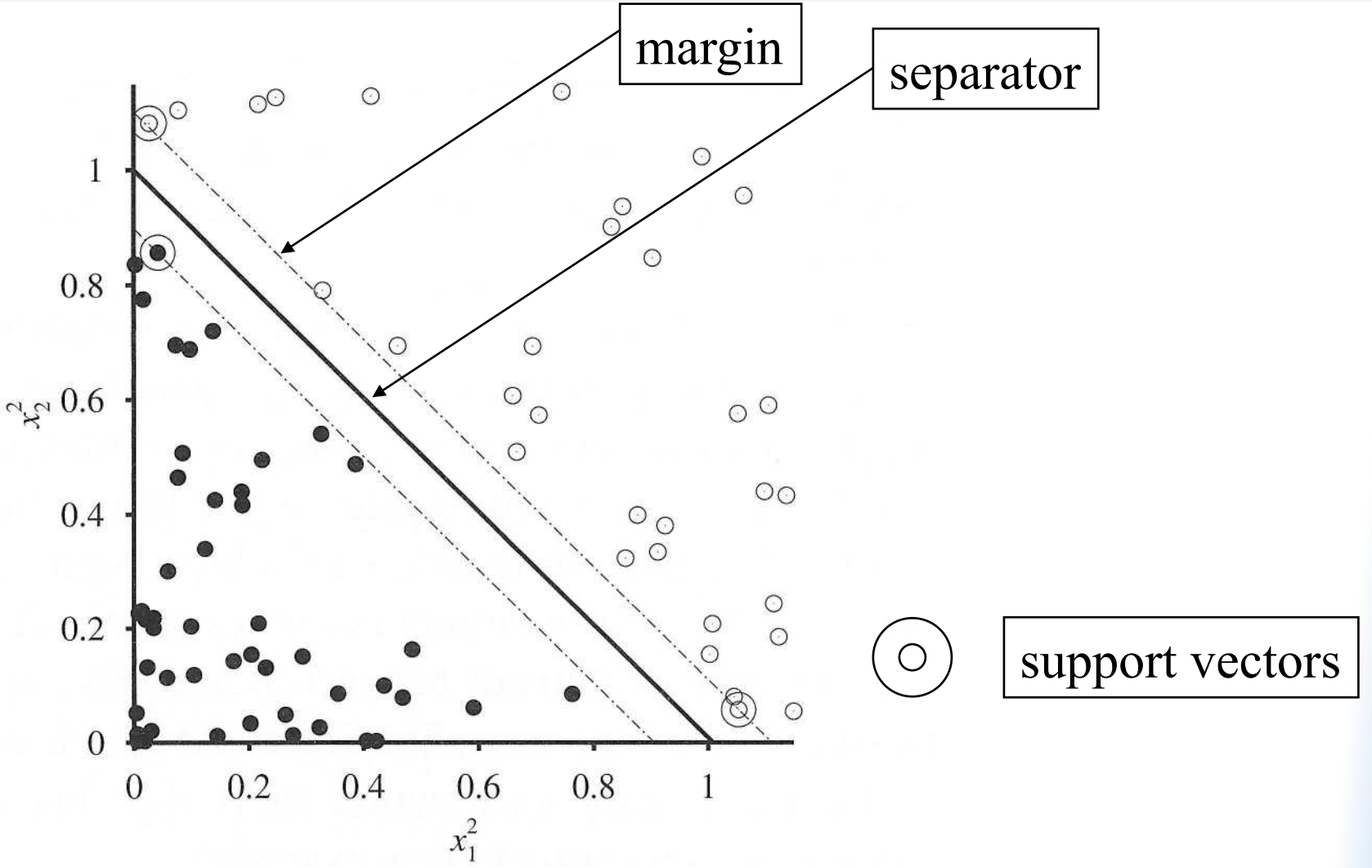
Nonlinear Separation



$$(x_1^2, x_2^2, \sqrt{2x_1x_2})$$



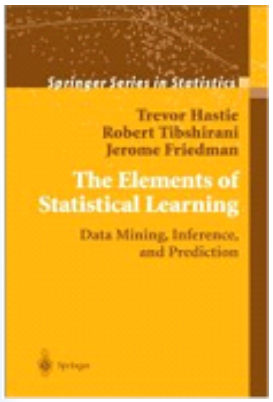
Support Vectors



Literature

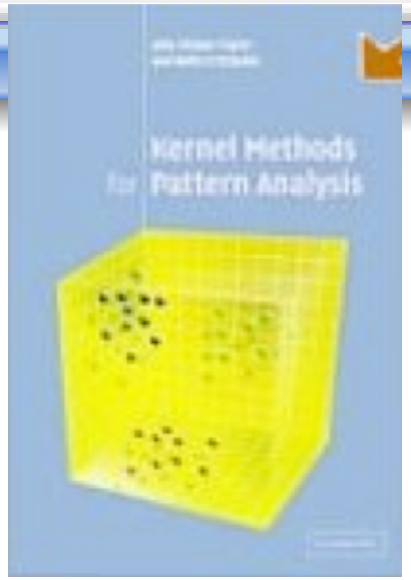
Mitchell (1989). Machine Learning. http://www.cs.cmu.edu/~tom/mlbook.html

Duda, Hart, & Stork (2000). Pattern Classification. http://rii.ricoh.com/~stork/DHS.html

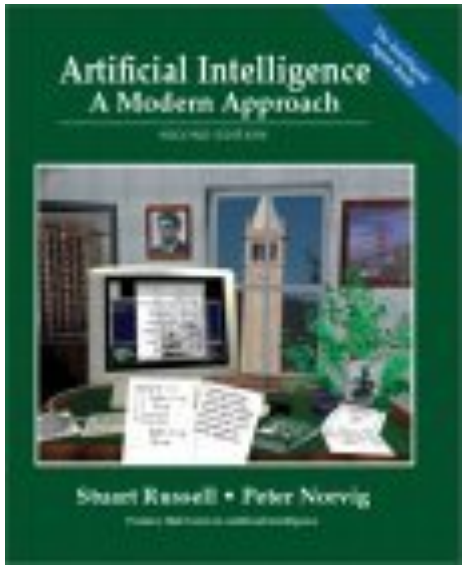


Hastie, Tibshirani, & Friedman (2001). The Elements of Statistical Learning. http://www-stat.stanford.edu/~tibs/ElemStatLearn/

Literature (cont.)



Shawe-Taylor & Cristianini. Kernel Methods for Pattern Analysis.
http://www.kernel-methods.net/



Russell & Norvig (2004). Artificial Intelligence.
http://aima.cs.berkeley.edu/