Seminar on Computational Intelligence

Organisational matters

- Introducing Lectures
- Seminar paper or project
- Oral presentation
- HIS-registration deadline: 6 May 2024
- Project: teamwork possible, max. 4 students per team
- Paper: one student per task

Organisational matters

- Seminar paper or project
- Issue date: 30 April 2024

- Paper
- Processing time: 6 weeks
- Paper and PowerPoint submission: 11 June 2024
- Paper presentation: June/July 2024, select a time slot in at the CampUAS course page

- Project
- Project submission: 30 September 2024

Organisational matters

HIS-registration deadline:

6 May 2024

Computational Intelligence

Computational Intelligence (CI) is the theory, design, application and development of biologically and linguistically motivated **computational** paradigms. Traditionally the three main pillars of CI have been Neural Networks, Fuzzy Systems and Evolutionary **Computation**.

IEEE

Computational Intelligence

- Machine Learning
- Artificial neural networks
- Deep Learning
- Deep convolutional networks
- Ambient intelligence
- Artificial life
- Cultural learning
- Artificial endocrine networks
- Social reasoning
- Artificial hormone networks
- Evolutionary computation
- Fuzzy logic
- Swarm intelligence

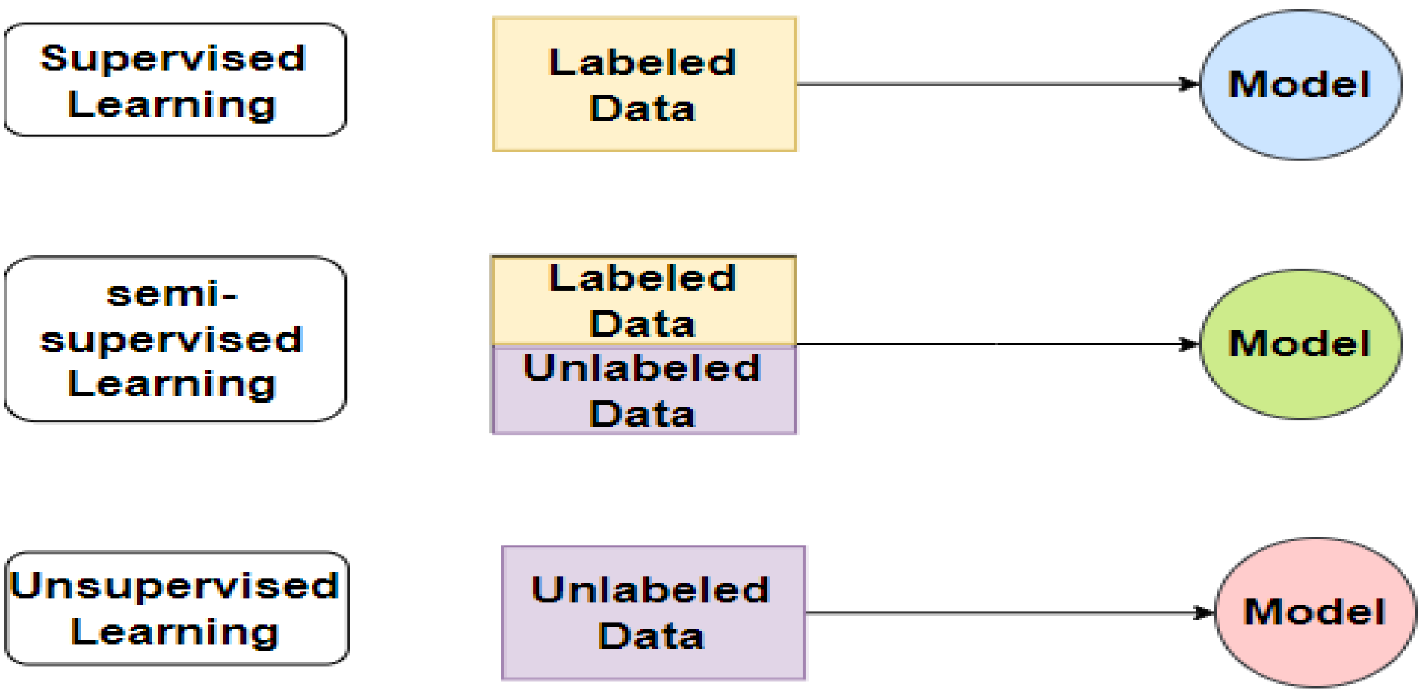
Computational Intelligence

Reference (Quelle): https://www.youtube.com/watch?v=aKed5FHzDTw&t=34s&index=5&list=WL

Computational Intelligence

- Machine Learning
- Artificial neural networks
- Deep Learning
- Deep convolutional networks
- Ambient intelligence
- Artificial life
- Cultural learning
- Artificial endocrine networks
- Social reasoning
- Artificial hormone networks
- Evolutionary computation
- Fuzzy logic
- Swarm intelligence

Machine Learning



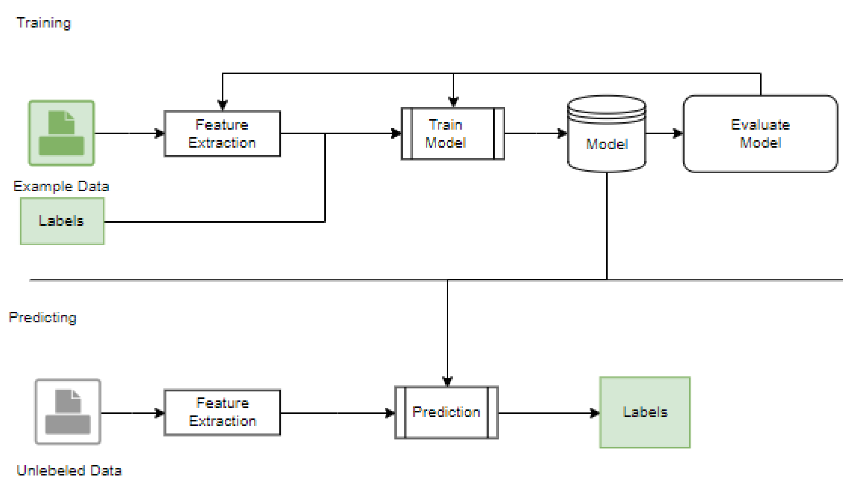
Machine Learning

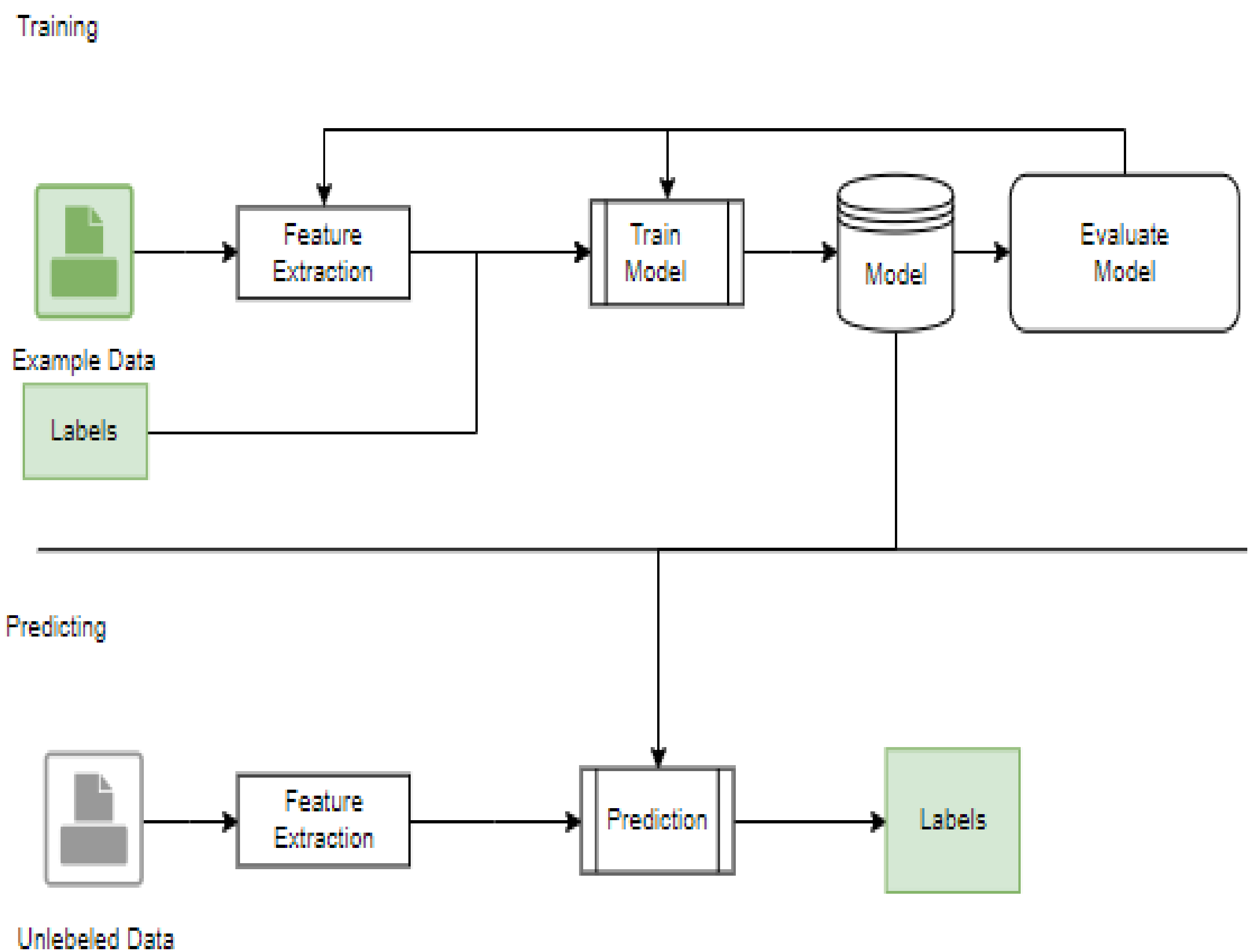
- Supervised learning:

  Given training set $\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$ with $x_i \in X$ and $y_k \in Y$ distributed independently and identically.

  X: set of examples

  Y: set of corresponding labels

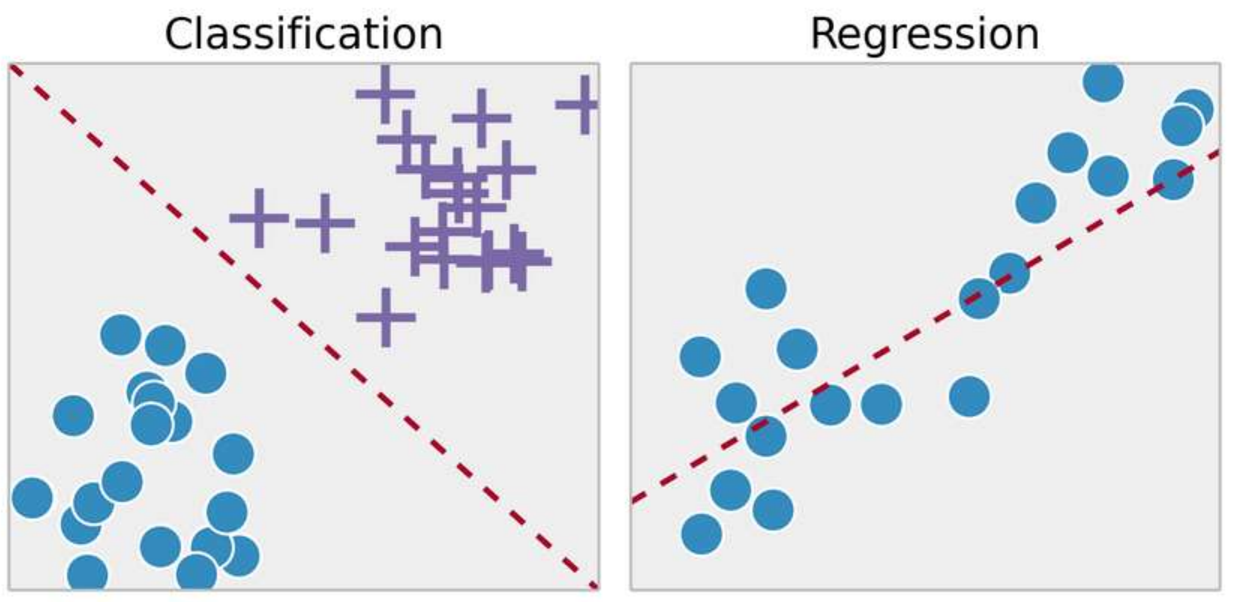



Training
Feature Extraction
Train Model
Model
Evaluate Model
Example Data
Labels
Predicting
Feature Extraction
Prediction
Labels
Unlebeled Data

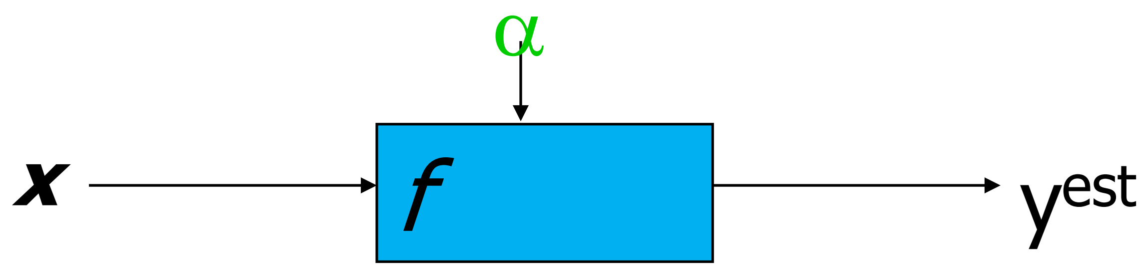
Machine Learning

- Supervised learning examples:

e.g. Naive Bayes classifier, linear regression, corresponding mathematics: next lecture

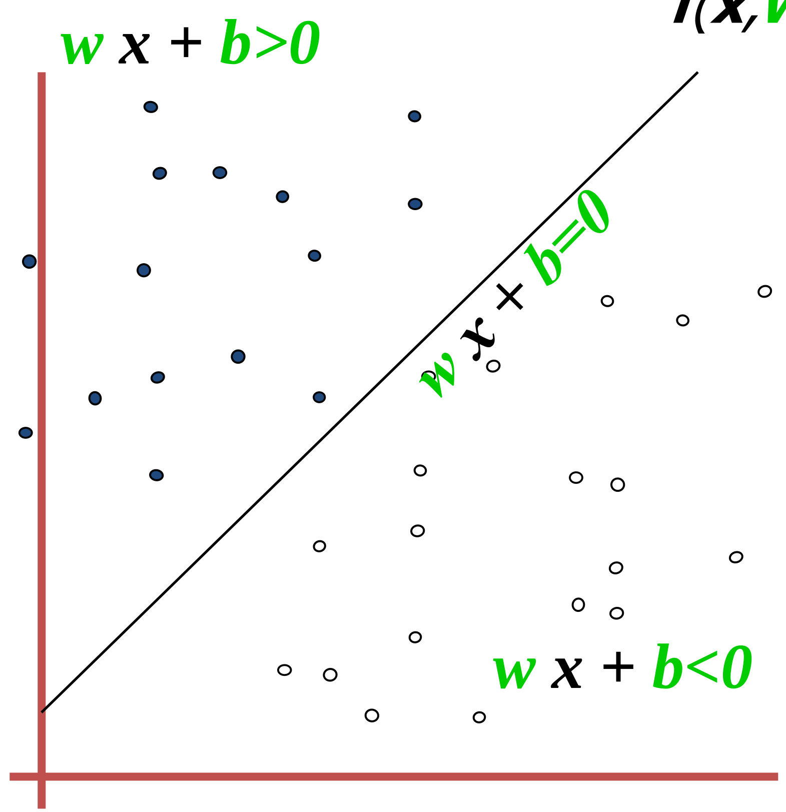


Machine Learning Example: SVM



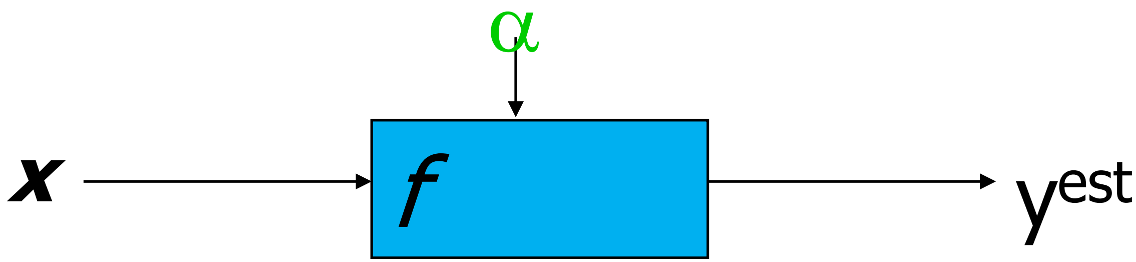
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\ \mathbf{x} + b)$

- • denotes +1
- ◦ denotes -1



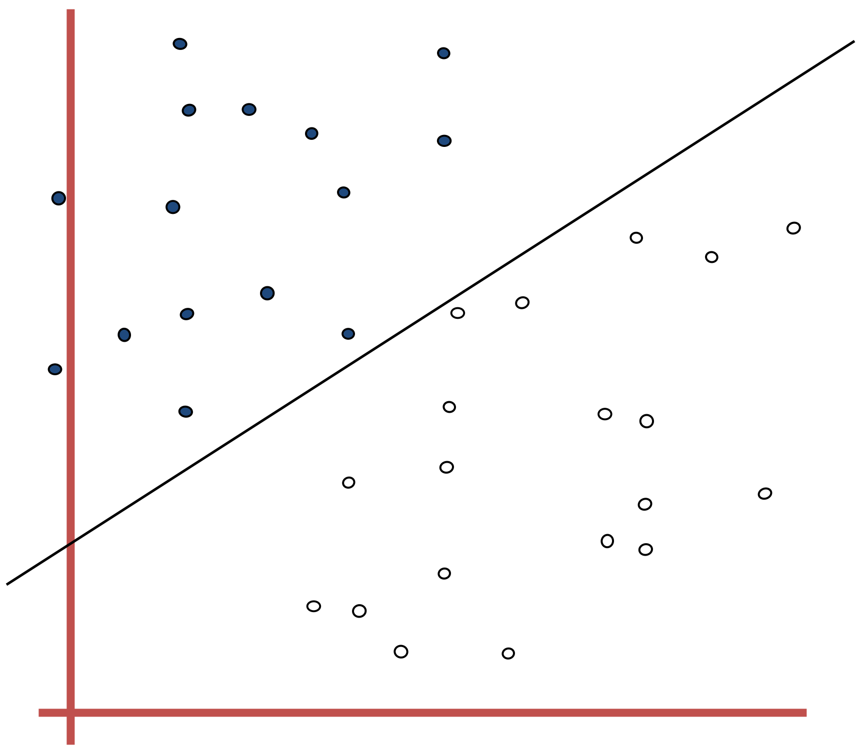
How would you classify this data?

Machine Learning Example: SVM



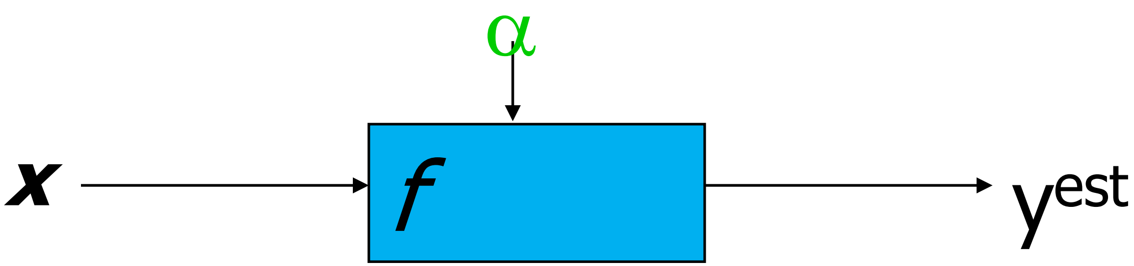
$f(x, w, b) = sign(w\ x + b)$

- denotes +1
- denotes -1

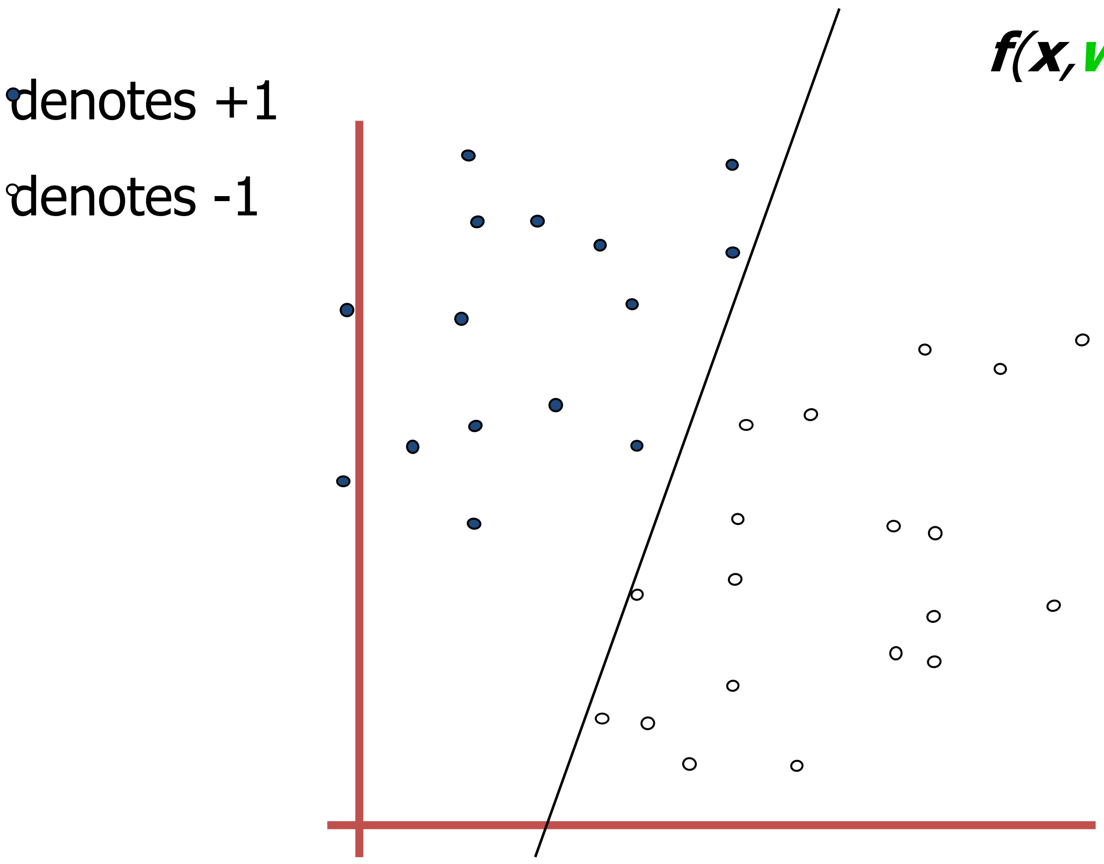


How would you classify this data?

Machine Learning Example: SVM

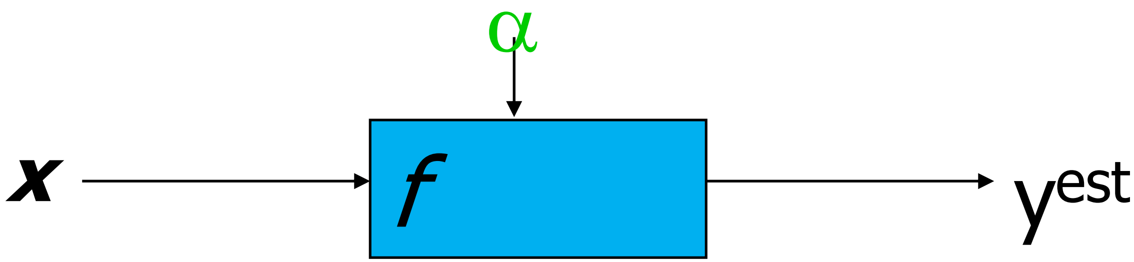


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

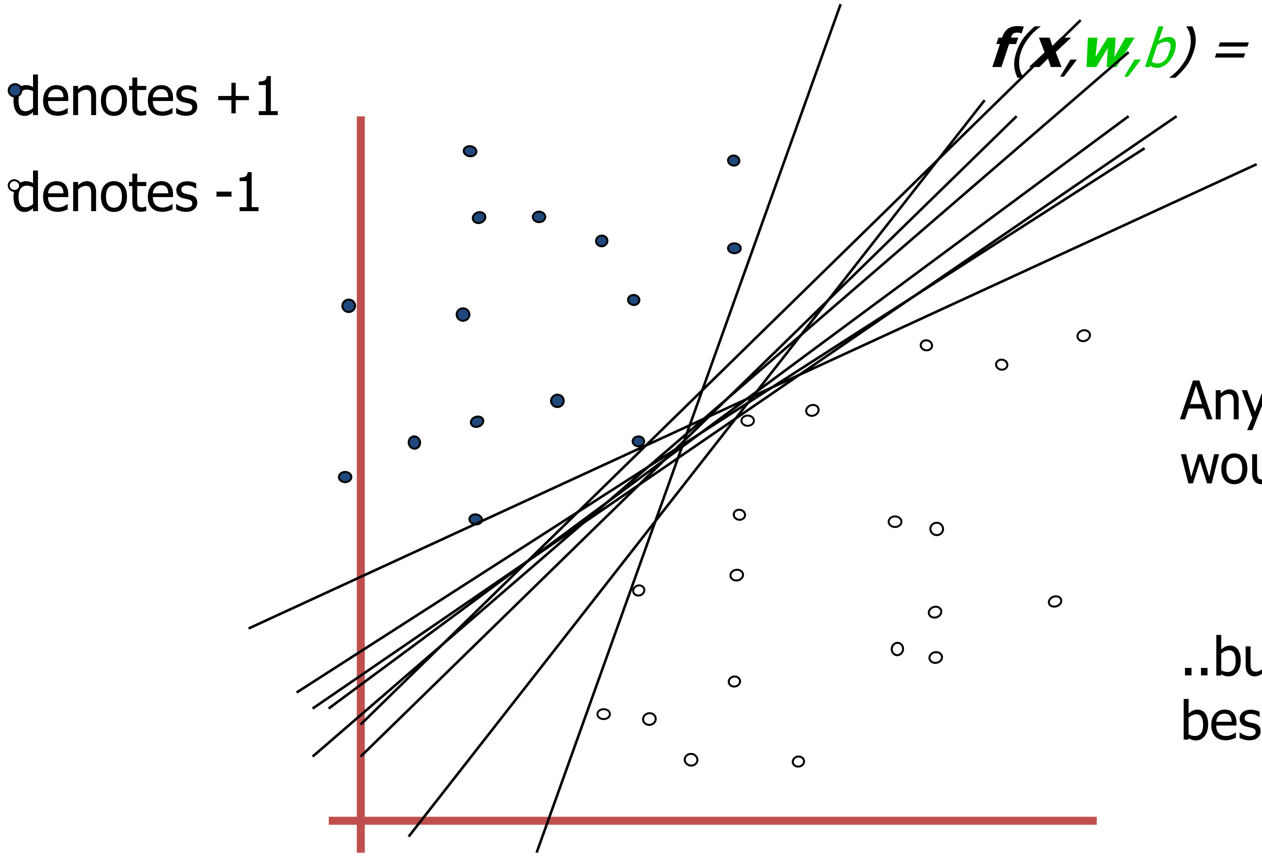


How would you classify this data?

Machine Learning Example: SVM



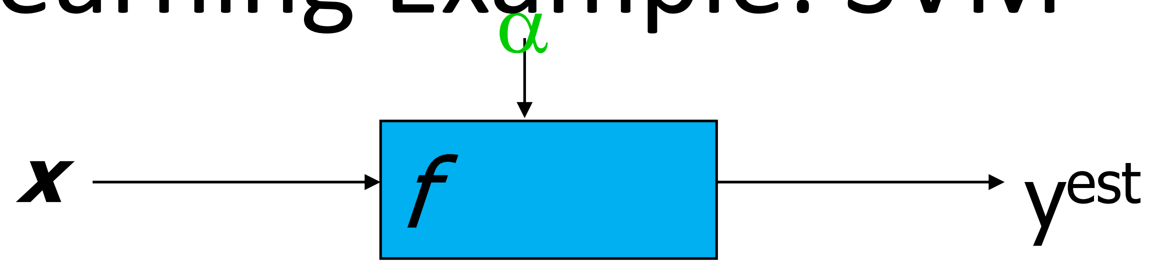
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$



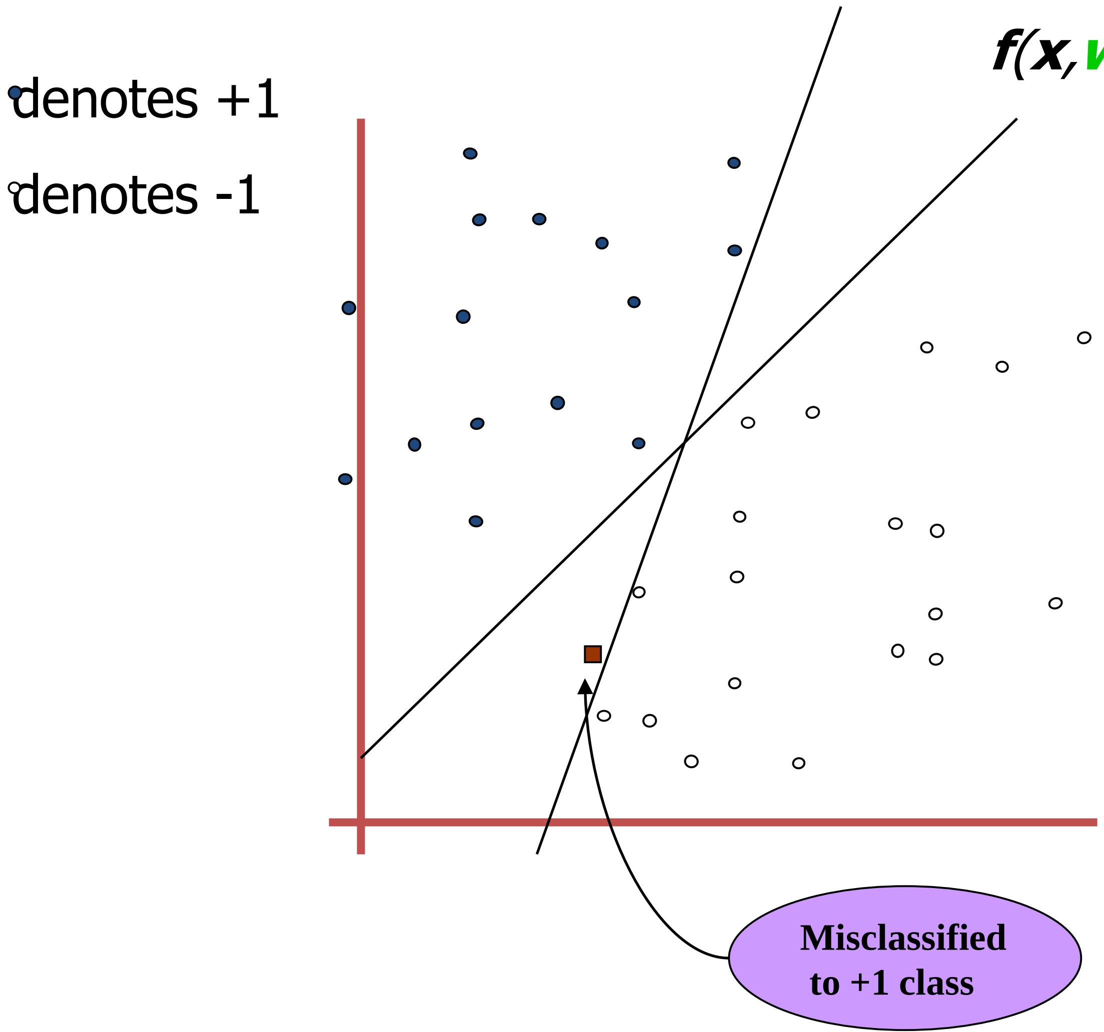
Any of these would be fine..

..but which is best?

Machine Learning Example: SVM



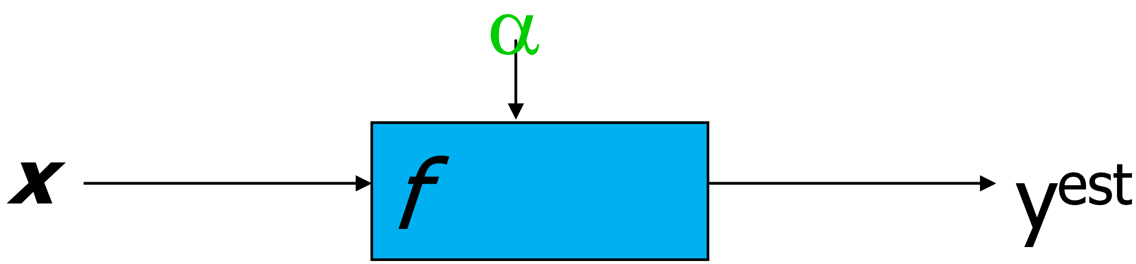
$f(\boldsymbol{x}, \boldsymbol{w}, b) = sign(\boldsymbol{w}\ \boldsymbol{x} + b)$



How would you classify this data?

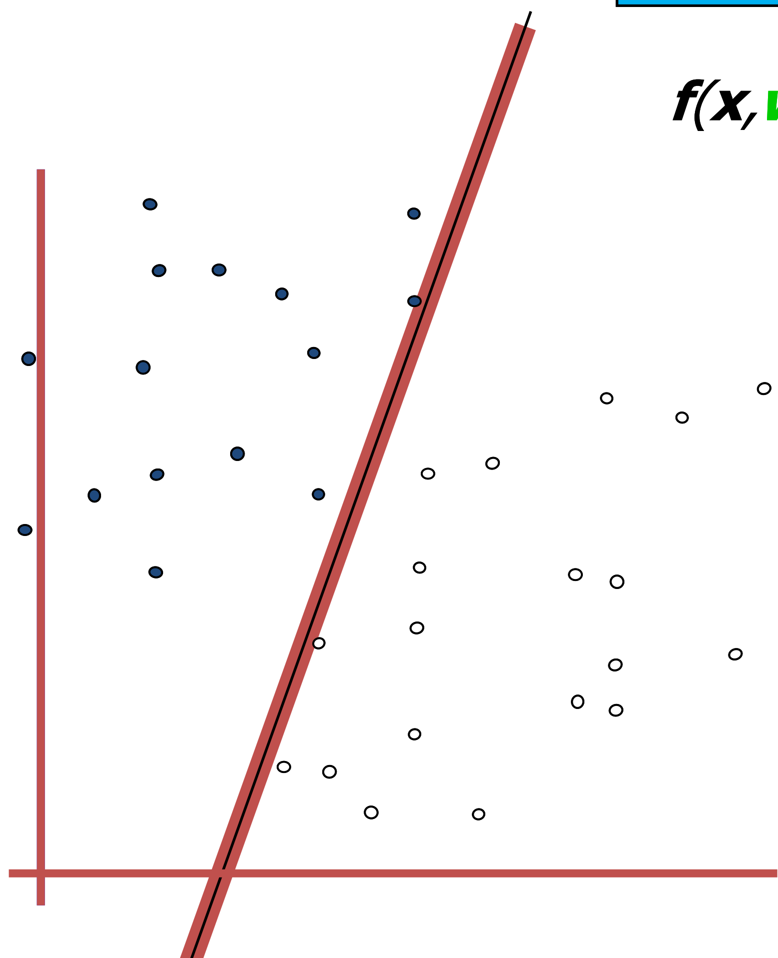
Machine Learning Example: SVM

Classifier Margin



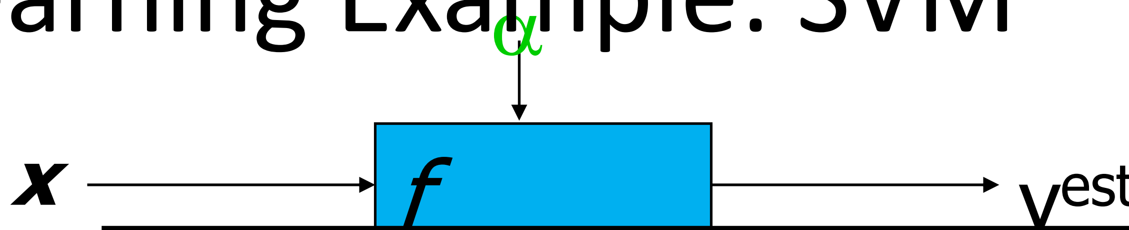
$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\,\boldsymbol{x} + b)$

- denotes +1
- denotes -1



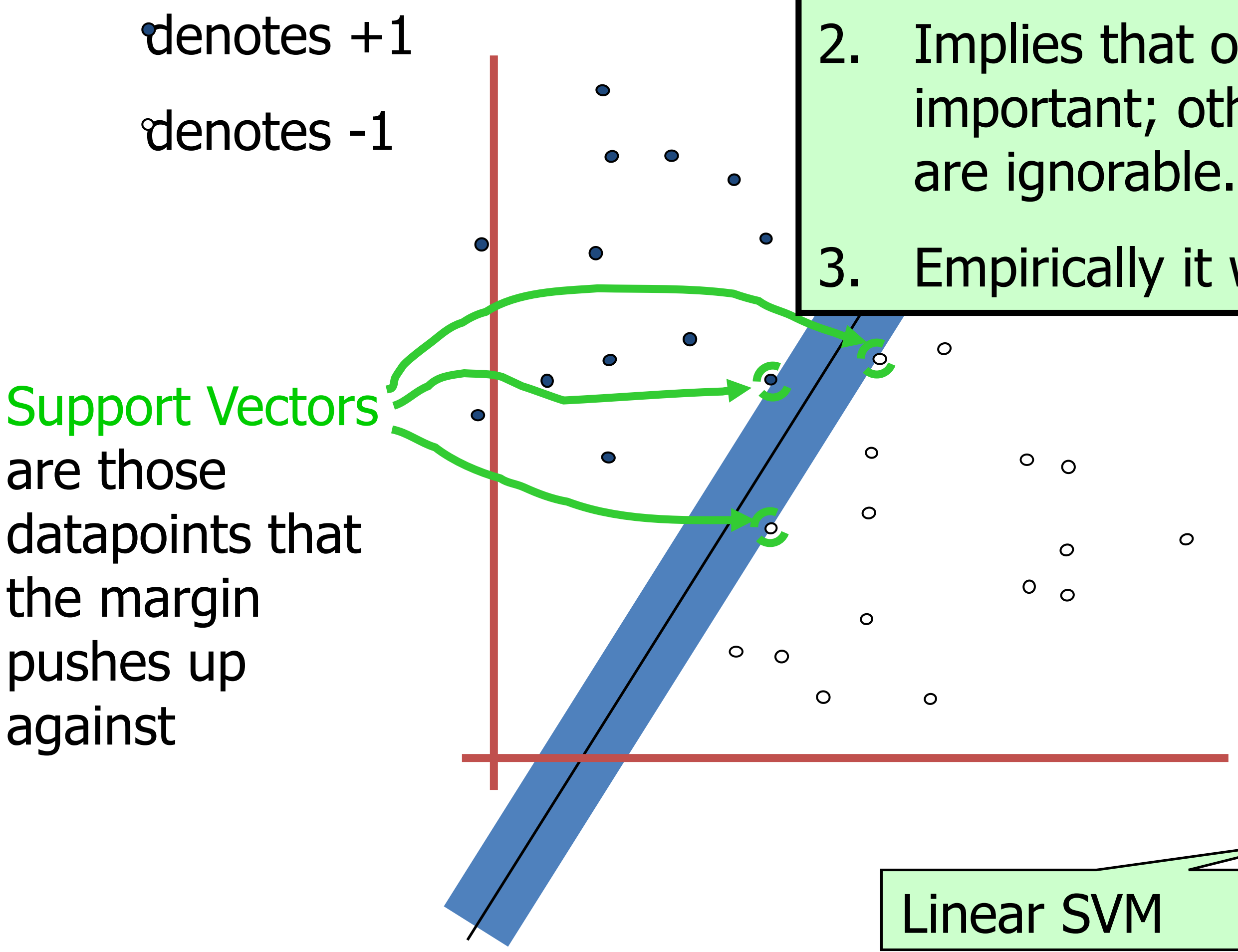
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Machine Learning Example: SVM



1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

Maximum Margin



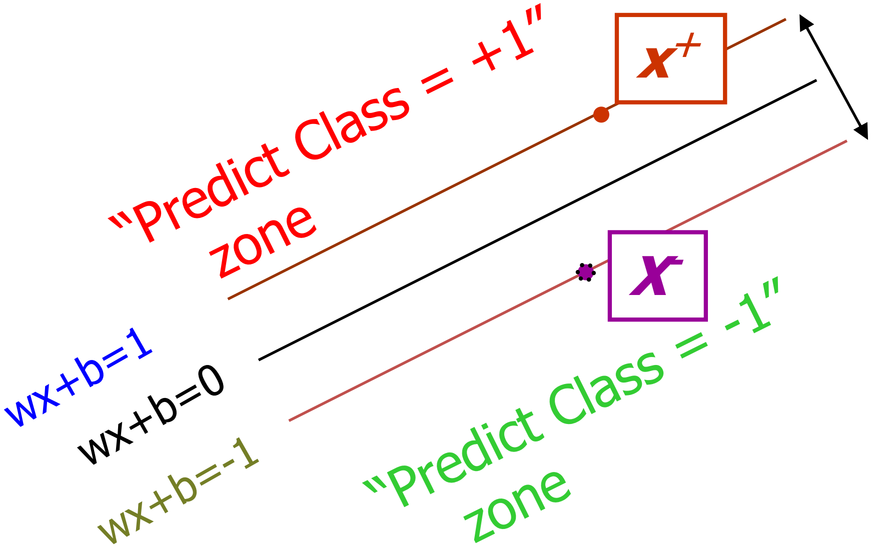
linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Machine Learning Example: SVM

Linear SVM Mathematically



M=Margin Width

What we know:

- ***w** . **x**⁺ + b = +1*
- ***w** . **x**⁻ + b = -1*
- ***w** . (**x**⁺-**x**⁻) = 2*

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Machine Learning Example: SVM

Linear SVM Mathematically

- Goal: 1) Correctly classify all training data

$$wx_i + b \geq 1 \text{ if } y_i = +1$$
$$wx_i + b \leq 1 \text{ if } y_i = -1$$
$$y_i(wx_i + b) \geq 1 \text{ for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2} w^t w$

- We can formulate a Quadratic Optimization Problem and solve for w and b
- Minimize $\Phi(w) = \frac{1}{2} w^t w$

subject to $y_i(wx_i + b) \geq 1 \quad \forall i$

Machine Learning Example: SVM

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\, \mathbf{w}^T\mathbf{w}$ is minimized;
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w^T x_i} + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^{\mathbf{T}}\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

Machine Learning Example: SVM

The Optimization Problem Solution

- The solution has the form:

$$\boxed{\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^\mathbf{T}\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

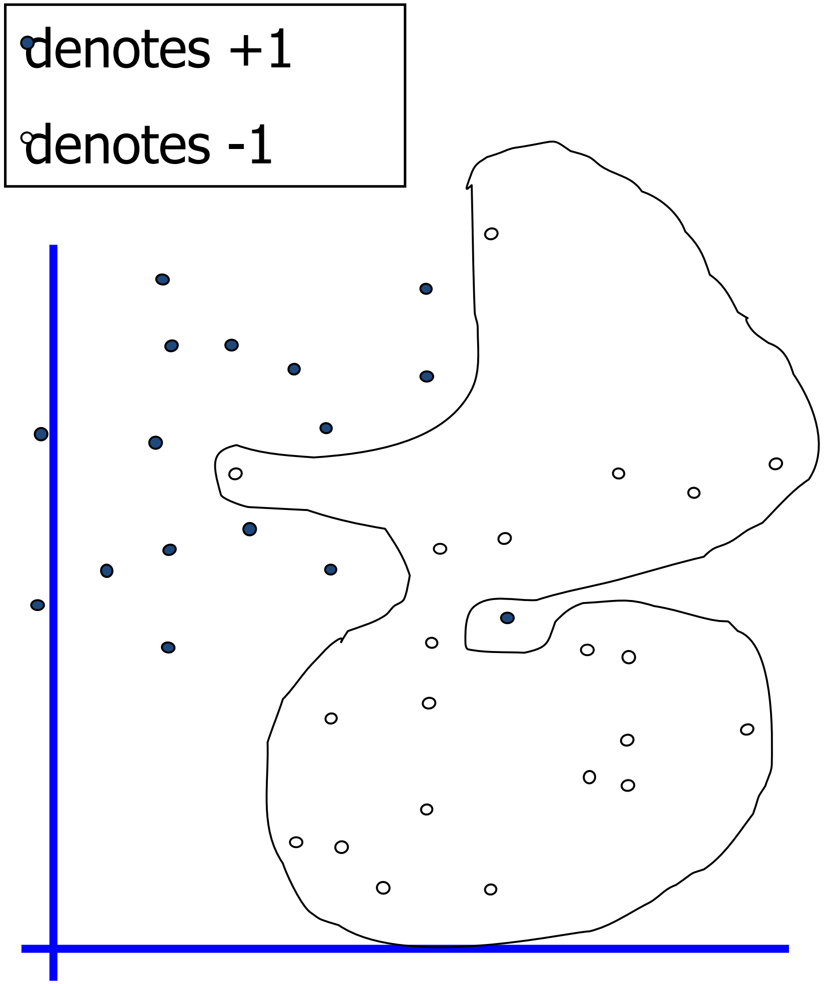
- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i}^\mathbf{T}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x_i}$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^\mathbf{T}\mathbf{x_j}$ between all pairs of training points.

Machine Learning Example: SVM

Dataset with noise



- **Hard Margin:** **So far we require all data points be classified correctly**

  - No training error

- **What if the training set is noisy?**

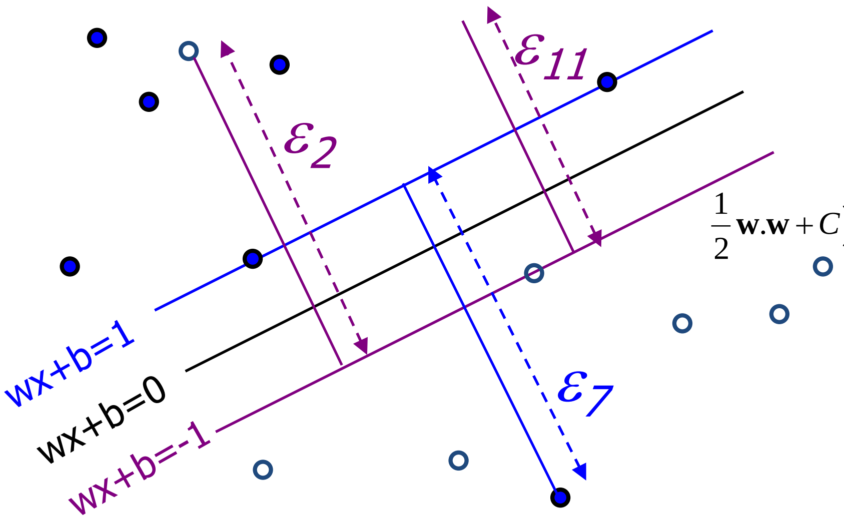
  - Solution 1: **use very powerful kernels**

OVERFITTING!

Machine Learning Example: SVM

Soft Margin Classification

***Slack variables ξi* can be added to allow misclassification of difficult or noisy examples.**



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Machine Learning Example: SVM

Hard Margin v.s. Soft Margin

- The old formulation:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\ (\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + \mathrm{b}) \geq 1$

- The new formulation incorporating slack variables:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\ (\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- Parameter C can be viewed as a way to control overfitting.

Machine Learning Example: SVM

Linear SVMs: Overview

- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

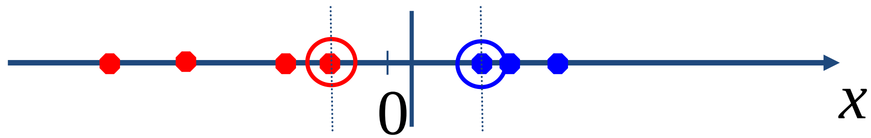
Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{x}_\mathrm{j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

$$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{x} + b$$

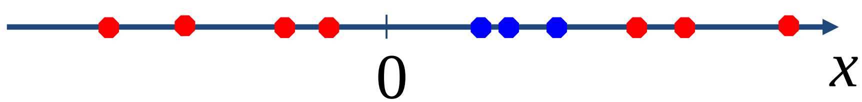
Machine Learning Example: SVM

Non-linear SVM

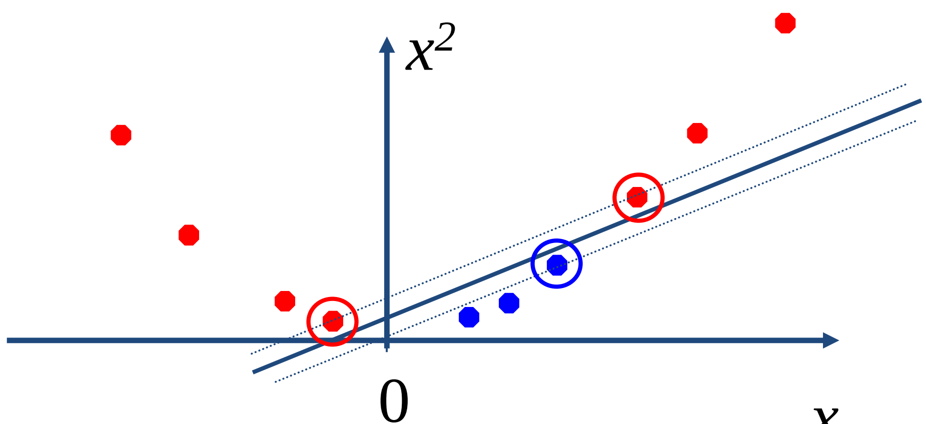
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



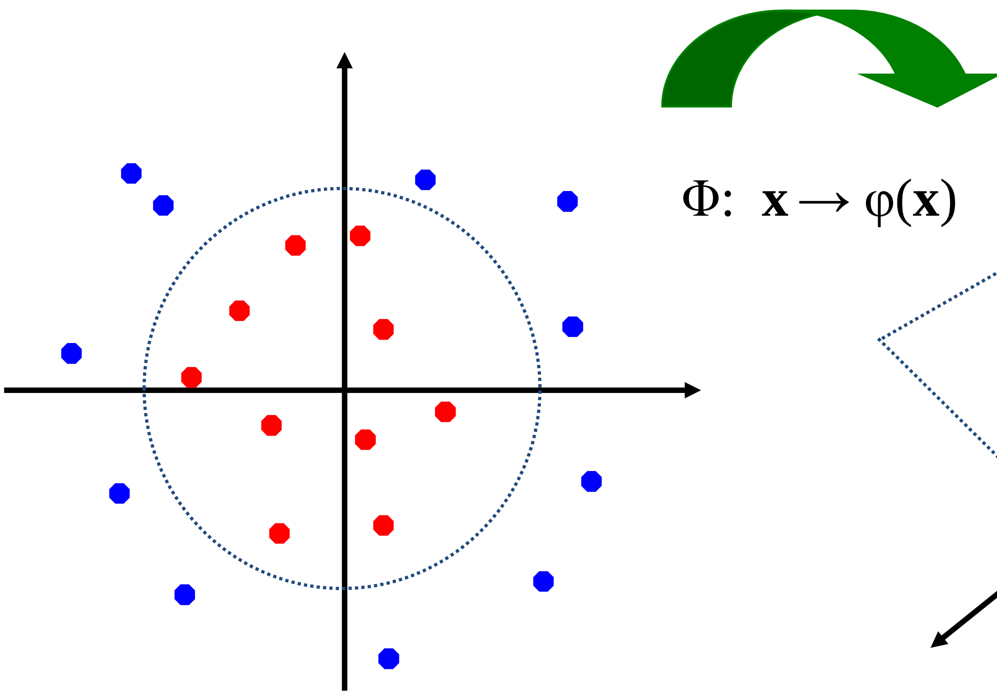
- How about… mapping data to a higher-dimensional space:



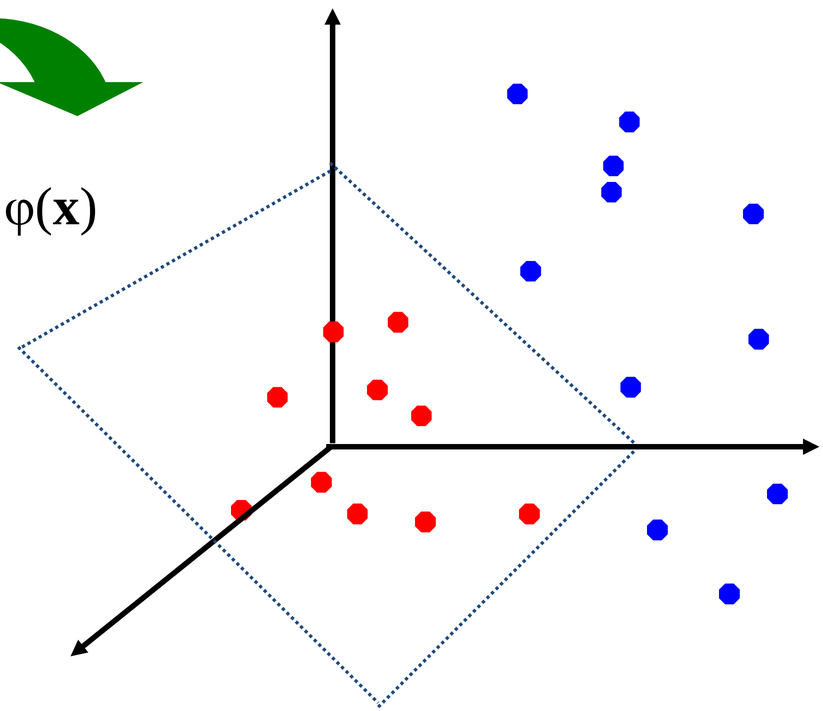
Machine Learning Example: SVM

Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$



Machine Learning Example: SVM

The "Kernel Trick"

- The linear classifier relies on dot product between vectors $K(\mathrm{x_i},\mathrm{x_j})=\mathrm{x_i}^T\mathrm{x_j}$
- If every data point is mapped into high-dimensional space via some transformation Φ: $\mathrm{x} \rightarrow \varphi(\mathrm{x})$, the dot product becomes:
- $K(\mathrm{x_i},\mathrm{x_j})= \varphi(\mathrm{x_i})^T\varphi(\mathrm{x_j})$
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example: 2-dimensional vectors $\mathrm{x}=[x_1\ \ x_2]$; let $K(\mathrm{x_i},\mathrm{x_j})=(1 + \mathrm{x_i}^T\mathrm{x_j})^2$,

  Need to show that $K(\mathrm{x_i},\mathrm{x_j})= \varphi(\mathrm{x_i})^T\varphi(\mathrm{x_j})$:

  $$K(\mathrm{x_i},\mathrm{x_j})=(1 + \mathrm{x_i}^T\mathrm{x_j})^2,$$

  $$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

  $$= [1\ \ x_{i1}^2\ \ \sqrt{2}\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^T [1\ \ x_{j1}^2\ \ \sqrt{2}\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$$

  $$= \varphi(\mathrm{x_i})^T\varphi(\mathrm{x_j}), \quad \text{where } \varphi(\mathrm{x}) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$$

Machine Learning Example: SVM

What Functions are Kernels?

- For some functions $K(x_i,x_j)$ checking that

$$K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j) \text{ can be cumbersome.}$$

- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_N)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_N)$
…	…	…	…	…
$K(\mathbf{x}_N,\mathbf{x}_1)$	$K(\mathbf{x}_N,\mathbf{x}_2)$	$K(\mathbf{x}_N,\mathbf{x}_3)$	…	$K(\mathbf{x}_N,\mathbf{x}_N)$

Machine Learning Example: SVM

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j})= \mathbf{x_i}^{\mathbf{T}}\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i},\mathbf{x_j})= (1+ \mathbf{x_i}^{\mathbf{T}}\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i},\mathbf{x_j}) = \exp\left(-\frac{\left\|\mathbf{x_i}-\mathbf{x_j}\right\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j})= \tanh(\beta_0\mathbf{x_i}^{\mathbf{T}}\mathbf{x_j} + \beta_1)$

Machine Learning Example: SVM

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \Sigma\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- Optimization techniques for finding α_i's remain the same!

Machine Learning Example: SVM

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Machine Learning Example: SVM

Properties of SVM

- Flexibility in choosing a similarity function
- Sparseness of solution when dealing with large data sets

  - only support vectors are used to specify the separating hyperplane
- Ability to handle large feature spaces

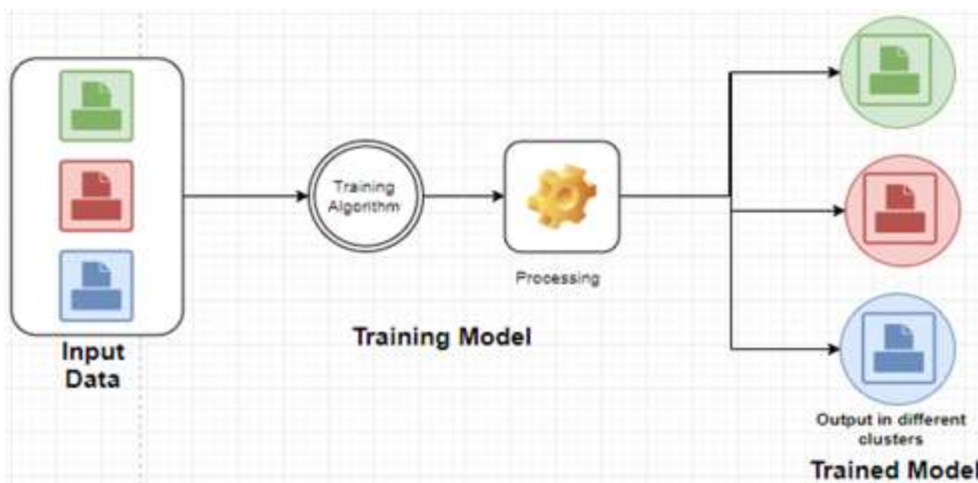
  - complexity does not depend on the dimensionality of the feature space
- Overfitting can be controlled by soft margin approach
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution
- Feature Selection

Machine Learning

- Unsupervised learning

  Given training set $X = (x_i^T)_{i \in [n]}^T$, unlabeled.

  Distributed independently and identically.



Machine Learning

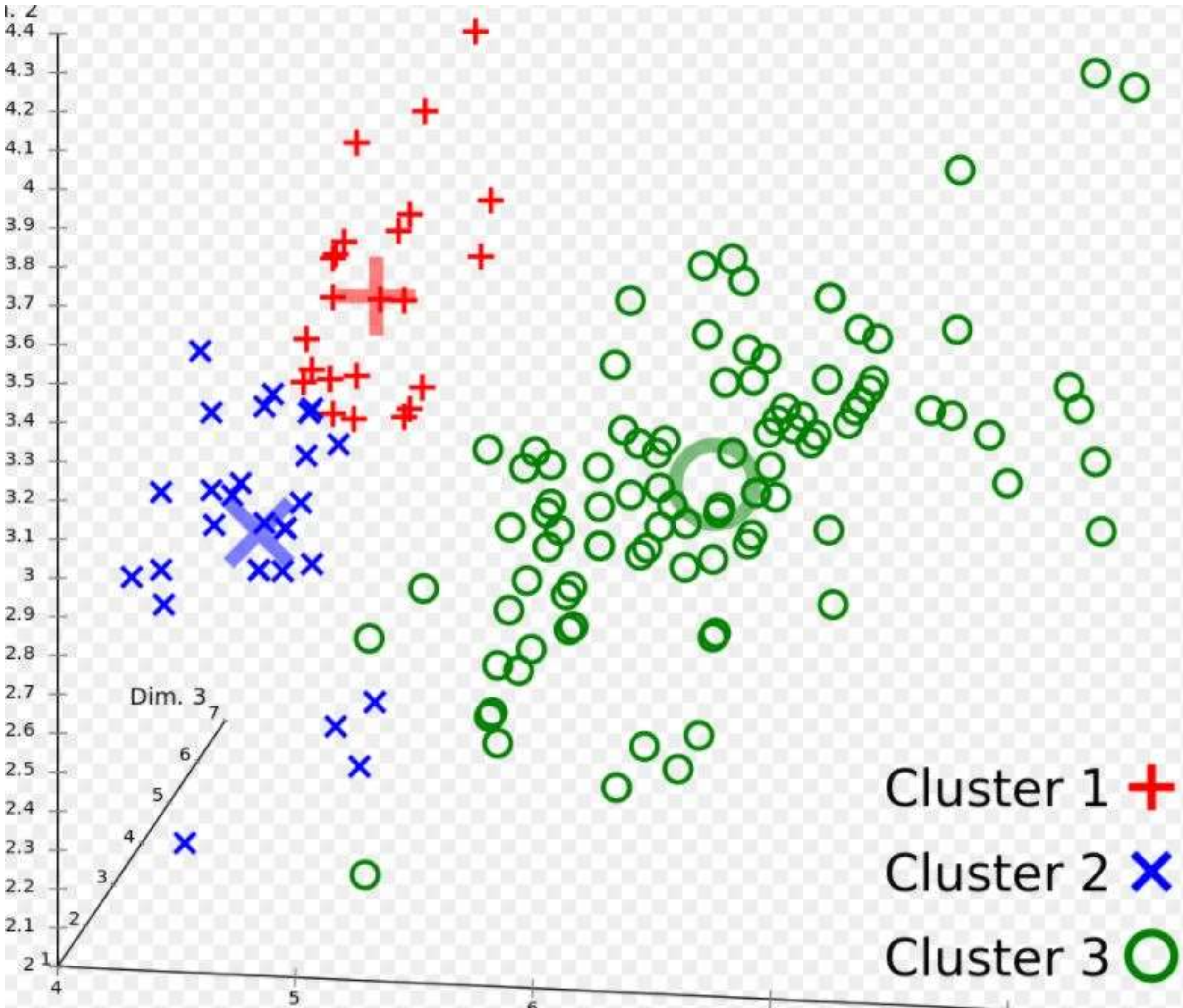
- Unsupervised learning

  Clustering: find „natural“ grouping of instances given unlabeled examples

  Associating: study the relation between data inside database.

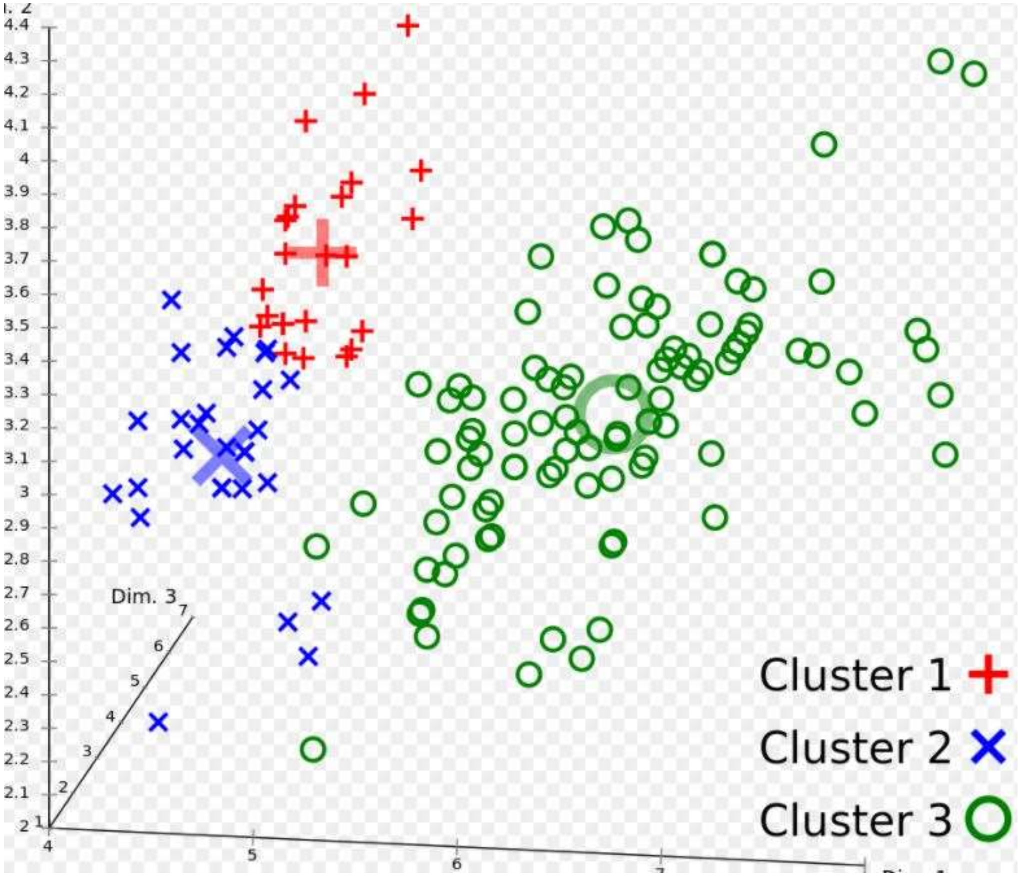
Machine Learning

- Unsupervised learning, example clustering



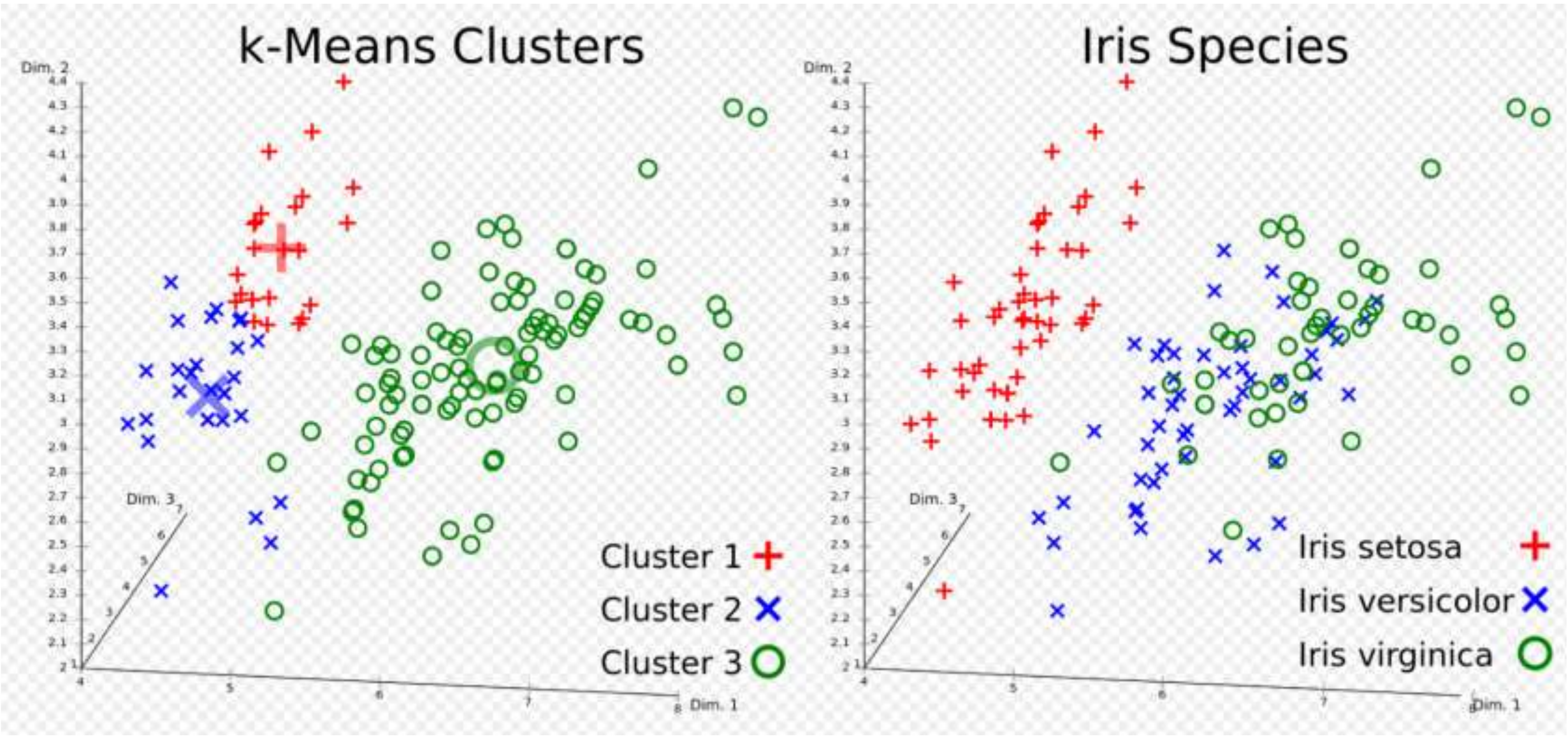
Machine Learning

- Unsupervised learning, example K-means clustering



Machine Learning

- Unsupervised learning, example K-means clustering, drawbacks



Machine Learning

- Semi-supervised learning

Use both labeled and unlabeled examples as training set to train the model for predicting labels for new unlabeled data.

Generally used only when the dataset contains lots of unlabeled data and few labeled data.

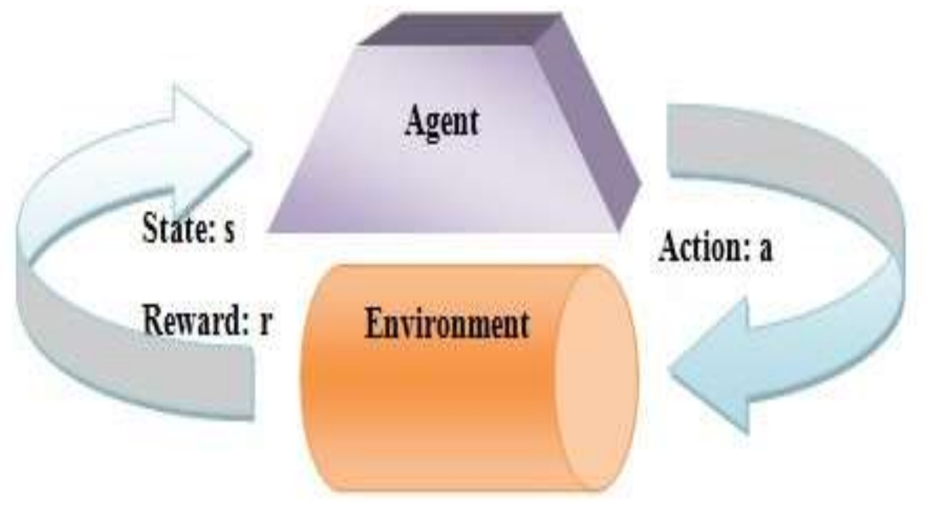
Machine Learning

- Semi-supervised learning
  - Self Training
  - Generative models
  - Co-training
  - Graph Based Algorithms
  - Semi Supervised Support Vector Machines (S3VMs)

Machine Learning

- Reinforcement learning

  - Area of machine learning in which an agent learns by interacting with its environment.
  - Used for solving *Markov Decision Problems*
  - *RL consists of an Agent/Model, set of states, set of actions & reward function.*



Basic Reinforcement Learning System

Machine Learning

- Reinforcement learning

Markov Decision Processes (MDP)

- Problems in which whatever has happened in the past is independent of the future if the current status is known.
- Finite MDP – State & Action spaces are finite.
- Infinite MDP – State or Action spaces are infinite.

Machine Learning

- Reinforcement learning

Elements of MDP

- An Agent or a Model
- A set of States [s ϵ S]
- A set of Actions [a ϵ A]
- Reward function [R (s,a,s')]
- Transition Probability function [P(s'|s,a) or T(s, a, s')]
- Policy [π: S→A]
- Performance metric
- A Start State
- A terminate State (Maybe)
- Performance Metric - Each policy is associated with its own performance metric. Goal is to select the policy having best performance metric.

Machine Learning

- Reinforcement learning

Probability Function and Reward (1)

- ➢ Probability of each possible next state

$$P''_{ss'} = P_r \{s_{(t+1)} = s' | s_t = s, a_t = a\} \quad (2)$$

- ➢ Expected Value of next reward

$$R''_{ss'} = E\{r_{(t+1)} | s_t = s, a_t = a, s_{t+1} = s'\}$$

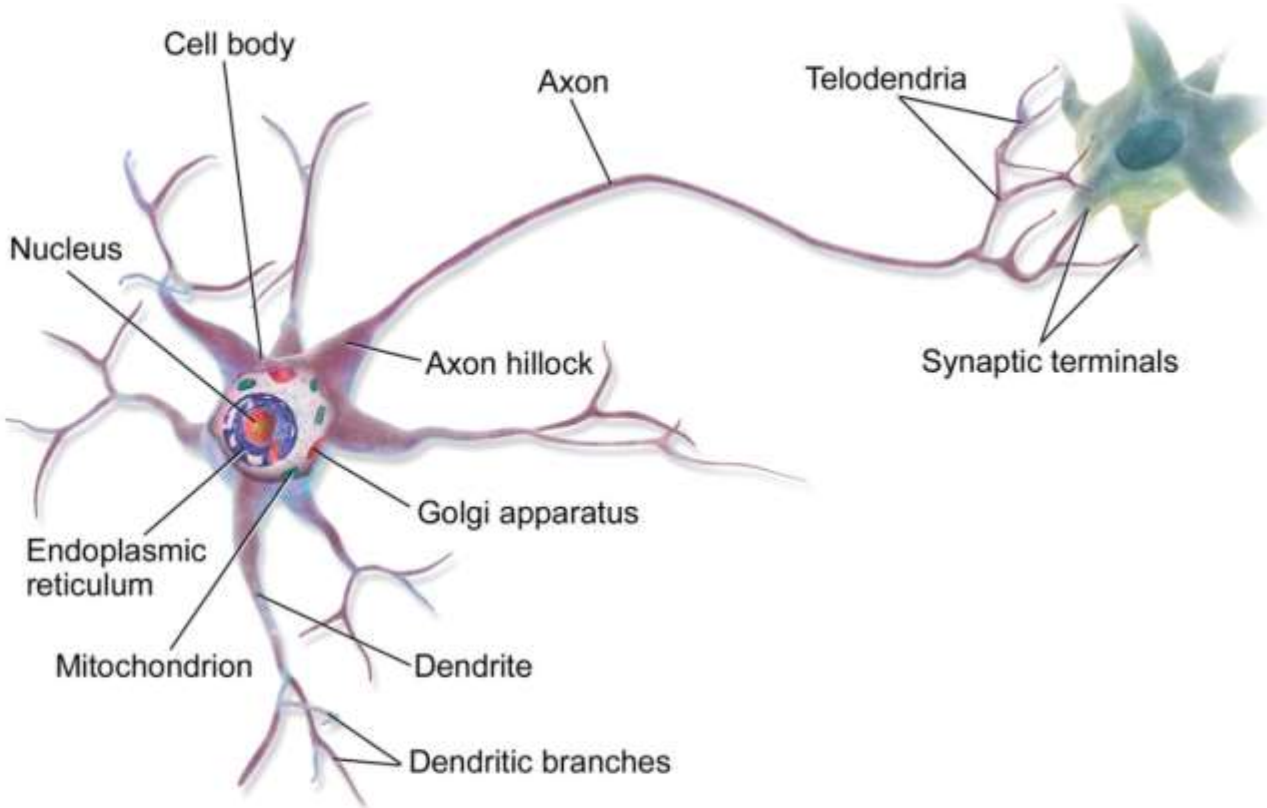
Artificial Neural Networks (ANN)

Objectives

- Modeling biological systems
- Modeling human brain
- Establishment of decision-making systems

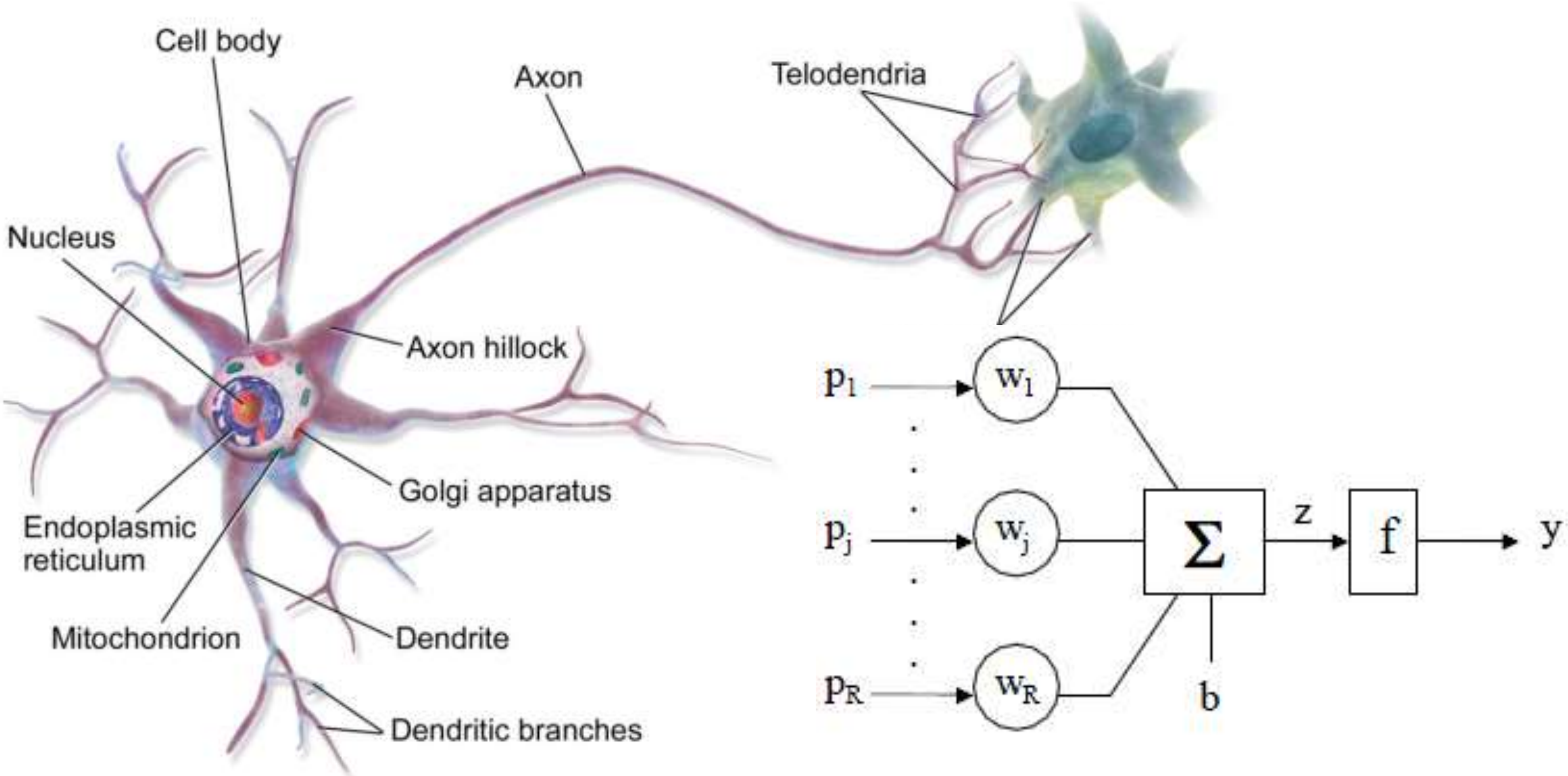
ANN: Biological Foundations

- Structure of a neuron



ANN: Biological Foundations

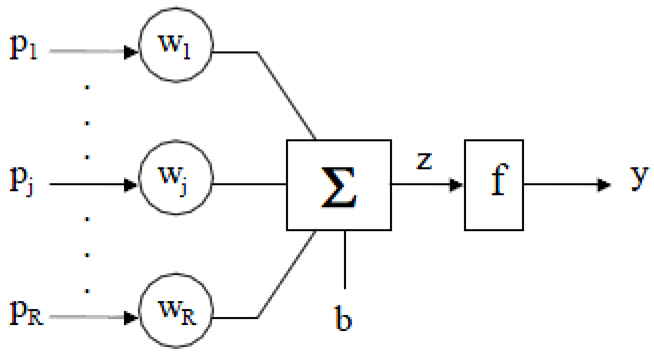
- Structure of a neuron



ANN: Early Neural Models

- McCulloch and Pitts Neuron

McCulloch-Pitts model of an artificial neuron



$$y = f(w_1 \cdot p_1 + \ldots + w_j \cdot p_j + \ldots w_R \cdot p_R + b)$$

$$\boxed{y = f(W \cdot p + b)}$$

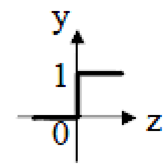
$p = (p_1, \ldots, p_R)^T$ is the input column-vector

$W = (w_1, \ldots, w_R)$ is the weight row-vector

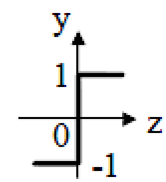
*) The bias b can be treated as a weight whose input is always 1.

<u>Some transfer functions "f"</u>

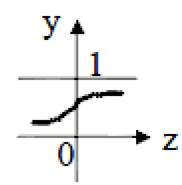
Hard Limit: $y = 0$ if $z<0$
$y = 1$ if $z>=0$



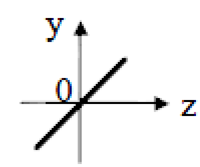
Symmetrical Hard Limit: $y = -1$ if $z<0$
$y = +1$ if $z>=0$



Log-Sigmoid:
$y = 1/(1+e^{-z})$



Linear:
$y = z$

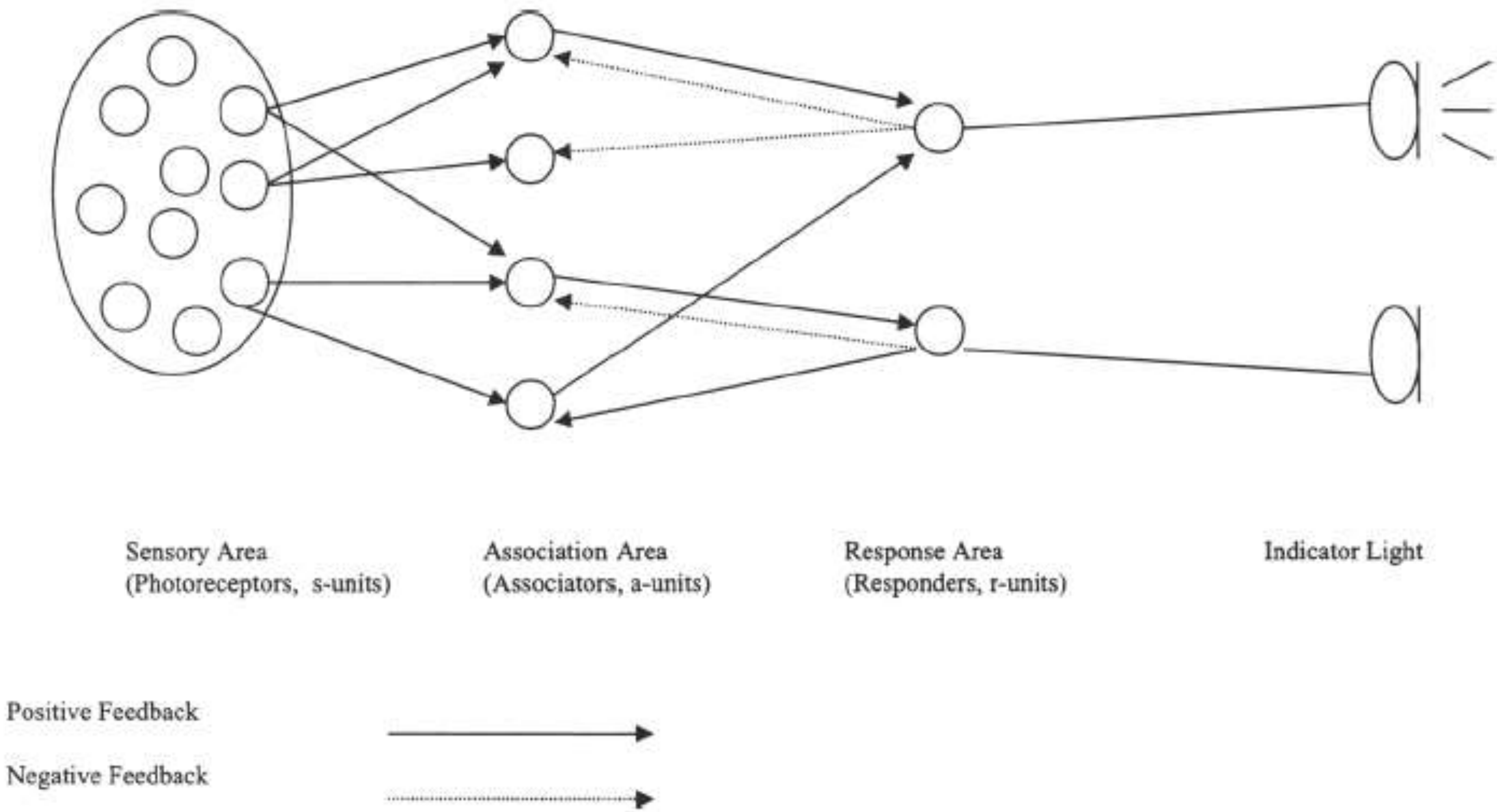


Early Neural Models

- Hebbian Learning: Learning Laws
- Excitatory neuron coupling weights were increased by a subsequent firing.
- Idea: Learning driven by activity
- Weights could only increase

ANN: Early Neural Models

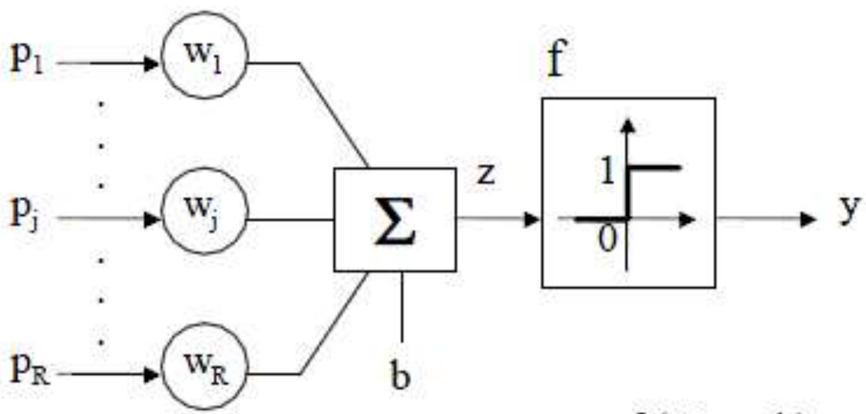
- The Rosenblatt Perceptron



ANN: Early Neural Models

- The Rosenblatt Perceptron

The **perceptron** is a neuron with a hard limit transfer function and a weight adjustment mechanism ("learning") by comparing the actual and the expected output responses for any given input /stimulus.



$$y = f\,(W\cdot p + b)$$

- Perceptrons are well suited for pattern classification/recognition.
- The weight adjustment/training mechanism is called the *perceptron learning rule.*

NB: W is a row-vector and p is a column-vector.

ANN: Early Neural Models

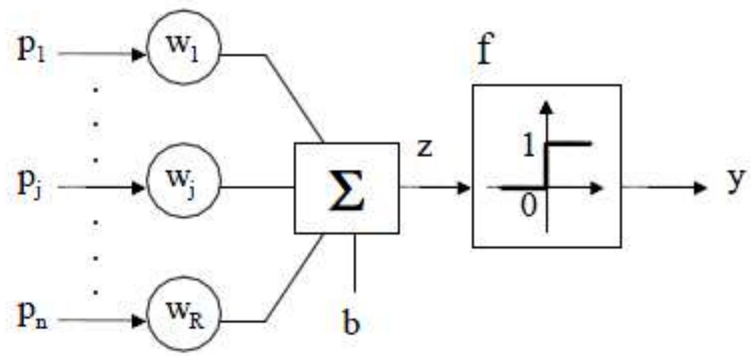
- The Perceptron Learning Rule

- **Supervised learning**

  t <== the target value

  e = t-y <== the error

Because of the perceptron's hard limit transfer function *y, t, e* can take only binary values



$p = (p_1, \ldots, p_R)^T$ is the input column-vector

$W = (x_1, \ldots, x_R)$ is the weight row-vector

Perceptron learning rule:

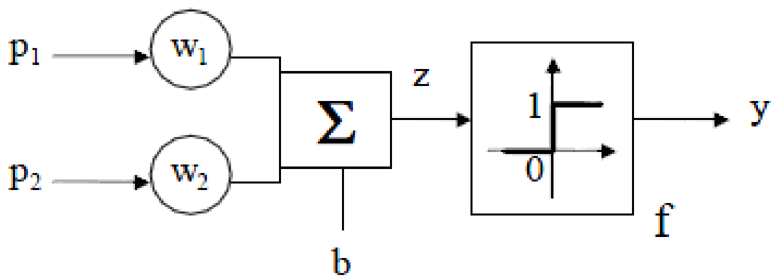
$$\begin{cases} \text{if } e = 1, \text{ then } W^{new} = W^{old} + p,\ b^{new} = b^{old} + 1; \\ \text{if } e = -1, \text{ then } W^{new} = W^{old} - p,\ b^{new} = b^{old} - 1; \\ \text{if } e = 0, \text{ then } W^{new} = W^{old}. \end{cases}$$

$$\boxed{\begin{aligned} W^{new} &= W^{old} + e p^T \\ b^{new} &= b^{old} + e \end{aligned}}$$

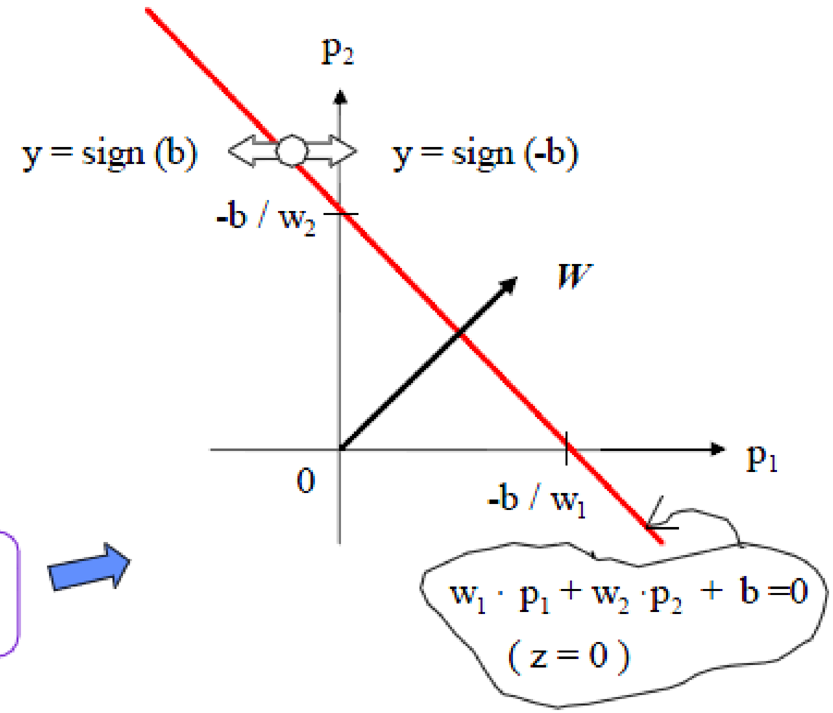
ANN: Early Neural Models

- The hard limit transfer function (threshold function) provides the ability to classify input vectors by deciding whether an input vector belongs to one of two *linearly separable classes.*

Two-Input Perceptron



$$y = hardlim\ (z) = hardlim\{\,[w_1\ , w_2] \cdot [p_1\ , p_2]^T + b\,\}$$



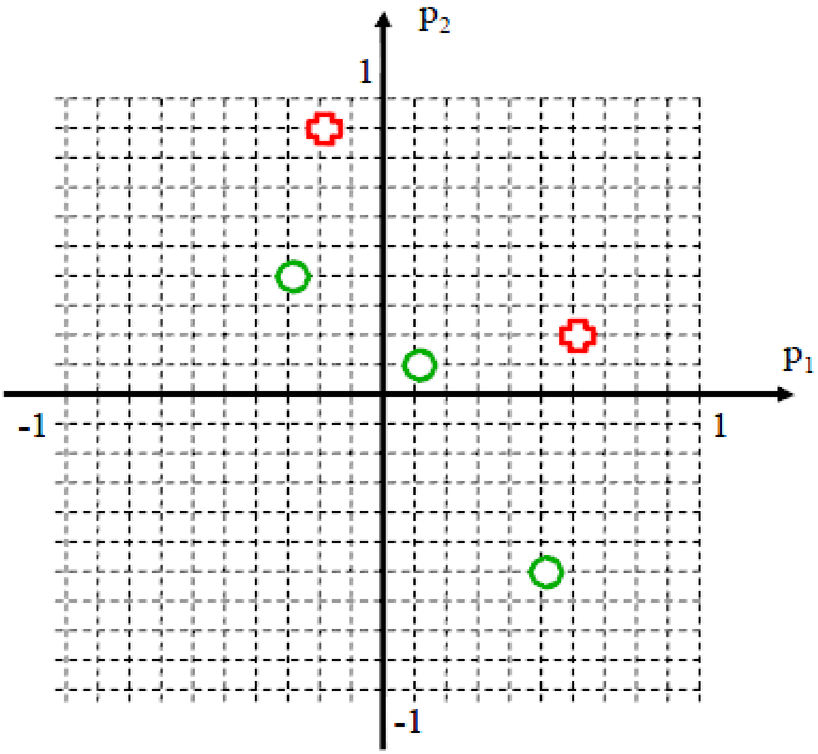
- The two classes (linearly separable regions) in the two-dimensional input space (p_1, p_2) are separated by the line of equation $z = 0$.
- The boundary is always orthogonal to the weight vector *W*.

ANN: Early Neural Models

- Teaching a two-input perceptron to classify five input vectors onto two classes

$$\left\{\begin{array}{l} p(1) = (0.6,\ 0.2)^T \\ t(1) = 1 \end{array}\right\} \quad \left\{\begin{array}{l} p(2) = (-0.2,\ 0.9)^T \\ t(2) = 1 \end{array}\right\} \quad \left\{\begin{array}{l} p(3) = (-0.3,\ 0.4)^T \\ t(3) = 0 \end{array}\right\} \quad \left\{\begin{array}{l} p(4) = (0.1,\ 0.1)^T \\ t(4) = 0 \end{array}\right\} \quad \left\{\begin{array}{l} p(5) = (0.5,\ -0.6)^T \\ t(5) = 0 \end{array}\right\}$$

▶ The MATLAB solution is:




```
P=[0.6 -0.2 -0.3 0.1 0.5;
   0.2  0.9  0.4 0.1 -0.6];
T=[1 1 0 0 0];
W=[-2 2];
b=-1;
plotpv(P,T);
plotpc(W,b);
nepoc=0
Y=hardlim(W*P+b);
while any(Y~=T)
Y=hardlim(W*P+b);
E=T-Y;
[dW,db]= learnp(P,E);
W=W+dW;
b=b+db;
nepoc=nepoc+1;
disp('epochs='),disp(nepoc),
disp(W), disp(b);
plotpv(P,T);
plotpc(W,b);
end
```

- Result

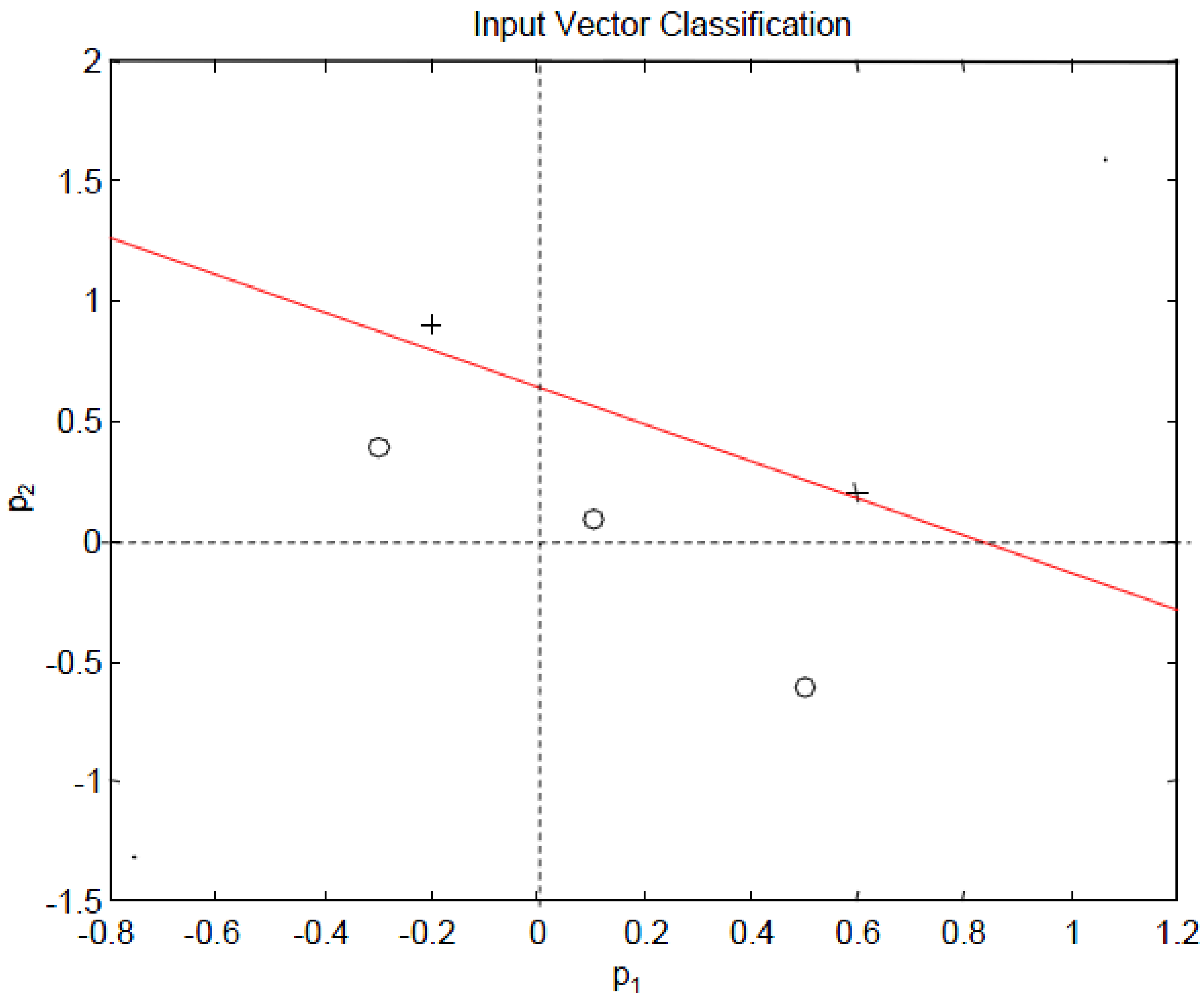
❑ **Example #1:**

After *nepoc* = 11 (epochs of training starting from an initial weight vector `W=[-2 2] and a bias b=-1`) the weights are:

$$\boxed{\begin{array}{l} w_1 = 2.4 \\ w_2 = 3.1 \end{array}}$$

and the bias is:

$$\boxed{b = -2}$$



ANN: Early Neural Models

- The Perceptron Learning Rule

- The larger an input vector p is, the larger is its effect on the weight vector W during the learning process

  Long training times can be caused by the presence of an "outlier," i.e. an input vector whose magnitude is much larger, or smaller, than other input vectors.

  Normalized perceptron learning rule, the effect of each input vector on the weights is of the same magnitude:

$$W^{new} = W^{old} + e p^T / \|p\|$$

$$b^{new} = b^{old} + e$$

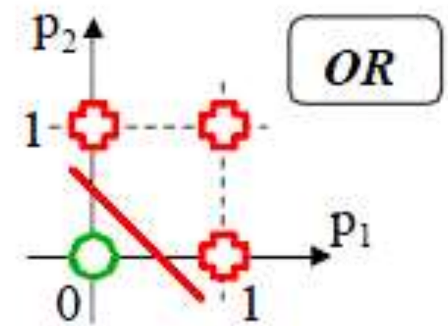
ANN: Early Neural Models

- Perceptron Networks for Linearly Separable Vectors

The hard limit transfer function of the perceptron provides the ability to classify input vectors by deciding whether an input vector belongs to one of two *linearly separable classes.*

$$\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 & 1; \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

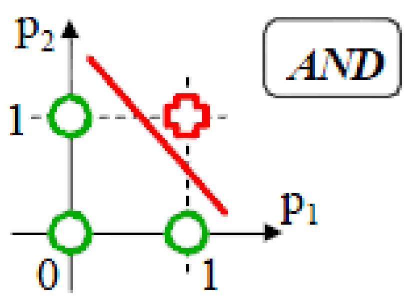
$$\mathbf{t}_{OR} = [\,0 \;\; 1 \;\; 1 \;\; 1\,]$$



$$W = [\,2 \;\; 2\,]$$
$$b = -1$$

$$\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 & 1; \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{t}_{AND} = [\,0 \;\; 0 \;\; 0 \;\; 1\,]$$



$$W = [\,2 \;\; 2\,]$$
$$b = -3$$

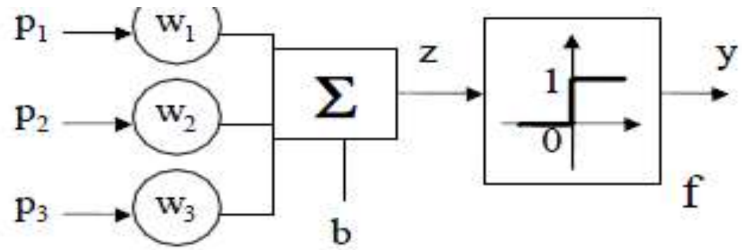
Sensing and Modelling Research Lab
SMRLab - Prof. Emil M. Petriu

sity of Ottawa

ANN: Early Neural Models

- Three-input Perceptron

The two classes in the 3-dimensional input space (p_1, p_2, p_3) are separated by the plane of equation $z = 0$.

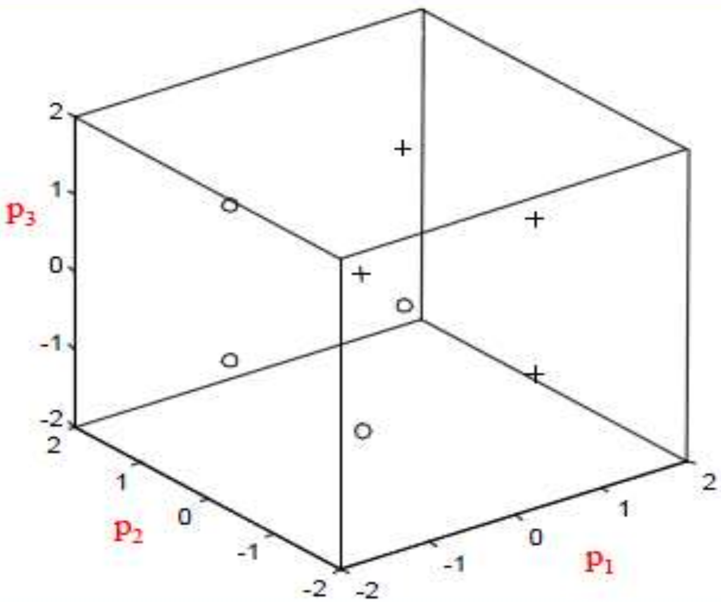


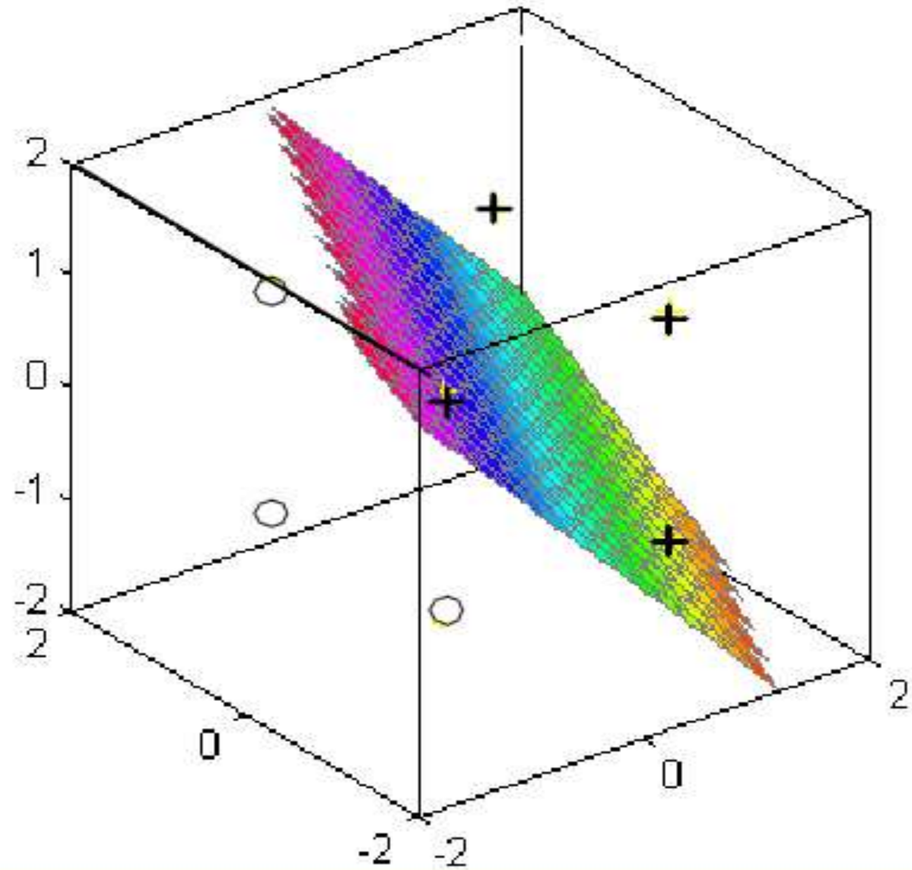
$$y = hardlim\,(z) = hardlim\,\{[w_1, w_2, w_3] \cdot [p_1, p_2\, p_3]^T + b\}$$

EXAMPLE

$$P = [\,-1\ \ 1\ \ 1\ \ -1\ -1\ \ 1\ \ 1\ -1;\ \ -1\ \ -1\ \ 1\ \ 1\ \ -1\ -1\ \ 1\ \ 1;\ \ -1\ \ -1\ -1\ -1\ \ \ 1\ \ 1\ \ 1\ \ 1]$$

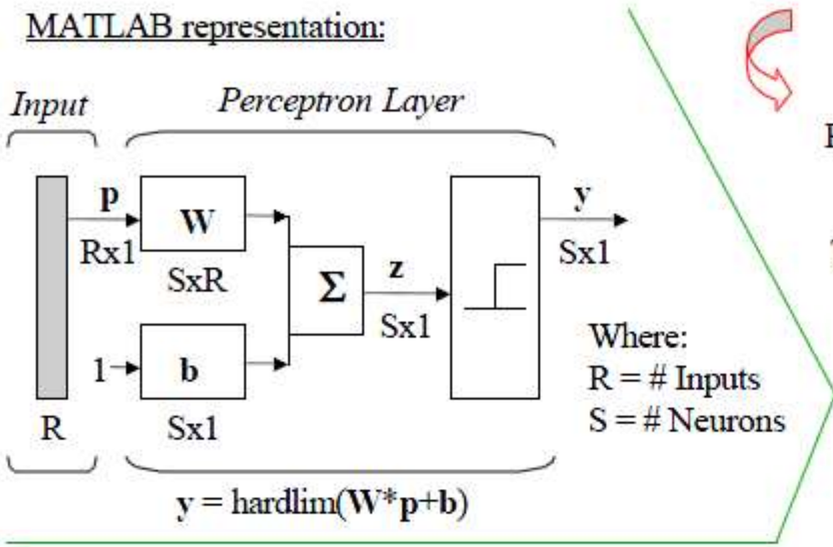
$$T = [\,0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\,]$$





ANN: Early Neural Models

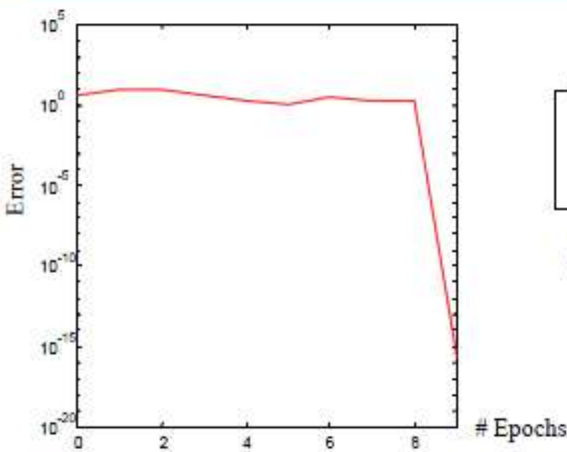
- One-layer multi-perceptron classification of linearly separable patterns



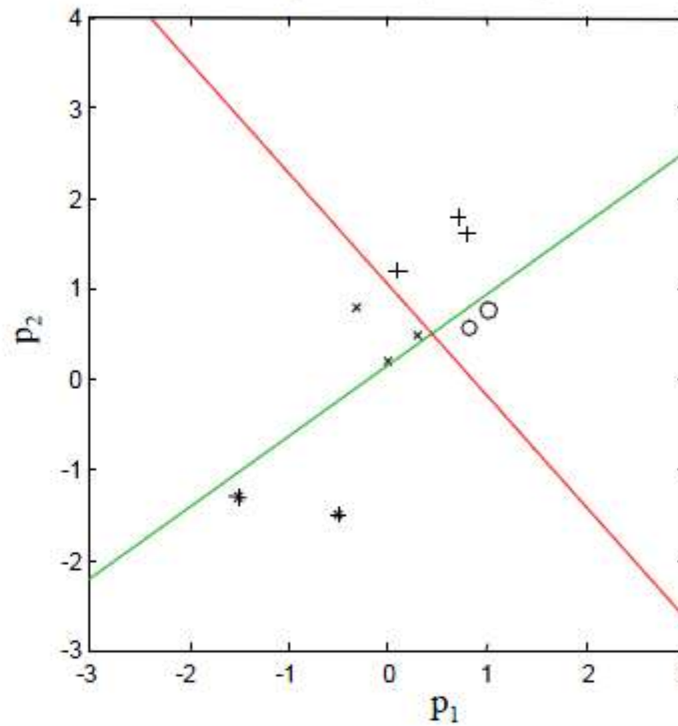
Demo P3 in the "MATLAB Neural Network Toolbox - User's Guide"

$$P = \begin{bmatrix} 0.1 & 0.7 & 0.8 & 0.8 & 1.0 & 0.3 & 0.0 & -0.3 & -0.5 & -1.5; \\ 1.2 & 1.8 & 1.6 & 0.6 & 0.8 & 0.5 & 0.2 & 0.8 & -1.5 & -1.3 \end{bmatrix}$$

$$T = \begin{bmatrix} 1\,1\,1\,0\,0\,1\,1\,1\,0\,0; \\ 0\,0\,0\,0\,0\,1\,1\,1\,1\,1 \end{bmatrix} \Leftrightarrow \left\{ \begin{array}{l} 00 = O \,;\; 10 = + \\ 01 = * \,;\; 11 = x \end{array} \right\}$$

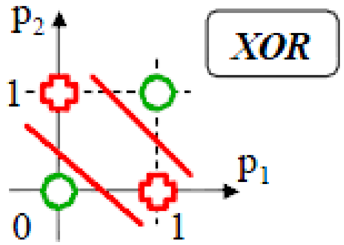


R = 2 inputs
S = 2 neurons



ANN: Early Neural Models

- Perceptron Network for Linearly Non-Separable Vectors

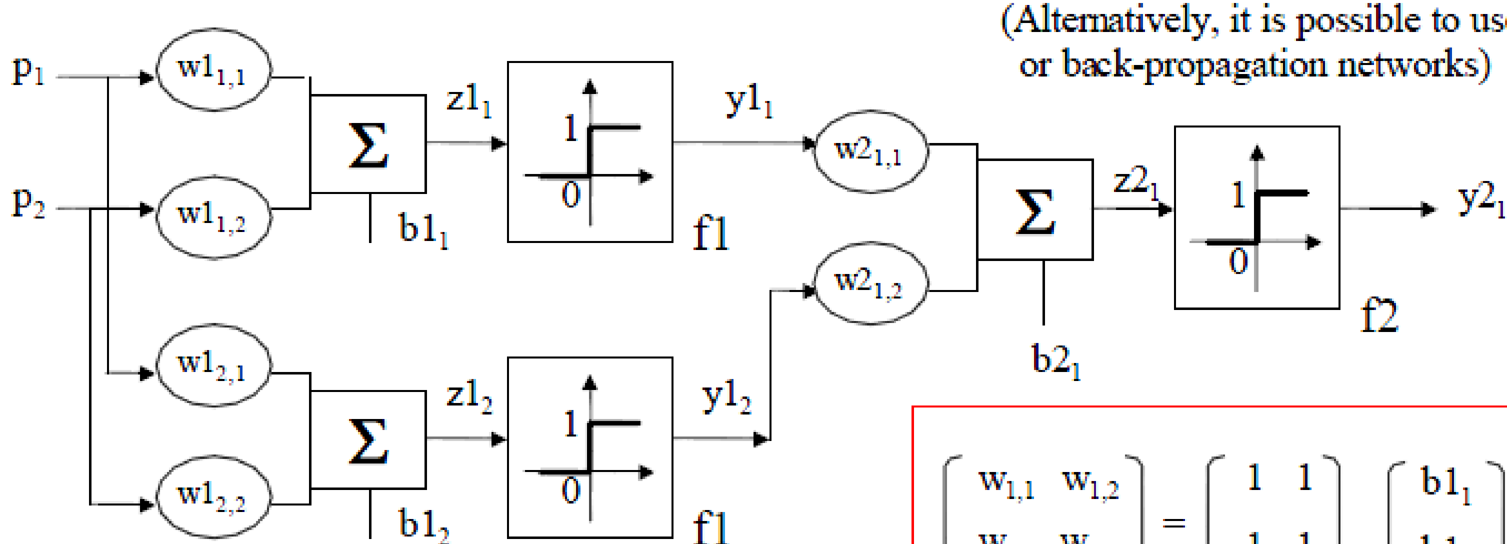


$$p = [\,0\ 0\ 1\ 1\,;\ \ 0\ 1\ 0\ 1\,]$$
$$t_{XOR} = [\,0\ 1\ 1\ 0\,]$$

If a straight line cannot be drawn between the set of input vectors associated with targets of 0 value and the input vectors associated with targets of 1, than a perceptron cannot classify these input vectors.

One solution is to use a two layer architecture, the perceptrons in the first layer are used as preprocessors producing linearly separable vectors for the second layer.

(Alternatively, it is possible to use linear ANN or back-propagation networks)



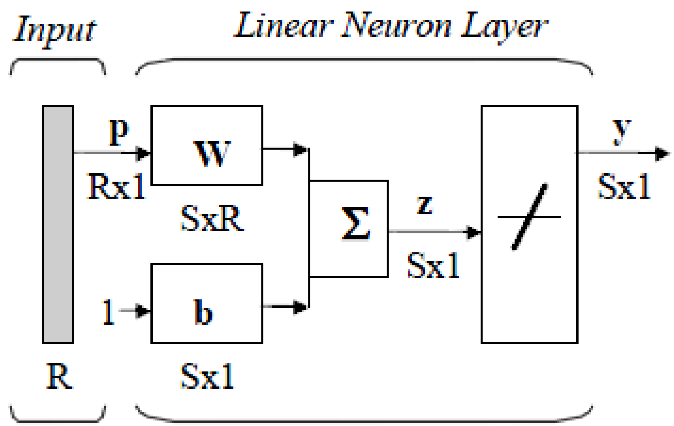
$$\begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} b1_1 \\ b1_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$$

$$[\,w2_{1,1}\ \ w2_{1,2}\,] = [-1\ \ 1] \qquad [\,b2_1\,] = [-0.5]$$

The row index of a weight indicates the destination neuron of the weight and the column index indicates which source is the input for that weight.

ANN: Early Neural Models

- ADALINE Networks



y = purelin(W*p+b)

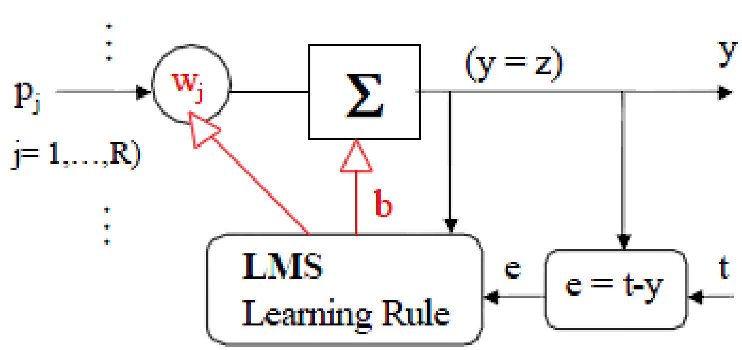
Where: R = # Inputs, S = # Neurons

- Linear neurons have a *linear transfer function* that allows to use a Least Mean-Square (LMS) procedure - *Widrow-Hoff learning rule*- to adjust weights and biases according to the magnitude of errors.

- Linear neurons suffer from the same limitation as the perceptron networks: they can only solve *linearly separable problems.*

(*ADALINE* <== *ADA*ptive *LI*near *NE*uron)

Widrow-Hoff Learning Rule (The • Rule)



The LMS algorithm will adjust ADALINE's weights and biases in such away to *minimize the mean-square-error E [e²] between all sets of the desired response and network's actual response*:

$$E\,[\,(t-y)^2\,] = E\,[\,(t - (w_1 \ldots w_R\;\; b) \cdot (p_1 \ldots p_R\;\; 1)^T)^2\,]$$
$$= E\,[\,(t - W \cdot p)^2\,]$$

(NB: E[…] denotes the "expected value"; **p** is column vector)

ANN: Early Neural Models

- ADALINE: Widrow-Hoff Algorithm

$$E[e^2] = E[(t - \mathbf{W}\cdot\mathbf{p})^2] = \{\text{as for deterministic signals the expectation becomes a time-average}\}$$
$$= E[t^2] - 2\mathbf{W}\cdot\underbrace{E[t\mathbf{p}]}_{} + \mathbf{W}\cdot\underbrace{E[\mathbf{p}\mathbf{p}^T]}_{}\cdot\mathbf{W}^T$$

The cross-correlation between the input vector and its associated target.

The input cross-correlation matrix

If the **input correlation matrix is positive** the LMS algorithm will converge as there will a *unique minimum of the mean square error.*

- The W-H rule is an iterative algorithm uses the "steepest-descent" method to reduce the mean-square-error. The key point of the W-H algorithm is that it replaces $E[e^2]$ estimation by the squared error of the iteration k: $e^2(k)$. At each iteration step k it estimates the gradient of this error ∇_k with respect to $\mathbf{W}$ as a vector consisting of the partial derivatives of $e^2(k)$ with respect to each weight:

$$\nabla_k^* = \frac{\partial e^2(k)}{\partial W(k)} = \left[\frac{\partial e^2(k)}{\partial w_1(k)} \cdots \frac{\partial e^2(k)}{\partial w_R(k)}, \frac{\partial e^2(k)}{\partial b(k)}\right]$$

The weight vector is then modified in the direction that decreases the error:

$$W(k+1) = W(K) - \mu \bullet \nabla_k^* = W(k) - \mu \bullet \frac{\partial e^2(k)}{\partial W(k)} = W(k) - 2\mu \bullet e(k) \bullet \frac{\partial e(k)}{\partial W(k)}$$

- As $t(k)$ and $p(k)$ - both affecting $e(k)$ - are independent of $\mathbf{W}(k)$, we obtain the final expression of the **Widrow-Hoff learning rule**:

$$\boxed{\mathbf{W}(k+1) = \mathbf{W}(k) + 2\mu\cdot e(k)\cdot\mathbf{p}(k)} \Rightarrow \quad b(k+1) = b(k) + 2\mu\cdot e(k)$$

where μ the "learning rate" and $e(k) = t(k)-y(k) = t(k)-\mathbf{W}(k)\cdot\mathbf{p}(k)$

ANN: Early Neural Models

- ADALINE in the MATLAB Toolbox

P = [1.0 -1.2]
T = [0.5   1.0]

One-neuron one-input ADALINE, starting from some random values for w = -0.96 and b= -0.90 and using the "*trainwh*" MATLAB NN toolbox function, reaches the target after 12 epochs with an error e < 0.001. The solution found for the weight and bias is: w = -0.2354 and b= 0.7066.