Lecture 3.1
Supervised learning:
Bayesian and linear classifiers

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

Alberto ORTIZ RODRÍGUEZ

Contents

- Introduction
- Bayesian classification
- Estimation of probability density functions
- Linear discriminant functions and the perceptron algorithm

- **Supervised classification**

  – It is about classifying a new sample in the correct class, having initially designed a classifier from the data available in a training set, in which, in particular, the **samples are labeled** with the class to which they belong.

Contents

- Introduction
- Bayesian classification
- Estimation of probability density functions
- Linear discriminant functions and the perceptron algorithm

Bayesian classification

- The **goal** is to classify a new sample in the **most likely class**
  - Given a **classification task** in M classes, $\omega_1, \omega_2, \dots, \omega_M$, and a **new sample** x, we deal with:

$$p(\omega_i|x), i = 1, 2, ..., M \ (\textbf{probabilities}\ a\ posteriori)$$

  - The classifier decides the most likely class based on the **maximum** of the probabilities *a posteriori*:
    - **Bayesian classification rule**

$$\underline{\text{if}}\ p(\omega_i|x) > p(\omega_j|x), \forall j \neq i,\ \underline{\text{then}}\ x \text{ is labelled as class } i$$

Bayesian classification

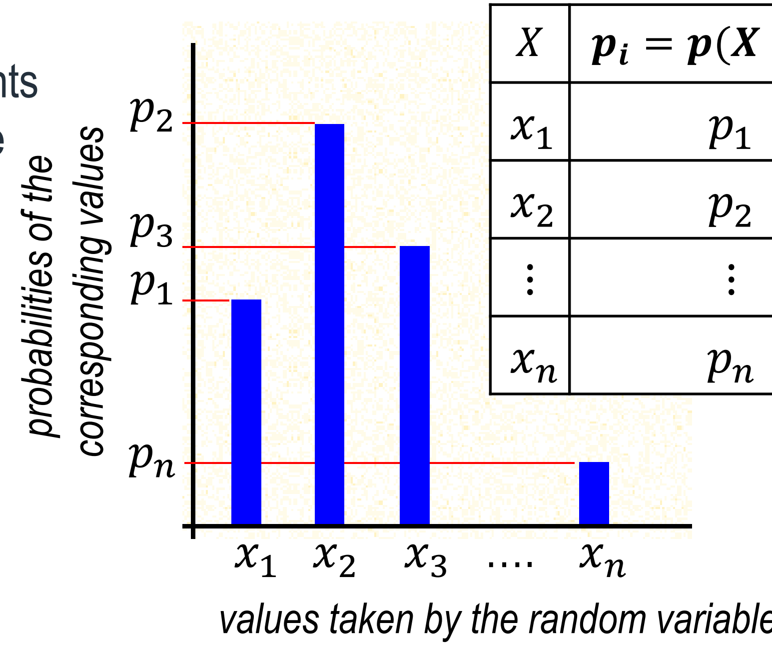
- Review of probability theory:
  - **Probability function**: $p : \Omega \rightarrow [0, 1]$

$$v \rightarrow p(v)$$

    - p assigns a value to each possible event v on the basis of how often that event occurs
    - Ω can be stated as the set of events corresponding to a certain discrete **random variable** X taking certain values, so that

$$p(A_i) \;=\; p(X = x_i)$$

X	$\boldsymbol{p_i = p(X = x_i)}$
x_1	p_1
x_2	p_2
$\vdots$	$\vdots$
x_n	p_n



    - Of particular relevance:

$$p(\Omega) = \sum_{i=1}^{n} p(A_i) = 1$$

- Review of probability theory:
  - **Law of total probability**

    Given M events $A_i, i = 1, ..., M$, such that $\sum_{i=1}^{M} p(A_i) = 1$, for any random event:

    $$p(B) = \sum_{i=1}^{M} p(B|A_i)p(A_i)$$

    where the probability of B conditioned to the occurrence of event A_i is defined as:

    $$p(B|A_i) = \frac{p(B \bigcap A_i)}{p(A_i)}$$

  - **Bayes Rule**

    From the definition of conditional probability, given two events A and B:

    $$p(B|A)p(A) = p(A|B)p(B)$$

❋ *All this is verified under exactly the same conditions by substituting probabilities by probability density functions (**pdf** 's)*

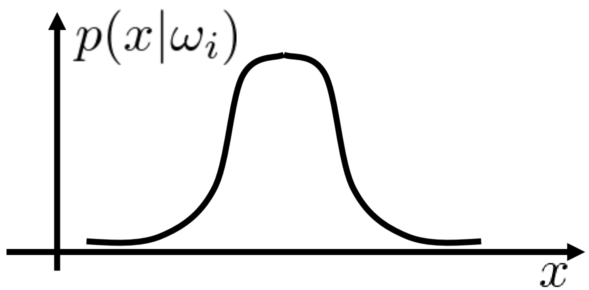
- Bayesian classification: **two-class case** (ω_1, ω_2)
  - Given the **probabilities *a priori*** of both classes $p(\omega_1)$ and $p(\omega_2)$
    - If you do not know $p(\omega_1)$ and $p(\omega_2)$, you can estimate them if necessary:

$$p(\omega_1) \approx \frac{n_1}{n_1+n_2}, \quad p(\omega_2) \approx \frac{n_2}{n_1+n_2}$$

and the pdf (**probability density functions**) of each class

$$p(x|\omega_i), i = 1, 2$$

- If they are unknown, they have to be estimated from the available training data (we will deal with this topic later)



using **Bayes' rule**, it follows that:

$$p(\omega_i|x) = \frac{p(x|\omega_i)p(\omega_i)}{p(x)} = \frac{p(x|\omega_i)p(\omega_i)}{\sum_{j=1}^{2} p(x|\omega_i)p(\omega_i)}$$

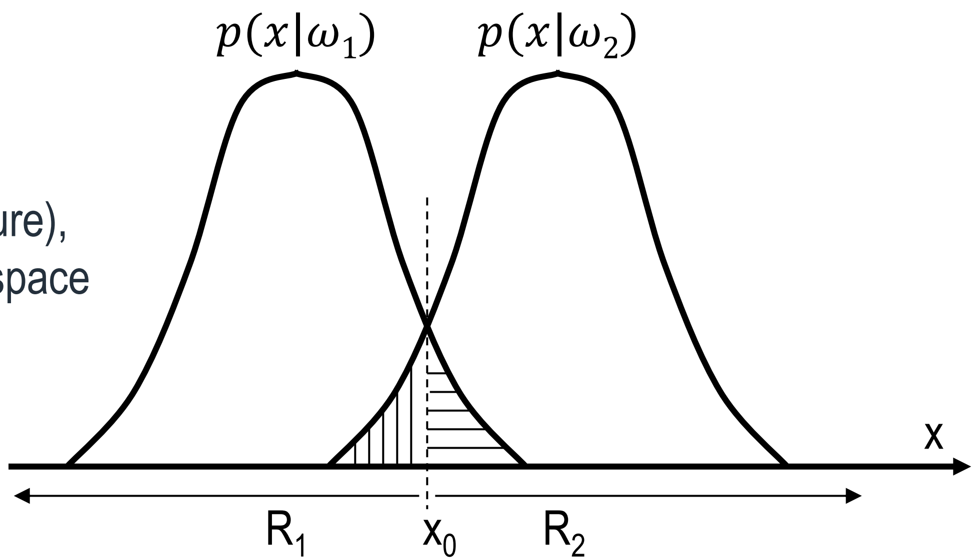
total probability

- Bayesian classification: **two-class case** (ω_1, ω_2)

$$\left.\begin{array}{l} p(\omega_i|x) = \dfrac{p(x|\omega_i)p(\omega_i)}{p(x)} \\ p(\omega_i|x) > p(\omega_j|x), \forall j \neq i \end{array}\right\} \Rightarrow p(x|\omega_i)p(\omega_i) > p(x|\omega_j)p(\omega_j), \forall j \neq i$$

  - If the probabilities *a priori* are equal ($p(\omega_i) = 1/M = 0.5$), then the classification rule becomes dependent only on the **pdf**s of the classes:

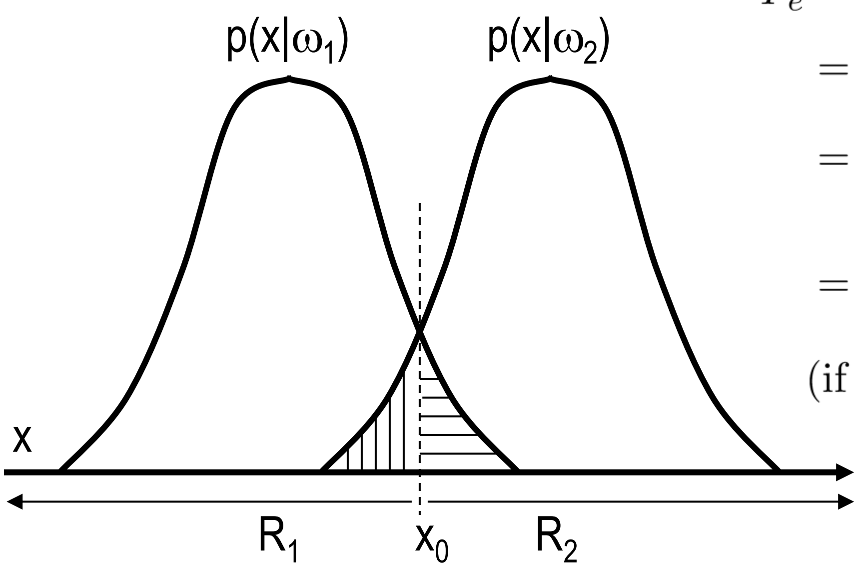
$$p(x|\omega_i) > p(x|\omega_j), \forall j \neq i$$



  - In the one-dimensional case (1 feature), x_0 is a threshold that partitions the space into two regions, R_1 and R_2
  - It is obvious that **classification errors** are unavoidable:
    - Sample x can be inside region R_2 and belong to the class ω_1 (same for R_1 and ω_2)

Bayesian classification

- Bayesian classification: **two-class case** (ω_1, ω_2)
  - In the one-dimensional case, the **probability of making a classification error** is given by:



$$
\begin{aligned}
P_e &= p(x \in R_1 \bigcap x \text{ is from } \omega_2) + p(x \in R_2 \bigcap x \text{ is from } \omega_1) \\
&= p(x \in R_1 | \omega_2) p(\omega_2) + p(x \in R_2 | \omega_1) p(\omega_1) \\
&= p(\omega_2) \int_{R_1} p(x|\omega_2) \mathrm{d}x + p(\omega_1) \int_{R_2} p(x|\omega_1) \mathrm{d}x \\
&= \frac{1}{2} \left(\int_{-\infty}^{x_0} p(x|\omega_2) \mathrm{d}x + \int_{x_0}^{+\infty} p(x|\omega_1) \mathrm{d}x \right)
\end{aligned}
$$

$(\text{if } p(\omega_1) = p(\omega_2) = 0.5)$

THEOREM

The Bayesian classifier minimizes the likelihood of classification error

That is to say, if we move x_0 left or right we will increase P_e

Bayesian classification

- Bayesian classification: **two-class case** (ω_1, ω_2)

Proof (optimality of the Bayesian classifier)

On the one hand:

$$P_e = p(x \in R_2 \bigcap \omega_1) + p(x \in R_1 \bigcap \omega_2) = p(x \in R_2|\omega_1)p(\omega_1) + p(x \in R_1|\omega_2)p(\omega_2)$$

$$= p(\omega_1) \int_{R_2} p(x|\omega_1)\mathrm{d}x + p(\omega_2) \int_{R_1} p(x|\omega_2)\mathrm{d}x$$

$$= \boxed{\int_{R_2} p(\omega_1|x)p(x)\mathrm{d}x} + \int_{R_1} p(\omega_2|x)p(x)\mathrm{d}x$$

On the other hand:

$$\int_{\Omega} p(x|\omega_1)\mathrm{d}x = 1 \Rightarrow \int_{\Omega} \frac{p(\omega_1|x)p(x)}{p(\omega_1)}\mathrm{d}x = 1$$

$$\Rightarrow \int_{R_1} p(\omega_1|x)p(x)\mathrm{d}x + \boxed{\int_{R_2} p(\omega_1|x)p(x)\mathrm{d}x} = p(\omega_1)$$

Therefore:

$$P_e = p(\omega_1) - \int_{R_1} (p(\omega_1|x) - p(\omega_2|x))p(x)\mathrm{d}x$$

$\Rightarrow P_e$ is minimum if R_1 is defined such that, inside R_1, $p(\omega_1|x) > p(\omega_2|x)$

Bayesian classification

- Bayesian classification:

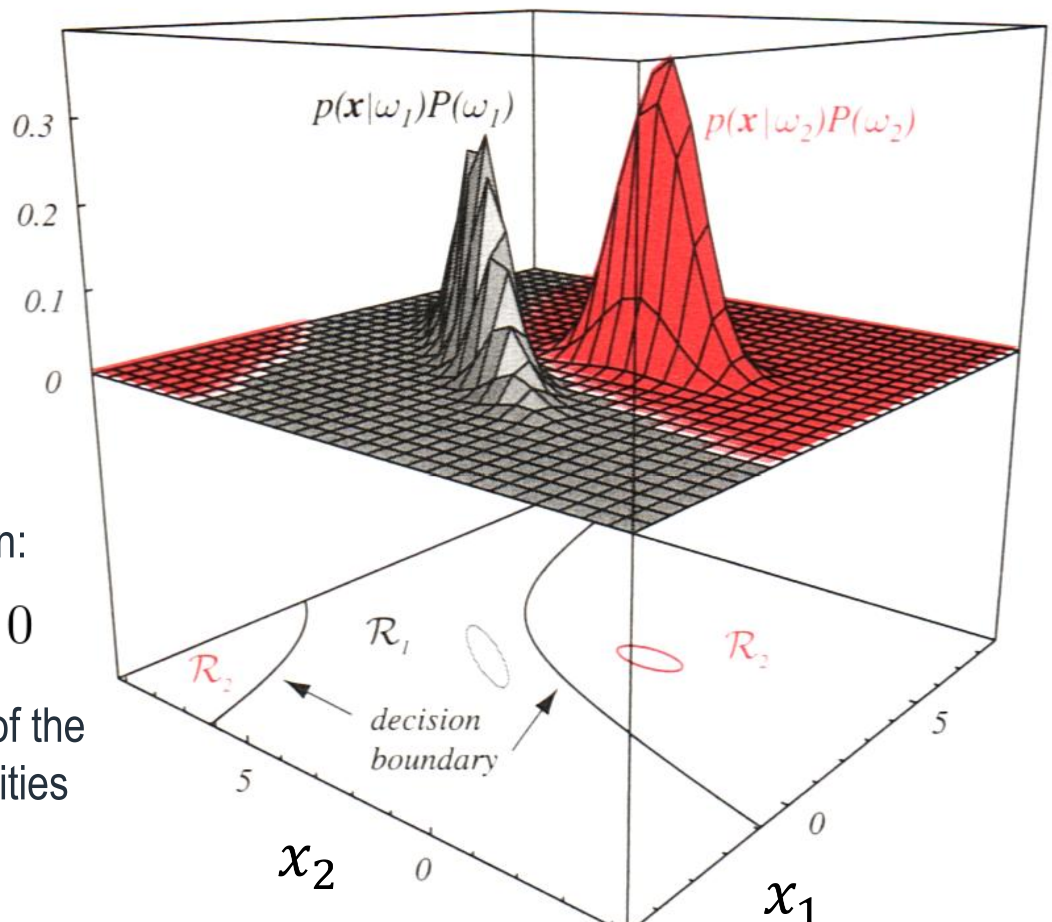
  two-class case (ω_1, ω_2) and **2 features** x_1 and x_2

  – Minimize the probability of error is equivalent to **partition the space of features into M regions** (as many as classes)

  – In general terms, regions R_i and R_j are separated by a **decision curve** that is described by the equation:

$$p(\omega_i|x) - p(\omega_j|x) = 0$$

  - Corresponds to the points of the space in which the probabilities *a posteriori* coincide



- **Bayesian Classification**
  - **Example** Let us consider a problem of 2 equiprobable classes ($p(\omega_1) = p(\omega_2) = 0.5$) such that the class pdfs are Gaussians of **variance 0.5** and **means 0 and 1** respectively:

$$p(x|\omega_1) = \frac{1}{\sqrt{\pi}} e^{-x^2}, \quad p(x|\omega_2) = \frac{1}{\sqrt{\pi}} e^{-(x-1)^2} \quad \left(N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right)$$

Calculate the optimal threshold x_0 for minimum error probability.

$$x_0 : p(\omega_1|x_0) = p(\omega_2|x_0)$$

$$x_0 : p(x_0|\omega_1)p(\omega_1) = p(x_0|\omega_2)p(\omega_2)$$

$$x_0 : e^{-x_0^2} = e^{-(x_0-1)^2} \Rightarrow x_0^2 = (x_0 - 1)^2 = x_0^2 - 2x_0 + 1 \Rightarrow x_0 = 0.5$$

- **Bayesian classification for normal distributions**
  - In the **one-dimensional case**:

$$p(x|\omega) = \frac{1}{\sqrt{2\pi}\sigma_\omega} e^{-\frac{(x-\mu_\omega)^2}{2\sigma_\omega^2}}$$

  - We assume that the pdf of the classes obey the **Gaussian L-dimensional distribution**:

$$p(x|\omega) = \frac{1}{\sqrt{(2\pi)^L |\Sigma_\omega|}} e^{-\frac{1}{2}(x-\mu_\omega)^T \Sigma_\omega^{-1} (x-\mu_\omega)}$$

    - For class ω:

$$\mu_\omega = (\mu_1, ..., \mu_s, ..., \mu_L) \quad , \quad \mu_s = \frac{1}{N} \sum_{i=1}^N x_{si}$$

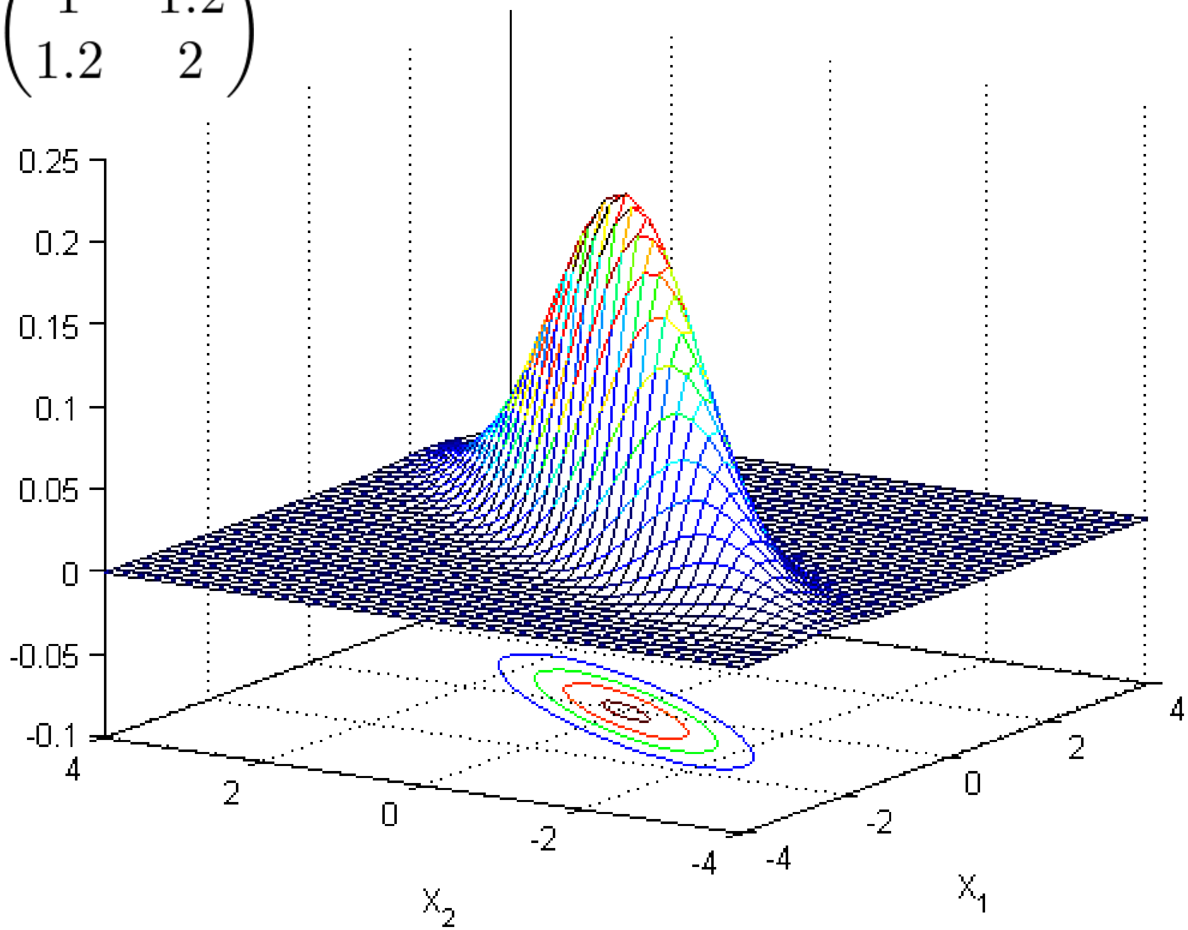
$$\Sigma_\omega = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1L} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2L} \\ \vdots & \vdots & & \vdots \\ \sigma_{L1} & \sigma_{L2} & \cdots & \sigma_L^2 \end{bmatrix} \quad , \quad \sigma_{st} = \frac{1}{N-1} \sum_{i=1}^N (x_{si} - \mu_s)(x_{ti} - \mu_t)$$

$$= \frac{1}{N-1} \sum_{i=1}^N (x - \mu_\omega)(x - \mu_\omega)^T$$

  - This distribution models properly many cases and is treatable mathematically and computationally, hence its popularity

- **Bayesian classification for normal distributions**
  - **Example**

$$\mu = (0,0) \quad \Sigma = \begin{pmatrix} 1 & 1.2 \\ 1.2 & 2 \end{pmatrix}$$



- **Bayesian classifier for normal distributions**
  - The goal is to derive the **Bayesian classifier** for the case

$$p(x|\omega_i) = \frac{1}{\sqrt{(2\pi)^L|\Sigma_i|}} e^{-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)}, i = 1, \ldots, M$$

- Due to the exponential form of the pdf, it is preferable to work with the following **discrimination functions** $g_i(x)$, which involve the **monotonous function** $\ln(\cdot)$:

$$g_i(x) = \ln(p(x|\omega_i)p(\omega_i)) = \ln p(x|\omega_i) + \ln p(\omega_i)$$
$$g_i(x) = c_i - \frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) + \ln p(\omega_i)$$
$$c_i = -\frac{L}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i|$$

- Finally:

$$g_i(x) = -\frac{1}{2}x^T\Sigma_i^{-1}x + \frac{1}{2}x^T\Sigma_i^{-1}\mu_i + \frac{1}{2}\mu_i^T\Sigma_i^{-1}x - \frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i + \ln p(\omega_i) + c_i$$

- **Bayesian classifier for normal distributions**
  - Case of **2 uncorrelated characteristics**

$$L = 2, \quad x = (x_1, x_2)^T, \quad \Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & 0 \\ 0 & \sigma_{i2}^2 \end{pmatrix}$$

$$g_i(x) = -\frac{1}{2}x^T\Sigma_i^{-1}x + \frac{1}{2}x^T\Sigma_i^{-1}\mu_i + \frac{1}{2}\mu_i^T\Sigma_i^{-1}x - \frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i + \ln p(\omega_i) + c_i$$

$$\Rightarrow g_i(x) = -\frac{1}{2}\left(\frac{x_1^2}{\sigma_{i1}^2} + \frac{x_2^2}{\sigma_{i2}^2}\right) + \left(\frac{\mu_{i1}x_1}{\sigma_{i1}^2} + \frac{\mu_{i2}x_2}{\sigma_{i2}^2}\right) - \frac{1}{2}\left(\frac{\mu_{i1}^2}{\sigma_{i1}^2} + \frac{\mu_{i2}^2}{\sigma_{i2}^2}\right) + \ln p(\omega_i) + c_i$$

$$= \alpha_i\, x_1^2 + \beta_i\, x_2^2 + \gamma_i\, x_1 + \delta_i\, x_2 + \epsilon_i$$

- The **decision rule** is now given by the equation $g_i(x) - g_j(x) = 0$
  - L = 2: ellipse, parabola, hyperbole, etc. – **conic**, rule = 2D curve
  - L = 3: ellipsoid, paraboloid, hyperboloid, etc. – **quadric**, rule = 3D surface
  - L > 3: **hiperquadric**

QUADRATIC CLASSIFIER

$$g_i(x) > g_j(x) \Rightarrow p(\omega_i|x) > p(\omega_j|x) \Rightarrow x \to \omega_i$$

$$g_i(x) - g_j(x) = (\alpha_i - \alpha_j)x_1^2 + (\beta_i - \beta_j)x_2^2 + (\gamma_i - \gamma_j)x_1 + (\delta_i - \delta_j)x_2 + (\epsilon_i - \epsilon_j) > 0 \Rightarrow x \to \omega_i$$

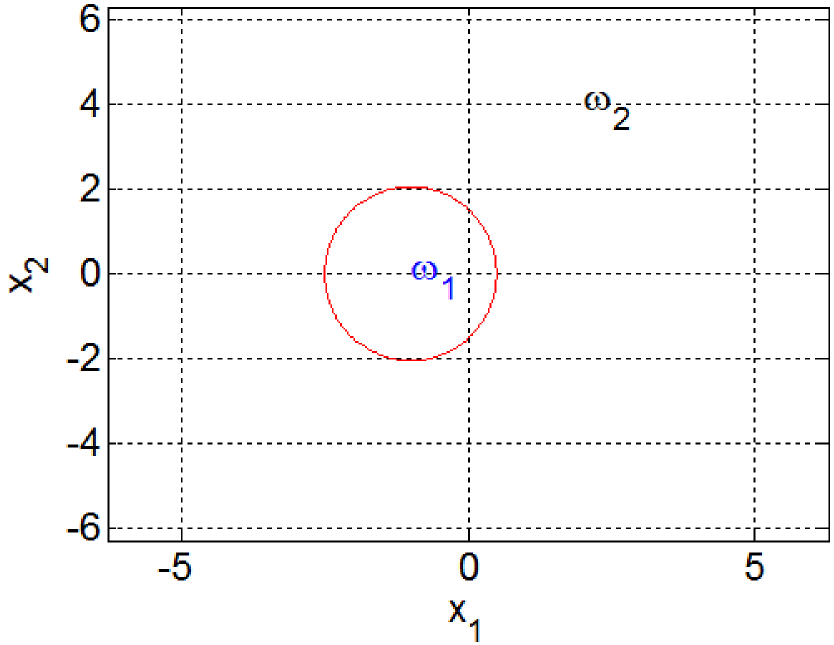
- **Bayesian classifier for normal distributions**
  - **Example** (equiprobable classes)

$$\mu_1 = (0,0)^T, \mu_2 = (1,0)^T, \Sigma_1 = \begin{pmatrix} 0.10 & 0.00 \\ 0.00 & 0.15 \end{pmatrix}, \boxed{\Sigma_2 = \begin{pmatrix} 0.20 & 0.00 \\ 0.00 & 0.25 \end{pmatrix}}$$

$$g_1(x) = -5.0x_1^2 - 3.3x_2^2 + 0.2620$$

$$g_2(x) = -2.5x_1^2 - 2.0x_2^2 + 5.0x_1 - 2.840$$

$$g_1(x) - g_2(x) = -2.500x_1^2 - 1.333x_2^2 - 5.0x_1 + 3.102 = 0$$



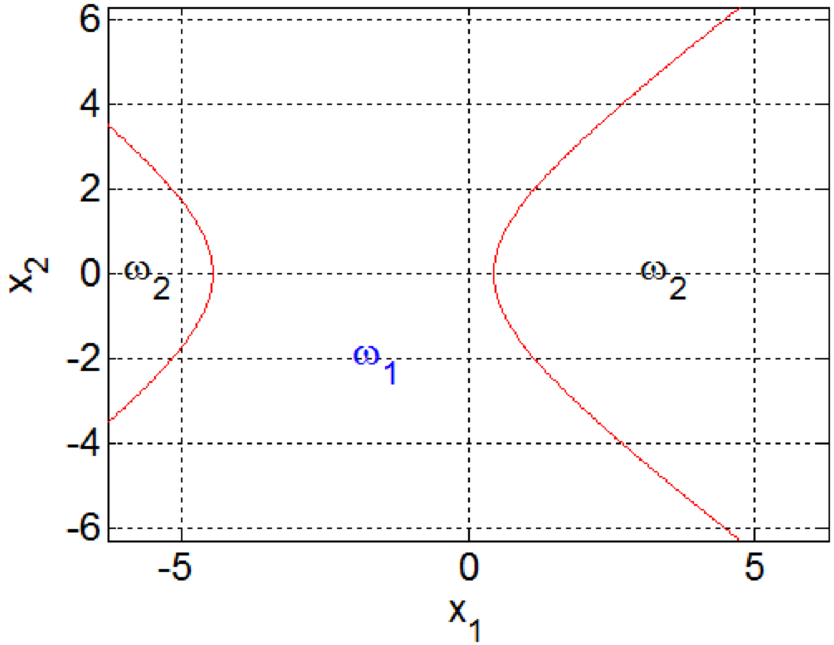
- **Bayesian classifier for normal distributions**
  - **<u>Example</u>** (equiprobable classes)

$$\mu_1 = (0,0)^T, \mu_2 = (1,0)^T, \Sigma_1 = \begin{pmatrix} 0.10 & 0.00 \\ 0.00 & 0.15 \end{pmatrix}, \boxed{\Sigma_2 = \begin{pmatrix} 0.15 & 0.00 \\ 0.00 & 0.10 \end{pmatrix}}$$

$$g_1(x) = -5.0x_1^2 - 3.3x_2^2 + 0.2620$$

$$g_2(x) = -3.333x_1^2 - 5.0x_2^2 + 6.667x_1 - 3.071$$

$$g_1(x) - g_2(x) = -1.667x_1^2 + 1.667x_2^2 - 6.667x_1 + 3.333 = 0$$



- **Bayesian classifier for normal distributions**
  - **Classes with the same covariance matrix:** decision hyperplanes
    - If the classes have the same covariance matrix ($\Sigma = \Sigma_i$) then the **quadratic term** and **part of the constant term** coincide in all discrimination functions:

$$g_i(x) = -\frac{1}{2}x^T\Sigma^{-1}x + \frac{1}{2}x^T\Sigma^{-1}\mu_i + \frac{1}{2}\mu_i^T\Sigma^{-1}x - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \ln p(\omega_i) + c$$

    - Therefore, they disappear from equations $g_i(x) - g_j(x) = 0$.
      This allows us to define **more useful and simpler discrimination functions**:

$$g_i(x) = w_i^T x + w_{i0} \qquad = \alpha_i x_1 + \beta_i x_2 + \gamma_i \quad \text{(case of 2 features)}$$

$$w_i^T = \mu_i^T\Sigma^{-1}, \quad w_{i0} = \ln p(\omega_i) - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i$$

LINEAR CLASSIFIER

    - In this way, the discrimination functions are linear (and not quadratic) and the decision rule turns out to be a **decision hyperplane**: (2D) a straight line, (3D) a plane, ...

$$g_i(x) > g_j(x), \forall j \quad p(\omega_i|x) > p(\omega_j|x), \forall j \Rightarrow x \to \omega_i$$

$\begin{bmatrix}\text{2D case}\\ \text{2 classes}\end{bmatrix}$

$$g_1(x) - g_2(x) = (\alpha_1 - \alpha_2)x_1 + (\beta_1 - \beta_2)x_2 + (\gamma_1 - \gamma_2) > 0 \Rightarrow x \to \omega_1$$

    - Let us have a look at **two cases of the covariance matrix**: (1) $\Sigma = \sigma^2 I$ and (2) any Σ

- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: $\Sigma = \sigma^2 I$
    - Then, **the discrimination functions** take the following form:

$$g_i(x) = \frac{1}{\sigma^2}\mu_i^T x + w_{i0} = \frac{1}{\sigma^2}\mu_i^T x + \ln p(\omega_i) - \frac{1}{2\sigma^2}\mu_i^T \mu_i$$

so that the **decision rules** can be written as:

$$g_{ij}(x) \equiv g_i(x) - g_j(x) = 0$$

$$\Rightarrow \frac{1}{\sigma^2}(\mu_i^T - \mu_j^T)x + \ln p(\omega_i) - \ln p(\omega_j) - \frac{1}{2\sigma^2}(\mu_i^T \mu_i - \mu_j^T \mu_j) = 0$$

$$\Rightarrow (\mu_i - \mu_j)^T x + \sigma^2 \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right) - \frac{1}{2}(\mu_i - \mu_j)^T(\mu_i + \mu_j) = 0$$

$$\Rightarrow (\mu_i - \mu_j)^T \left[x + \sigma^2 \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right)\frac{\mu_i - \mu_j}{\|\mu_i - \mu_j\|^2} - \frac{1}{2}(\mu_i + \mu_j)\right] = 0$$

$$\Rightarrow w^T(x - x_0) = 0$$

$$w = \mu_i - \mu_j, x_0 = \frac{1}{2}(\mu_i + \mu_j) - \sigma^2 \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right)\frac{\mu_i - \mu_j}{\|\mu_i - \mu_j\|^2}$$

Bayesian classification

- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: $\Sigma = \sigma^2 I$
    - Decision rule

$$g_{ij}(x) : w^T(x - x_0) = 0$$

$$w = \mu_i - \mu_j, x_0 = \frac{1}{2}(\mu_i + \mu_j) - \sigma^2 \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right) \frac{\mu_i - \mu_j}{\|\mu_i - \mu_j\|^2}$$

    - **Two-feature case**

$$g_{ij}(x) : w^T(x - x_0) = 0$$

$$\Rightarrow (\mu_{1,1} - \mu_{2,1})x_1 + (\mu_{1,2} - \mu_{2,2})x_2 - (\mu_{1,1} - \mu_{2,1})x_{0,1} - (\mu_{1,2} - \mu_{2,2})x_{0,2} = 0$$

$$\Rightarrow \alpha_{ij}x_1 + \beta_{ij}x_2 + \gamma_{ij} = 0$$

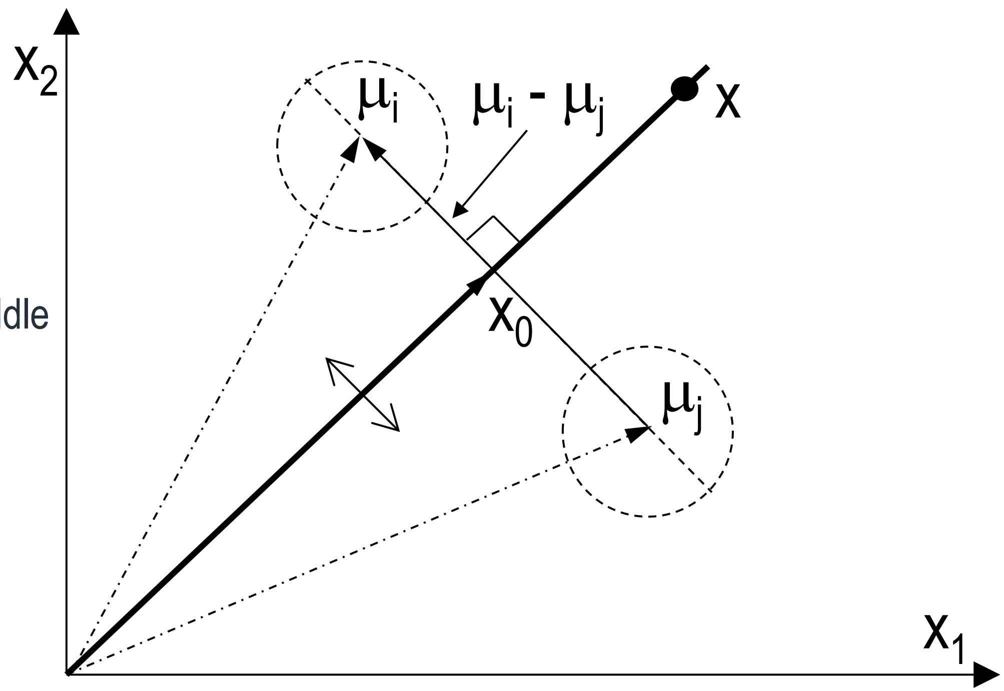
    - Which is this straight line?

- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: $\Sigma = \sigma^2 I$
    - Decision rule for the **two-feature case**

$$g_{ij}(x) : w^T(x - x_0) = 0$$

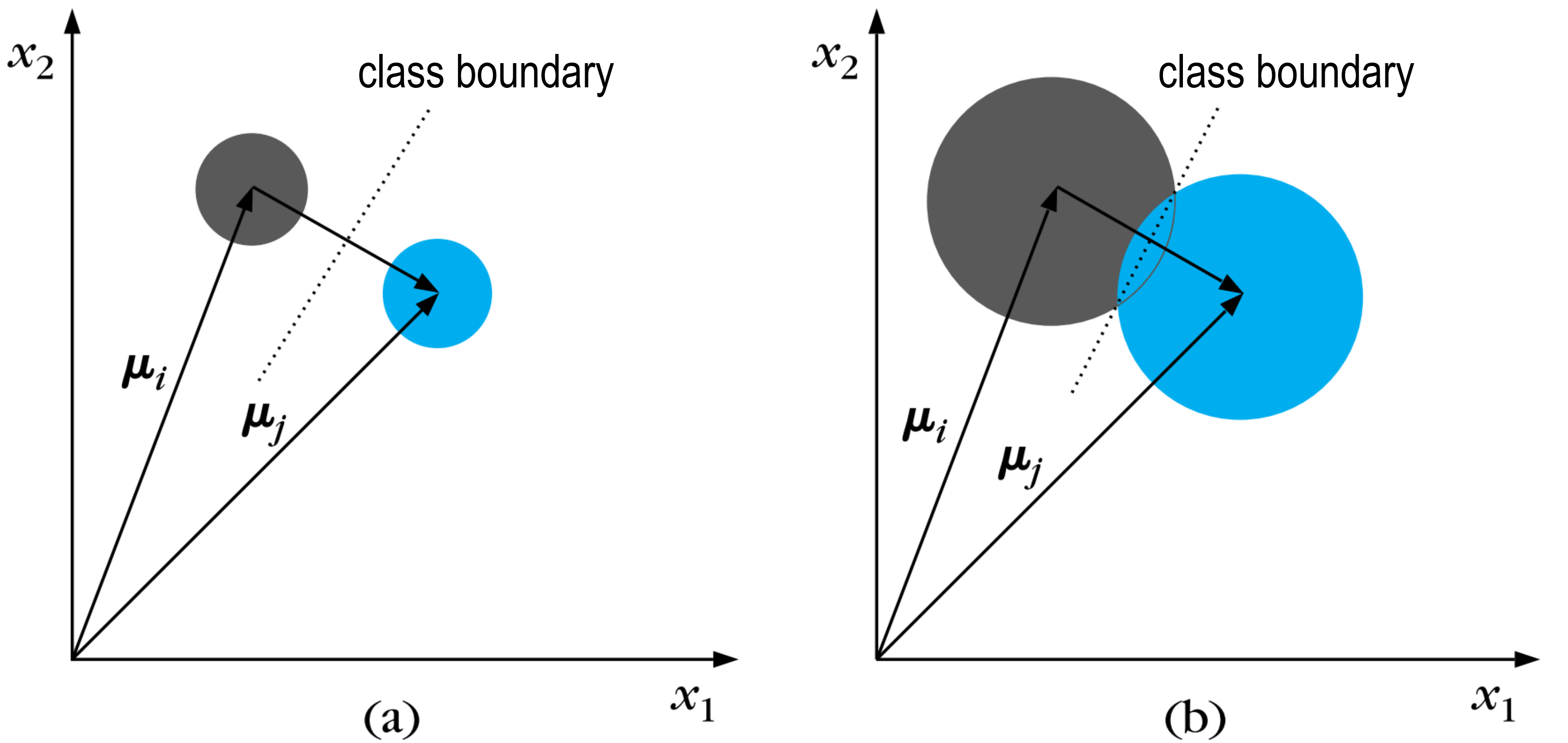
$$w = \mu_i - \mu_j, x_0 = \frac{1}{2}(\mu_i + \mu_j) - \sigma^2 \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right)\frac{\mu_i - \mu_j}{\|\mu_i - \mu_j\|^2}$$

    - Any point x such that $x - x_0$ is orthogonal to $\mu_i - \mu_j$ belongs to the straight line
    - x_0 is always along the vector $\mu_i - \mu_j$
      - if $p(\omega_i) = p(\omega_j)$, x_0 is the middle point between μ_i and μ_j
      - if $p(\omega_i) < p(\omega_j)$, x_0 moves towards μ_i along vector $\mu_i - \mu_j$



Bayesian classification

- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: $\Sigma = \sigma^2 I$
    - the circles expand to $3\sigma \equiv 98\%$



- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: any Σ
    - We recover the original linear discrimination functions:

$$g_i(x) = w_i^T x + w_{i0}$$

$$w_i^T = \mu_i^T \Sigma^{-1}, \quad w_{i0} = \ln p(\omega_i) - \frac{1}{2}\mu_i^T \Sigma^{-1} \mu_i$$

    - Then:

$$\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2 + \mu_1^T\Sigma^{-1}\mu_2 - \mu_2^T\Sigma^{-1}\mu_1$$
$$= (\mu_1 - \mu_2)^T\Sigma^{-1}\mu_1 + (\mu_1 - \mu_2)^T\Sigma^{-1}\mu_2$$
$$= (\mu_1 - \mu_2)^T\Sigma^{-1}(\mu_1 + \mu_2)$$

$$g_{ij}(x) \equiv g_i(x) - g_j(x) = 0$$

$$\Rightarrow (\mu_i - \mu_j)^T \Sigma^{-1} x + \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right) - \frac{1}{2}(\mu_i - \mu_j)^T \Sigma^{-1}(\mu_i + \mu_j) = 0$$

$$\Rightarrow (\mu_i - \mu_j)^T \Sigma^{-1}\left[x + \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right)\frac{\mu_i - \mu_j}{(\mu_i - \mu_j)^T\Sigma^{-1}(\mu_i - \mu_j)} - \frac{1}{2}(\mu_i + \mu_j)\right] = 0$$

$$\Rightarrow w^T(x - x_0) = 0$$

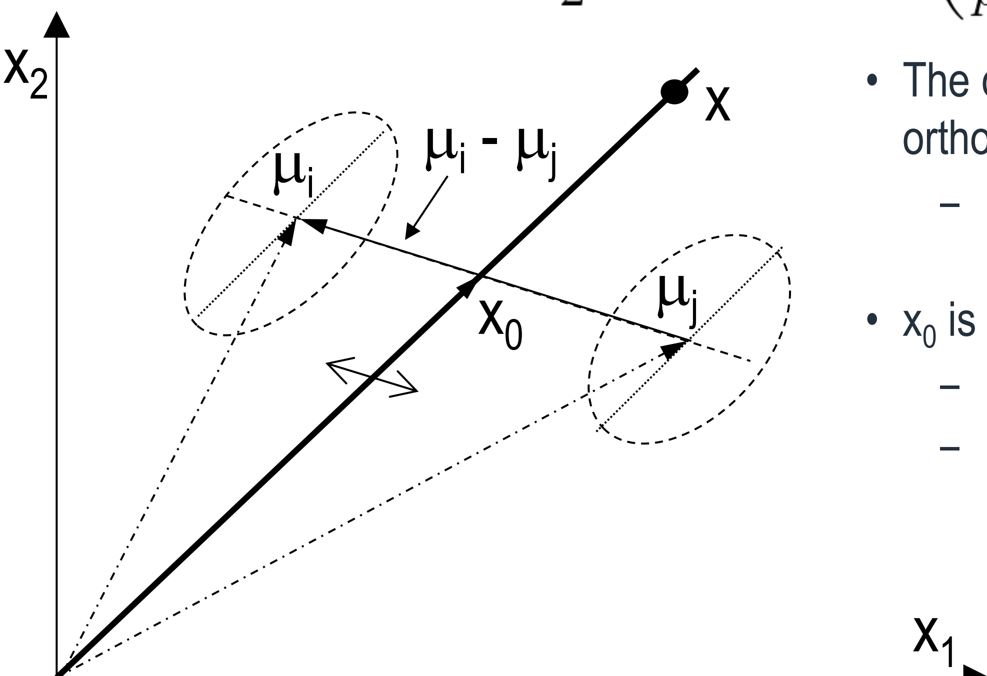
$$w = \Sigma^{-1}(\mu_i - \mu_j), x_0 = \frac{1}{2}(\mu_i + \mu_j) - \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right)\frac{\mu_i - \mu_j}{(\mu_i - \mu_j)^T\Sigma^{-1}(\mu_i - \mu_j)}$$

- **Bayesian classifier for normal distributions**
  - Classes with the **same covariance matrix**: any Σ

$$g_{ij} : w^T (x - x_0) = 0$$

$$w = \Sigma^{-1}(\mu_i - \mu_j)$$

$$x_0 = \frac{1}{2}(\mu_i + \mu_j) - \ln\left(\frac{p(\omega_i)}{p(\omega_j)}\right) \frac{\mu_i - \mu_j}{(\mu_i - \mu_j)^T \Sigma^{-1} (\mu_i - \mu_j)}$$



- The decision hyperplane is no longer necessarily orthogonal to $\mu_i - \mu_j$ but to $\Sigma^{-1}(\mu_i - \mu_j)$
  - $\Sigma^{-1}(\mu_i - \mu_j)$ is the result of transforming $(\mu_i - \mu_j)$ through the matrix Σ^{-1}
- x_0 is always on the vector $\mu_i - \mu_j$
  - if $p(\omega_i) = p(\omega_j)$, x_0 is the average of μ_i y μ_j
  - if $p(\omega_i) < p(\omega_j)$, x_0 moves towards μ_i along $\mu_i - \mu_j$

- **Bayesian classifier for normal distributions**
  - **Example** In a two-dimensional classification problem with two equiprobable classes, the classes follow two normal distributions with the following parameters:

$$\mu_1 = (0,0)^T, \mu_2 = (3,3)^T, \Sigma = \Sigma_1 = \Sigma_2 = \begin{pmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{pmatrix}$$

Classify the sample $x = (1.0{,}2.2)^T$ using a Bayesian classifier.

$$g_{12}(x) = w^T(x - x_0)$$

$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$x_0 = \frac{1}{2}(\mu_1 + \mu_2) - \ln\left(\frac{p(\omega_1)}{p(\omega_2)}\right)\frac{\mu_1 - \mu_2}{(\mu_2 - \mu_2)^T\Sigma^{-1}(\mu_1 - \mu_2)}$$

$$g_{12} = (-3,-3)\begin{pmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{pmatrix}^{-1}\left((1,2.2)^T - (1.5,1.5)^T\right)$$

$$= 0.36 > 0$$

Therefore, $x \rightarrow \omega_1$.

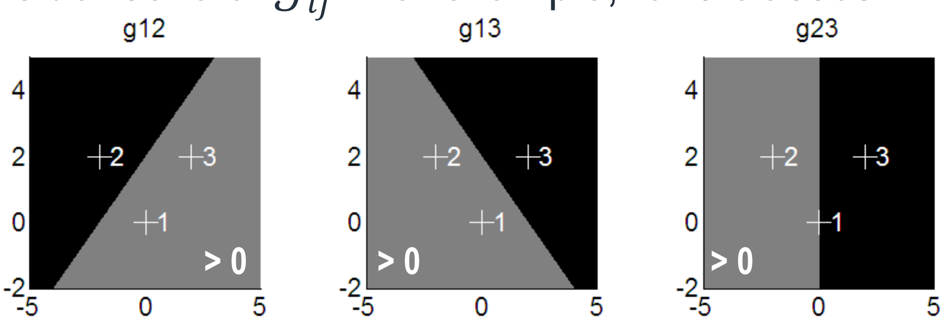
Bayesian classification

- **Bayesian classifier for normal distributions**
  - So far we have considered **two-class cases** and derived the boundaries between class pairs through the discrimination functions:

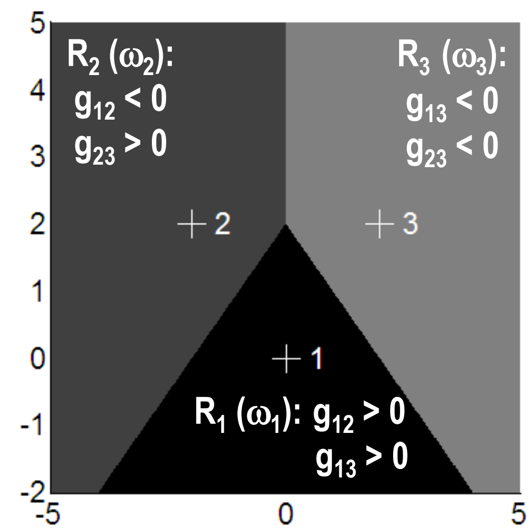
$$g_{ij}(x) = g_i(x) - g_j(x) = 0$$

  - For **two classes** $\begin{array}{l} \text{if } g_{12}(x) > 0, \text{ then } x \rightarrow \omega_1 \\ \text{if } g_{12}(x) < 0, \text{ then } x \rightarrow \omega_2 \end{array} \begin{bmatrix} \text{if } g_1(x) > g_2(x), \text{then } x \rightarrow \omega_1 \\ \text{if } g_1(x) < g_2(x), \text{then } x \rightarrow \omega_2 \end{bmatrix}$
  - **If there are more than 2 classes**, we have to determine $g_i \mid g_i > g_j, \forall j \neq i$. In case of using the joint discrimination functions g_{ij} to decide, we have to consider several g_{ij}. For example, for 3 classes:

μ_i
$(\ \ 0,0)$
$(\ -2,2)$
$(+2,2)$
$\sigma = 1$



    - to determine R_i we need several g_{ik}, g_{ki}
      M classes: M-1 pairs



Bayesian classification

- **Bayesian classifier for normal distributions**
  - **Minimum distance classifiers**
    - We can see the above from another point of view
    - We assume **equiprobable classes with the same covariance matrix**. Then:

$$\left.\begin{aligned} g_i(x) &= c_i - \frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) + \ln p(\omega_i) \\ c_i &= -\frac{L}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| \end{aligned}\right\} \rightarrow$$

$$\rightarrow g_i(x) = -\tfrac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i)$$

- We assign **x** to the class for which the probability is greater $\Rightarrow g_i(x) > g_j(x)\ \forall j \neq i$
  - (1) If $\mathbf{\Sigma = \sigma^2 I}$, $g_i(x)$ is higher the closer it is x to μ_i

    $\Rightarrow$ assign **x** to the class whose center $\boldsymbol{\mu}_i$ is closer (**Euclidean distance**)
  - (2) For generic Σ, we have to assign **x** to the class for which the following expression takes a lowest value: $$d_m^2 = (x-\mu_i)^T \Sigma^{-1}(x-\mu_i)$$
    - $d_m \equiv$ **Mahalanobis distance** (considers the scattering present in the features)

- **Bayesian classifier for normal distributions**
  - <u>**Example**</u> In a two-dimensional classification problem into two equiprobable classes the classes have two normal distributions with the following parameters:

$$\mu_1 = (0,0)^T, \mu_2 = (3,3)^T, \Sigma = \Sigma_1 = \Sigma_2 = \begin{pmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{pmatrix}$$

Classify the vector $\mathbf{x} = (\mathbf{1.0}, \mathbf{2.2})^T$ using a Bayesian classifier.

$$d_m^2(x,\mu_1) = (x-\mu_1)^T \Sigma^{-1} (x-\mu_1)$$
$$= (1.0, 2.2) \begin{pmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{pmatrix} \begin{pmatrix} 1.0 \\ 2.2 \end{pmatrix} = 2.952$$
$$d_m^2(x,\mu_2) = (x-\mu_2)^T \Sigma^{-1} (x-\mu_2)$$
$$= (-2.0, -0.8) \begin{pmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{pmatrix} \begin{pmatrix} -2.0 \\ -0.8 \end{pmatrix} = 3.672$$

- $d_m(x,\mu_1) < d_m(x,\mu_2) \Rightarrow \mathbf{x} \rightarrow \boldsymbol{\omega_1}$.
- <u>REMARK</u>: the Euclidean distances would be $d_e(x,\mu_1) = 2.417$ and $d_e(x,\mu_2) = 2.154$, so, if we used them, we would assign $\mathbf{x} \rightarrow \boldsymbol{\omega_2}$.

Contents

- Introduction
- Bayesian classification
- Estimation of probability density functions
- Linear discriminant functions and the perceptron algorithm

Estimation of probability density functions

- **Estimation of probability density functions** (pdfs)
  - The Bayesian classifier assumes that we have **knowledge on the pdfs of the classes** of the problem
  - There are different methods to get this type of information:
    - The expression of the pdf is known but the parameters are unknown $\rightarrow$ **parametric estimation**
      - **Maximum likelihood estimators**
      - others
    - The expression of the pdf is not known $\rightarrow$ **non-parametric estimation**
      - **Parzen windows** method
      - **k-nearest neighbours** method (KNN)
      - others

Estimation of probability density functions

- **Estimation of probability density functions**
  - **Maximum likelihood estimators**
    - Let us consider an M-class classification problem whose samples are distributed in accordance to $\boldsymbol{p(x|\omega_i;\ \theta_i), i = 1, \dots, M}$, where $\boldsymbol{\theta_i}$ is the **vector of parameters** for class $\boldsymbol{\omega_i}$
      - It's about estimating $\boldsymbol{\theta_i}$ by means of a set of samples $x_1, x_2, \dots, x_k$ from class $\boldsymbol{\omega_i}$
    - We assume that the samples of one class do not affect the estimation of parameters for the other classes in order to formulate the problem irrespective of the class

      $\Rightarrow$ the estimation is repeated for each class
    - In this way, given the statistically independent samples $x_1, x_2, \dots, x_N$ from $\boldsymbol{p(x_k|\theta)}$, we calculate the following joint pdf:

$$p(X;\theta) = p(x_1, x_2, \dots, x_N;\theta) = \Pi_{k=1}^{N} p(x_k;\theta)$$

    - Then, the **maximum likelihood estimator** of θ is given by:

$$\theta_{ML} \mid p(X;\widehat{\theta}_{ML}) = \max\{p(X;\theta)\}$$

      - This represents that θ that better explains samples $x_1, x_2, \dots, x_N$

- **Estimation of probability density functions**
  - **Maximum likelihood estimators**
    - To simplify the calculations we will go once again to the function ln(·) to define the **log-likelihood function**:

$$L(\theta) \equiv \ln \Pi_{k=1}^{N} p(x_k;\theta) = \sum_{k=1}^{N} \ln p(x_k;\theta)$$

    - Now, you can find the derivative of the log-likelihood and equal to 0:

$$\frac{\partial L(\theta)}{\partial \theta} = \sum_{k=1}^{N} \frac{\partial \ln p(x_k;\theta)}{\partial \theta} = \sum_{k=1}^{N} \frac{1}{p(x_k;\theta)} \frac{\partial p(x_k;\theta)}{\partial \theta} = 0$$

    - For sufficiently high N values, the maximum likelihood estimator is **asymptotically unbiased**, follows a **normal distribution** and exhibits **minimal variance**

- **Estimation of probability density functions**
  - **Maximum likelihood estimators**
    - L-dimensional Gaussian distribution with **Σ known**

$$\widehat{\mu}_{ML} = \frac{1}{N}\sum_{k=1}^{N} x_k$$

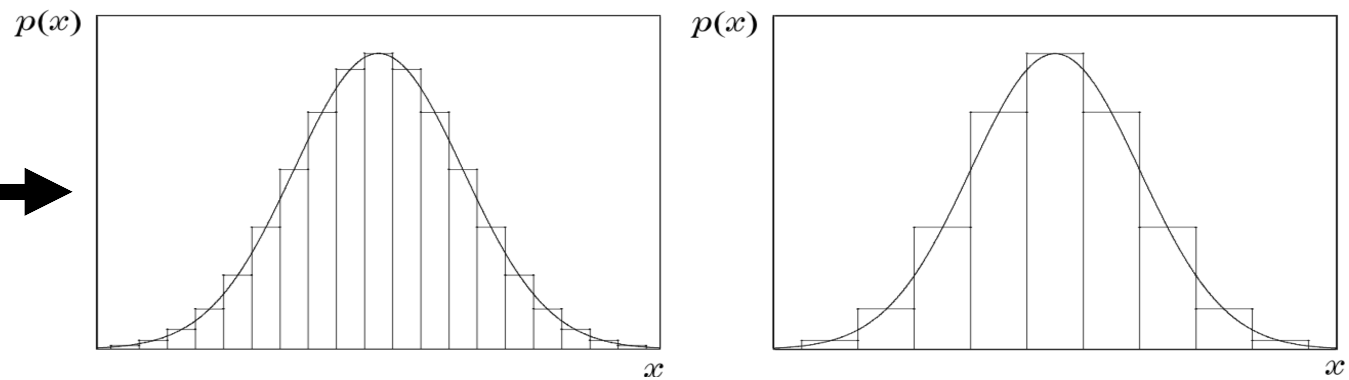
    - L-dimensional Gaussian distribution, **μ and Σ unknown**

$$\widehat{\mu}_{ML} = \frac{1}{N}\sum_{k=1}^{N} x_k, \quad \widehat{\Sigma}_{ML} = \frac{1}{N}\sum_{k=1}^{N}(x_k - \widehat{\mu}_{ML})(x_k - \widehat{\mu}_{ML})^T$$

Estimation of probability density functions

- **Estimation of probability density functions**
  - Non-parametric estimation: **first approximation**
    - It is about estimating a certain pdf **p(x)** without setting any expression for the pdf
    - Let us assume we have **N** independent samples $x_1, x_2, \ldots, x_N$ that come from the pdf that we want to estimate
    - To this end, we build a **histogram** using bins R_j of the same size:

for different sizes of R_j



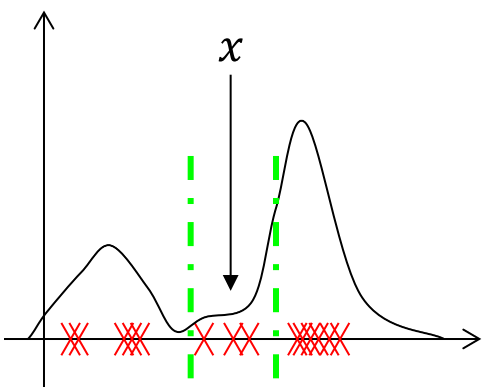
  - $k_{N,Rj}$ = how many of the N samples belong to bin R_j
  - $\{k_{N,Rj}/N\}$ is an aproximation of $P_{Rj} =$ probability that x belongs to R_j:

$$P_{R_j} = p(x \in R_j) = \int_{R_j} p(x')\mathrm{d}x'$$

- **Estimation of probability density functions**
  - Non-parametric estimation: **first approximation**
    - We now define sufficiently small regions R around x, aiming at calculating $p(x)$. In this regard, if $p(x)$ is assumed constant inside R:

$$P_R = \int_R p(x')\mathrm{d}x' \approx p(x) \int_R \mathrm{d}x' = p(x)V$$

      - V is the (hyper)volume occupied by region R (1D – length, 2D – area, 3D – volume, etc.)
      - e.g. if R is a (hyper)cube of dimension L and side length h, $V = h^L$
    - Therefore: $p(x) \approx \dfrac{P_R}{V} = \dfrac{k_{N,R}/N}{V} = p_{N,R}(x)$
    - $p_{N,R}(x) \to p(x)$ as $N \to \infty$ if the following holds:
      - $V \to 0$ (small regions)
      - $k_{N,R} \to \infty$ (sufficient number of samples in each R)
      - $k_{N,R}/N \to 0$ (high total number of samples)
    - **In short**: for each x, $p(x)$ is approximated by defining a small R region around x and counting how many x_i fall into R (= $k_{N,R}$); if $k_{N,R}$ ↑↑ and N ↑↑↑, then $p_{N,R}(x) \approx p(x)$



Estimation of probability density functions

- **Estimación de funciones de densidad de probabilidad**
  - Non-parametric estimation: **Parzen windows** (Parzen, 1962)
    - Let us consider a region R shaped like a (hyper) L-dimensional cube of side h_N. Then:

$$V_N = (h_N)^L$$

    - Let us consider the following function (**box function** or ***kernel***):

$$\varphi(u) = \begin{cases} 1 & |u_j| \leq \frac{1}{2}, j = 1, \ldots, L \\ 0 & \text{otherwise} \end{cases}$$

    - So, for a certain $\boldsymbol{x}$:

$$\varphi\left(\frac{x - x_i}{h_N}\right) = 1 \text{ if } x_i \in \text{(hyper)cube with volume } V_N \text{ centered at } x$$

    - Then, the number of samples that are inside the x-centered (hyper)cube is:

$$k_N(x) = \sum_{i=1}^{N} \varphi\left(\frac{x - x_i}{h_N}\right)$$

- **Estimation of probability density functions**
  - Non-parametric estimation: **Parzen windows**
    - At last:

$$p_N(x) = \frac{k_N(x)/N}{V_N} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_N^L}\varphi\left(\frac{x - x_i}{h_N}\right)$$

    - Therefore, given a certain x, to obtain the estimate of $p(x)$: (1D case)

```
def parzen_box_1D(x,X,h):
# x = point where to evaluate the PDF
# X = table of samples
# h = side of the (hyper)cube

  N = X.shape[0]; kn = 0
  for i in range(N):
    if abs((x - X[i])/h) <= 0.5:
      kn = kn+1
  p = (kn/N)/h
  return p
```

- **Estimation of probability density functions**
  - Non-parametric estimation: **Parzen windows**
    - Comments on p_N:
      - p_N **is a legitimate pdf**

1. $p_N(x) \geq 0, \forall x$ (obvious, it is a sum of 1's)
2. $\int p_N(x) \mathrm{d}x = 1$

$$\int \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^L} \varphi\left(\frac{x - x_i}{h_N}\right) \mathrm{d}x = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^L} \int \varphi\left(\frac{x - x_i}{h_N}\right) \mathrm{d}x =$$

$$= \{u = x - x_i\} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^L} \int \varphi\left(\frac{u}{h_N}\right) \mathrm{d}u =$$

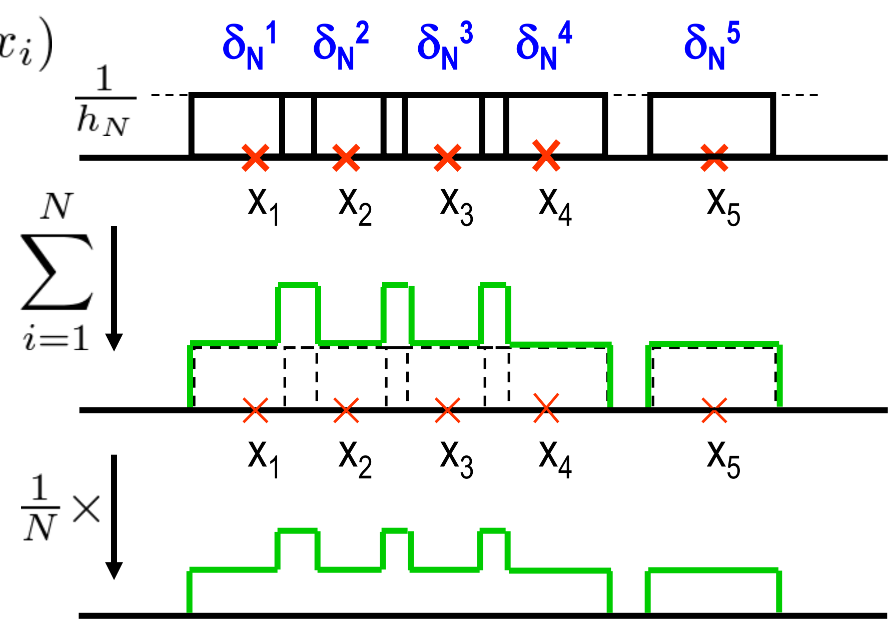
$$= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^L} \int_{(-h_N/2,\ldots,-h_N/2)}^{(+h_N/2,\ldots,+h_N/2)} \mathrm{d}u = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^L} h_N^L = 1$$

Estimation of probability density functions

- **Estimation of probability density functions**
  - Non-parametric estimation: **Parzen windows**
    - Comments on p_N:
      - function $p_N(x)$ **can be seen as the average of N functions centred in the samples** x_i

$$p_N(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_N^L}\varphi\left(\frac{x-x_i}{h_N}\right)$$

$$\begin{cases} p_N(x) = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\delta_N(x-x_i) \\ \delta_N(u) = \dfrac{1}{h_N^L}\varphi\left(\dfrac{u}{h_N}\right) \end{cases}$$



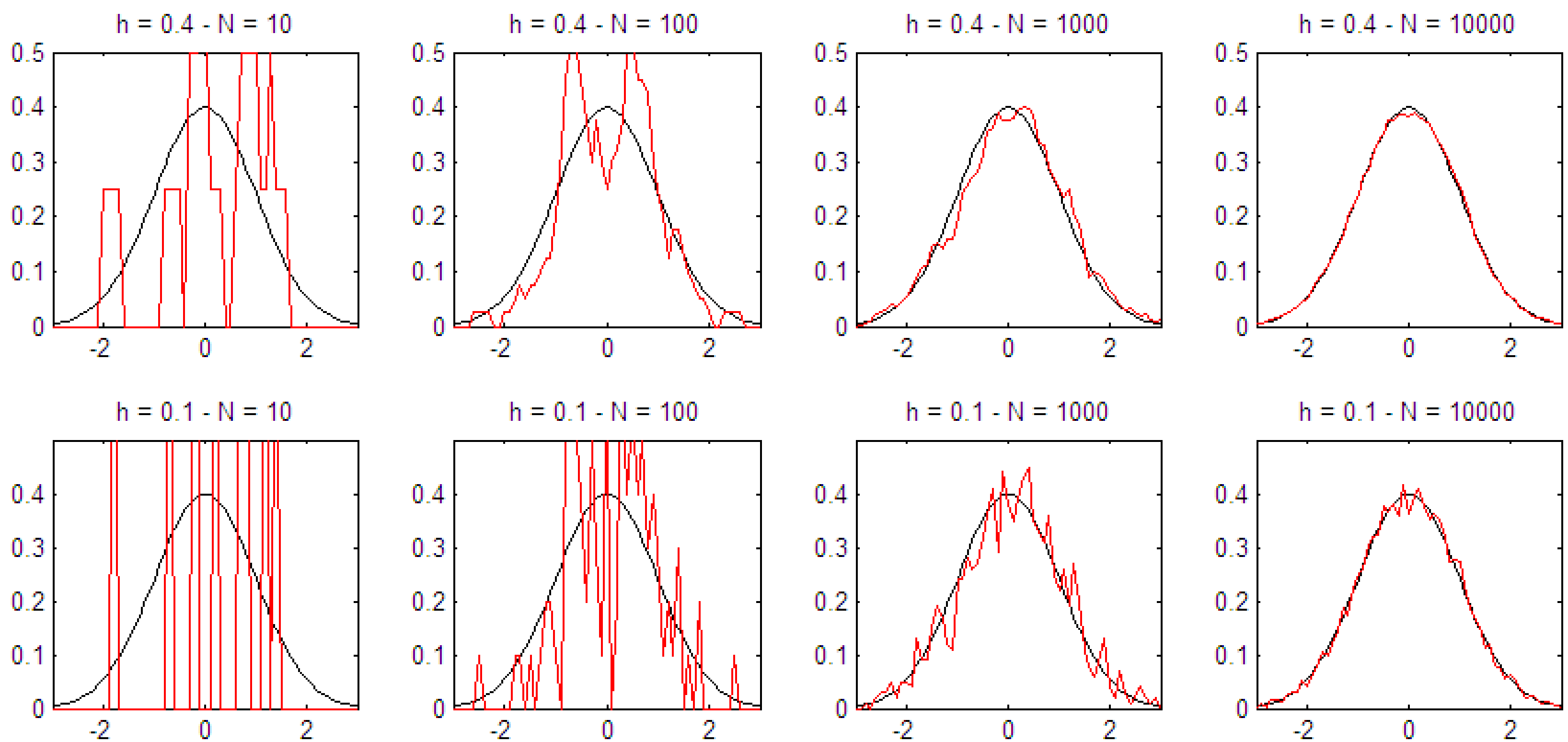
      - if h_N is **large**, the amplitude of δ_N is **small** and p_N is the superposition of N *wide pulses*
      - if h_N is **small**, the amplitude of δ_N is **large** and p_N is the superposition of N *narrow pulses*
      - $h_N \to 0 \Rightarrow \delta_N \to \delta$

Estimation of probability density functions

- **Estimation of probability density functions**
  - – Non-parametric estimation: **Parzen windows**
    - **Example**: N random values extracted from a distribution $N(0,1)$ – rectangular *kernel*

softer

noisier



better estimation →

- **Estimation of probability density functions**
  - Non-parametric estimation: **Parzen windows**
    - As we have already seen in the previous example, when approximating continuous functions [$p(\cdot)$] by discontinuous kernel functions [φ($\cdot$)], the resulting estimate also presents **discontinuities**
    - To avoid this, it is suggested to use **continuous kernels φ(·)**
      - It can be shown that the resulting estimate $p_N(x)$ is a legitimate pdf if:

$$\varphi(u) \geq 0 \text{ and } \int \varphi(u)\mathrm{d}u = 1$$

      - One of the most commonly used kernels is the Gaussian kernel (mean 0, variance 1):

$$\varphi(u) = \frac{1}{\sqrt{(2\pi)^L}} e^{-\frac{1}{2}u^T u}$$

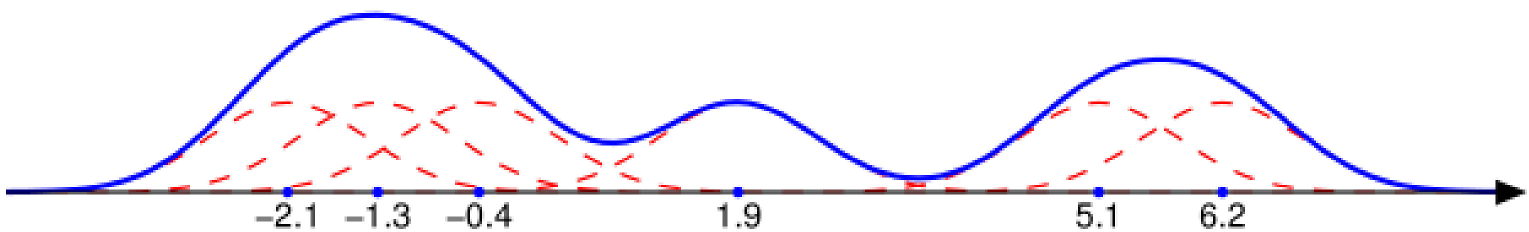
$$p_N(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_N^L}\varphi\left(\frac{x-x_i}{h_N}\right) = \frac{1}{N\,h_N^L}\sum_{i=1}^{N}\frac{1}{\sqrt{(2\pi)^L}}e^{-\frac{1}{2}u_i^T u_i}$$

$$\left(u_i = \frac{x-x_i}{h_N}\right)$$

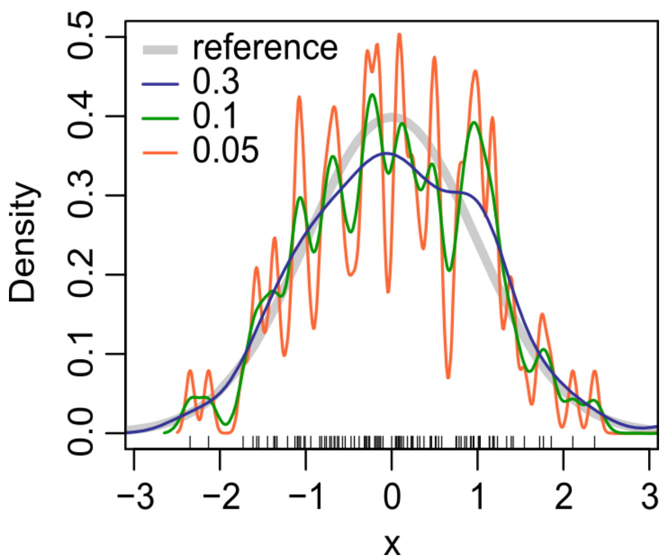
```
def parzen_gauss_1D(x,X,h):
# x = point where to evaluate the PDF
# X = data samples
# h = side of the (hyper)cube
  N = X.shape[0]; kn = 0
  for i in range(N):
      kn += 1/(sqrt(2*pi))*exp(-0.5*((x-X[i])/h)**2)
  p = (kn/N)/h
  return p
```

Estimation of probability density functions

- **Estimation of probability density functions**
  - Non-parametric estimation: **Parzen windows**
    - Now $p_N(x)$ is obtained as the average of N Gaussians centered on the samples x_i



    - Effect of varying bandwidth h_N

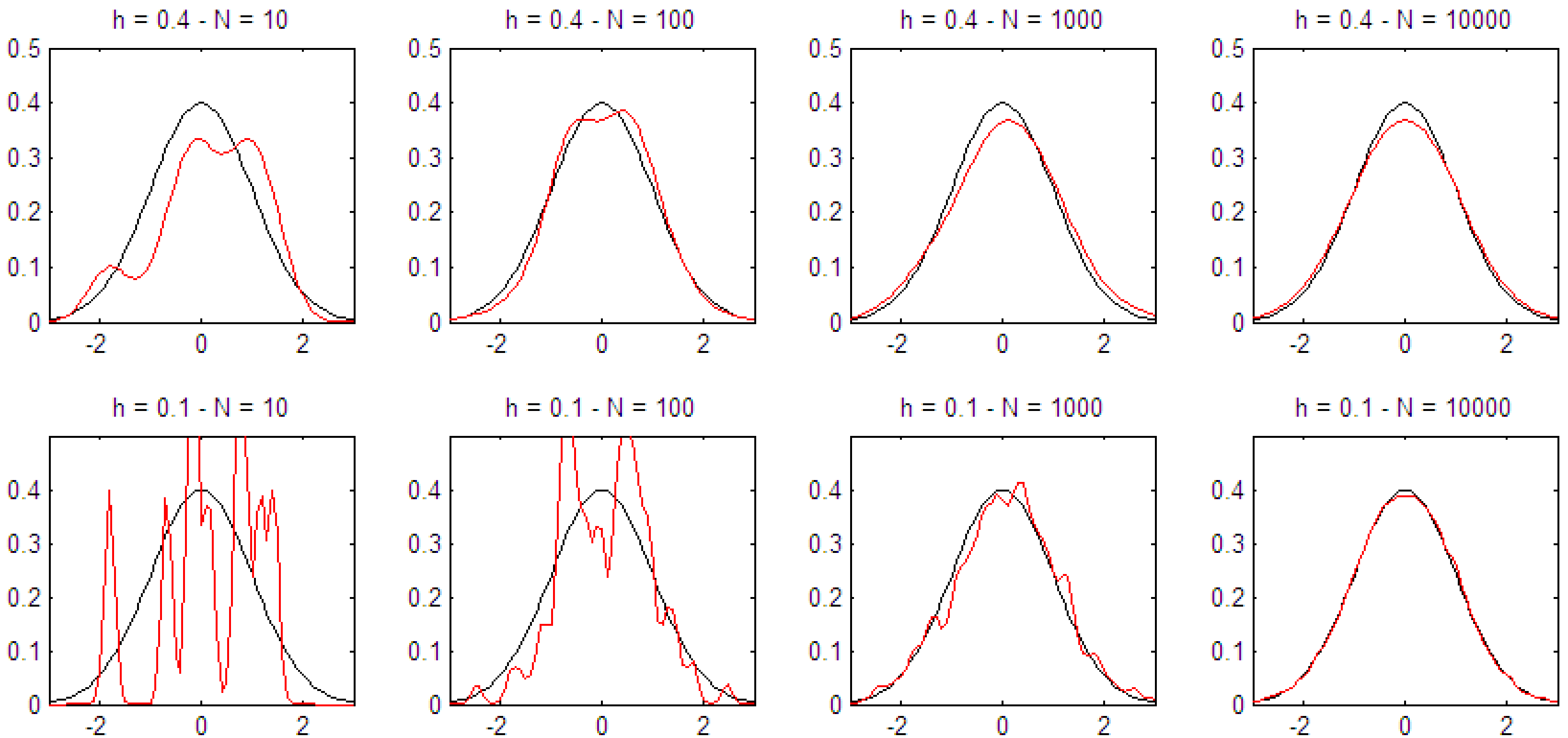


Estimation of probability density functions

- **Estimation of probability density functions**
  – Non-parametric estimation: **Parzen windows**
    - **Example**: N random values extracted from a distribution $N(0,1)$ – Gaussian *kernel*

softer

noisier



better estimation →

- **Estimation of probability density functions**
  - Non-parametric estimation: **k nearest neighbours**
    - Given x and a collection of samples $x_1, x_2, \dots, x_N$ from a certain pdf **p(x)**, to estimate **p(x)**:
      - **Parzen windows method** – first, a search volume is set around x, V_N, and next we determine the number of samples k_N belonging to that volume

We set V_N and find k_N → $$p_N(x) = \frac{k_N/N}{V_N}$$

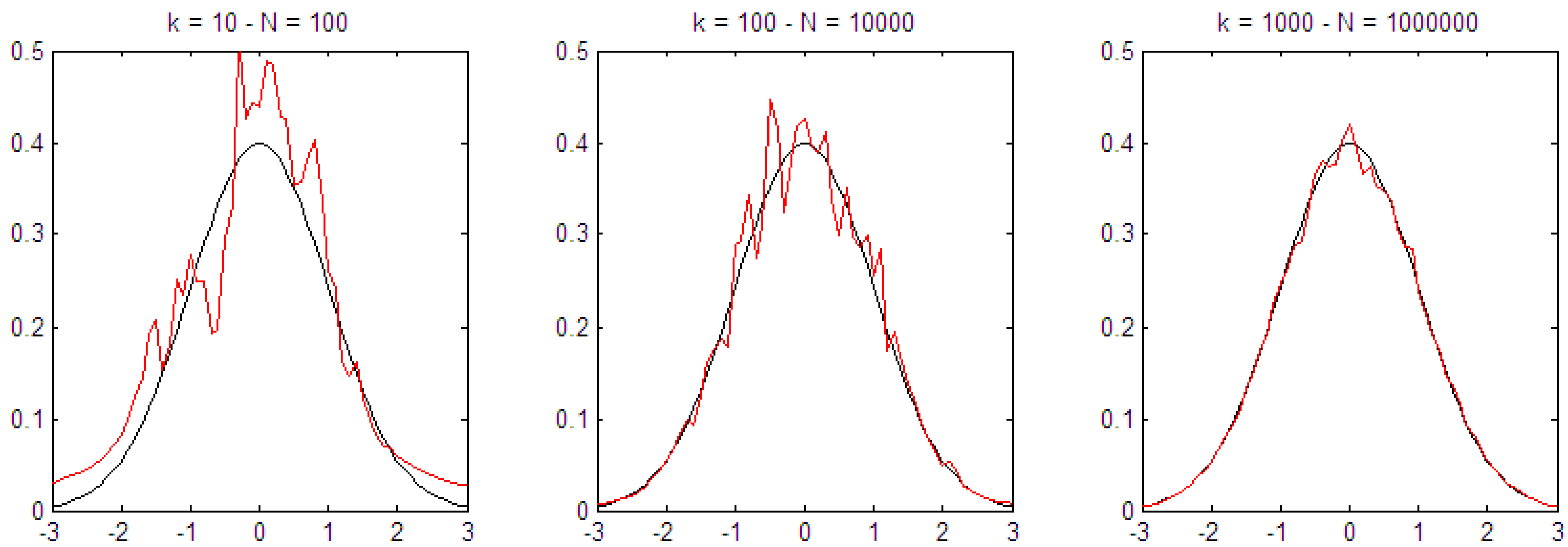
- **Method of the k nearest neighbors** – we first find the k_N samples nearest to x, and next we determine the minimal volume V_{kN} where they are contained in

We set k_N and find V_N → $$p_N(x) = \frac{k_N/N}{V_{kN}}$$

```
def knn_1D(x,X,k):
# x = point where to evaluate the PDF
# X = data samples
# k = number of neighbours
  N = X.shape[0]; d = []
  for i in range(N):
      d.append(abs(x-X[i]))
  d.sort()
  V = 2 * d[min(N,k)-1]
  p = (k/N)/V
  return p
```

Estimation of probability density functions

- **Estimation of probability density functions**
  - Non-parametric estimation: **k nearest neighbours**
    - **<u>Example</u>**: N random values sampled from a distribution $N(0,1)$



best estimate $\rightarrow$

- In this case, it has been used: $k_N = \sqrt{N}$
- In the case of Parzen windows, it is suggested to use: $h_N = \frac{h_1}{\sqrt{N}}$

Contents

- Introduction
- Bayesian classification
- Estimation of probability density functions
- Linear discriminant functions and the perceptron algorithm

Linear discrimination functions and the perceptron algorithm

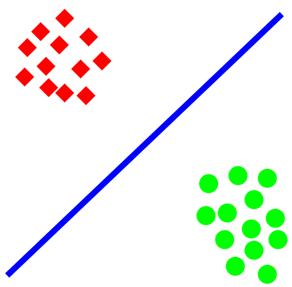
- We have already seen that, depending on the pdf 's of the classes (Gaussian case), a Bayesian classifier can derive in a set of **linear discrimination functions**. For example, for 2 classes:

$$g_{12}(x) = w^T(x - x_0), \quad g_{12}(x) = \begin{cases} > 0 & x \in \omega_1 \\ < 0 & x \in \omega_2 \end{cases}$$

  - simple and computationally very interesting classifier
- In this section, we concentrate again on linear discrimination functions, but from a different perspective: **we do not assume any pdf for the classes**
  - Therefore, regardless of the pdf of the classes, we expect them to be separable by (hyper)planes (1D – point, 2D – straight line, 3D – plane, etc.)
    - In this case it is said that the **classes are linearly separable**
  - We will see how you can find a (hyper)plane that separates the classes from each other (**perceptron algorithm**)

Linear discrimination functions and the perceptron algorithm

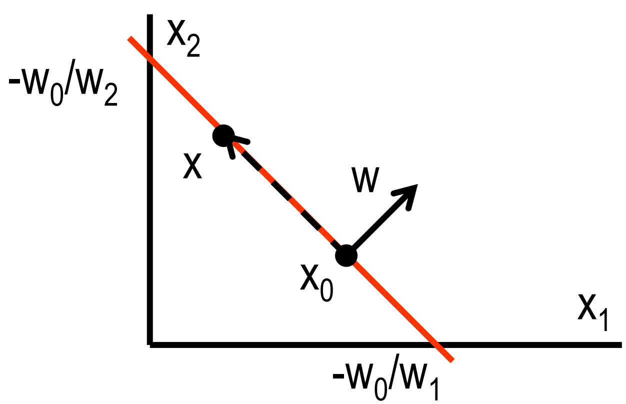
- **Linear discrimination functions**
  - **Goal**: find a (hyper)plane that allows us to separate the training samples in 2 classes



$$g_{12}(x) = w^T(x - x_0), \quad g_{12}(x) = \begin{cases} > 0 & x \in \omega_1 \\ < 0 & x \in \omega_2 \end{cases}$$

(hyper)plane: $x \mid g_{12}(x) = w^T(x - x_0) = 0$

  - For example, for $L = 2$ features:



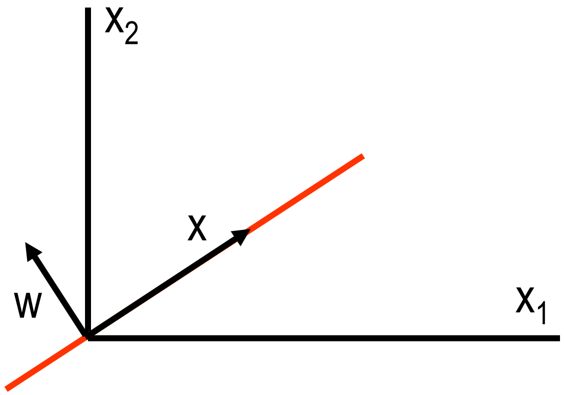
$$(w_1, w_2)\left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} x_{01} \\ x_{02} \end{pmatrix}\right] = 0$$

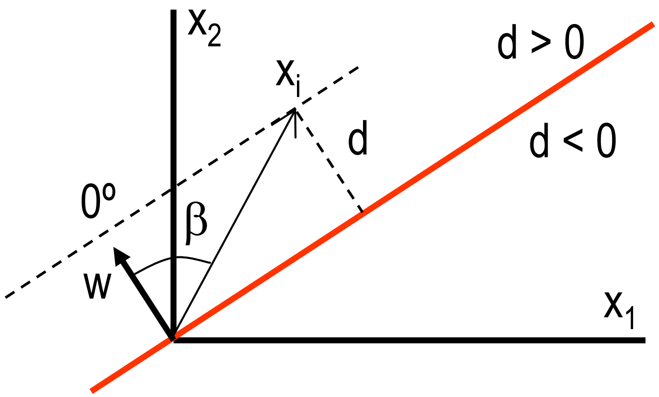
$$w_1x_1 + w_2x_2 - (w_1x_{01} + w_2x_{02}) = 0$$

$$w_1x_1 + w_2x_2 + w_0 = 0 \qquad \textbf{(normal form)}$$

$$x_2 = \left(-\frac{w_1}{w_2}\right)x_1 + \left(-\frac{w_0}{w_2}\right) \qquad \textbf{(slope-intercept form)}$$
$$= ax_1 + b$$

- **Linear discrimination functions**. For now, we will only consider hyperplanes which go through the origin

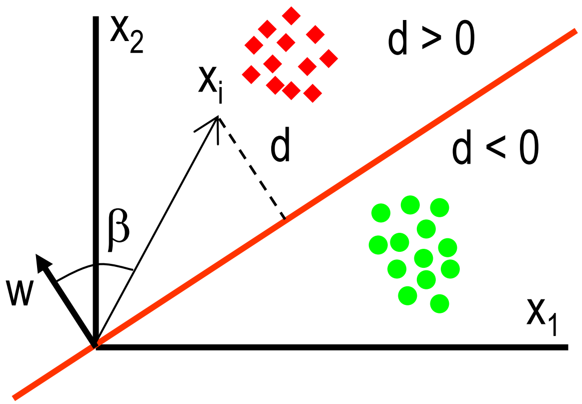




  - This means that $x_0 = (0{,}0)^T$ and thus in $w^T(x - x_0)$
$$w_0 = w^T x_0 = 0$$
$$\Downarrow$$
$$w_1 x_1 + w_2 x_2 = 0$$
$$\Downarrow$$
$$w^T x = 0$$
  - For any point x_i outside the hyperplane we can state:
$$w^T x_i = \|w\| \|x_i\| \cos\beta \neq 0$$
$$\Downarrow \qquad \|w\| = 1$$
$$\|x_i\| \cos\beta = d$$
  - The sign of d depends on the **relative position** of x_i with regard to the hyperplane:
$$\begin{cases} \beta \in \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right] & w^T x_i = d > 0 \\ |\beta| > \frac{\pi}{2} & w^T x_i = d < 0 \end{cases}$$
  - $|d| = |w^T x_i|$ indicates **how far away the sample is** from the (hyper)plane of discrimination

Linear discrimination functions and the perceptron algorithm

- We assume that the classes ω_1 and ω_2 are **linearly separable**, i.e. the hyperplane exists
- The goal is thus to find a function $g_{12}(x) = w^T x$ such that $g_{12}(x)$ is as follows for each sample x_i of the training set:



$$g_{12}(x_i) = w^T x_i > 0, \forall x_i \in \omega_1$$

$$g_{12}(x_i) = w^T x_i < 0, \forall x_i \in \omega_2$$

- g_{12} defined in this way is also named as a **discrimination function**, which in this case turns out to be linear, and thus it is a **linear discrimination function**

- To find the hyperplane, we consider the following function (**perceptron cost**):

$$J(w) = \sum_{x_i \in \mathcal{Y}} (\delta_{x_i} w^T x_i)$$

where:
- $\mathcal{Y}$ is the set of samples x_i wrongly classified by w
- $\delta_{x_i} = -1$ if $x_i \in \omega_1$ y $\delta_{x_i} = +1$ if $x_i \in \omega_2$
- $J(w) \geq 0, \forall w$ ($x_i \in \omega_1$ but $x_i \to \omega_2$, then $\delta_{x_i} \omega^T x_i = (-1)(< 0) > 0$)
- $J(w) = 0$ if all samples are well classified ($\mathcal{Y} = \emptyset$)
- $J(w)$ is piece-wise linear $\Rightarrow$ minimization is not trivial

- The **perceptron algorithm** (Rosenblatt, 1950s) is able to find, through the next iterative approach, the required hyperplane: (sort of **gradient descent**)

$$w(t+1) = w(t) - \rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i$$
$$w(0) = \text{any vector of } \mathbb{R}^L$$

- The algorithm converges if the classes are **linearly separable** and if the sequence of learning factor values ρ_t meets certain conditions:

$$\lim_{t\to\infty} \sum_{k=0}^{t} \rho_k = \infty, \quad \lim_{t\to\infty} \sum_{k=0}^{t} \rho_k^2 < \infty$$

where t denotes iteration number.
  - The sequence ρ_t determines the convergence speed.
  - For instance, $\rho_t = c/t$ and $\rho_t = \rho$ (ρ bounded) meet those conditions.

Linear discrimination functions and the perceptron algorithm

- **Details** on the use of the (basic) perceptron algorithm:

  (1a) To deal with hyperplanes that do not contain the origin, the feature vectors must be **augmented** in one additional dimension $x_i^* = (x_i, 1)^T$. In this way:

$$w^T x + w_0 = 0 \equiv \overbrace{(\underbrace{w_1, w_2, \ldots, w_L}_{w^T}, w_0)}^{(w^*)^T} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_L \\ 1 \end{pmatrix} = (w^*)^T x^* = 0$$

❖ We will use w^T instead of $(w^*)^T$ and x instead of x^* to simplify the notation

  (1b) The rule for modifying w within every iteration can be generically implemented as follows:

1 iteration

$$w(t+1) = w(t) - \rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i \Rightarrow$$

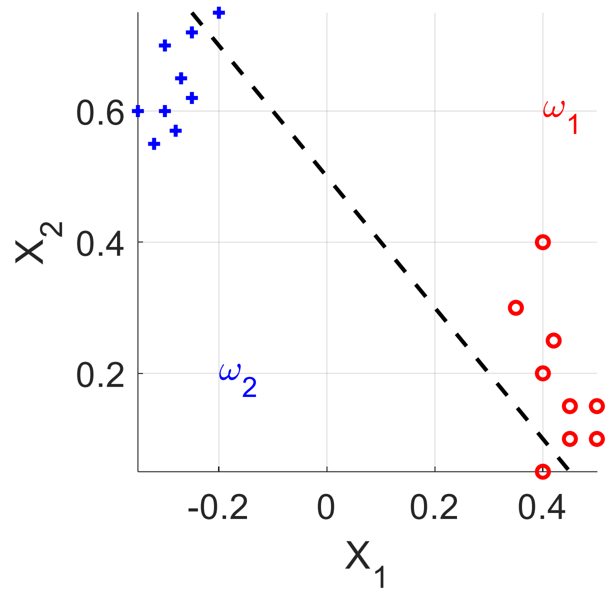
$$\begin{array}{l}
S = (\overbrace{0, 0, \ldots, 0}^{L+1})^T \\
\textbf{for } i = 1 \textbf{ to } \text{n_training_samples} \\
\quad \textbf{if } x_i \in \omega_1 \textbf{ and } w(t)^T x_i < 0 \textbf{ then } S = S + x_i \\
\quad \textbf{if } x_i \in \omega_2 \textbf{ and } w(t)^T x_i > 0 \textbf{ then } S = S - x_i \\
\textbf{end for} \\
w(t+1) = w(t) + \rho_t\, S
\end{array}$$

Linear discrimination functions and the perceptron algorithm

- **Details** of the (basic) perceptron algorithm:
  - **(1) <u>Example 1</u>**
    - The dashed line corresponds to:

$$(1, 1, -0.5)x =$$
$$x_1 + x_2 - 0.5 = 0$$

where $w(t) = (1,1,-0.5)^T$ is the result of the previous step of the perceptron algorithm using $\rho_t = \rho = 0.7$



    - The incorrectly classified samples are: **$(0.40,0.05)^T$** and **$(-0.20,0.75)^T$**
    - The new iteration yields

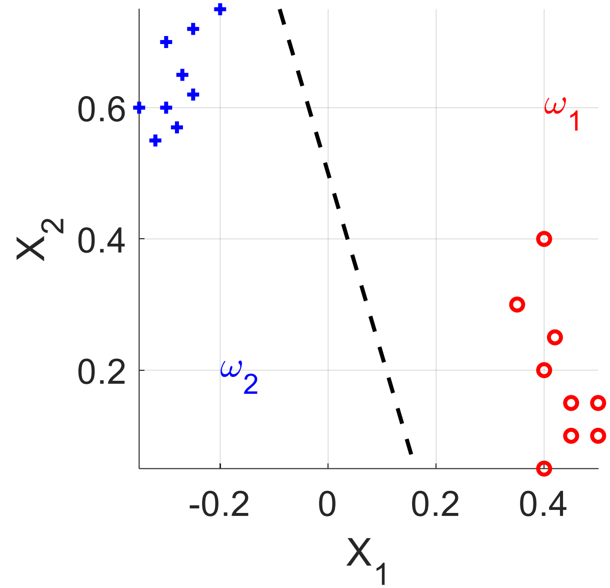
$$\boxed{\begin{aligned} w(t+1) =& w(t) - \\ & -\rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i \end{aligned}}$$

$$w(t+1) = \begin{pmatrix} 1 \\ 1 \\ -0.5 \end{pmatrix} - 0.7(-1) \begin{pmatrix} 0.4 \\ 0.05 \\ 1 \end{pmatrix} - 0.7(+1) \begin{pmatrix} -0.2 \\ 0.75 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.42 \\ 0.51 \\ -0.5 \end{pmatrix}$$

- **Details** of the (basic) perceptron algorithm:
  - **(1) Example 1**
    - The dashed line corresponds to:

$$(1,1,-0.5)x = $$
$$x_1 + x_2 - 0.5 = 0$$

where $w(t) = (1,1,-0.5)^T$ is the result of the previous step of the perceptron algorithm using $\rho_t = \rho = 0.7$



- The incorrectly classified samples are: **(0.40,0.05)**^{T} and **(−0.20,0.75)**^{T}
- The new iteration yields

$$w(t+1) = w(t) - \rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i$$

$$w(t+1) = \begin{pmatrix} 1 \\ 1 \\ -0.5 \end{pmatrix} - 0.7(-1)\begin{pmatrix} 0.4 \\ 0.05 \\ 1 \end{pmatrix} - 0.7(+1)\begin{pmatrix} -0.2 \\ 0.75 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.42 \\ 0.51 \\ -0.5 \end{pmatrix}$$

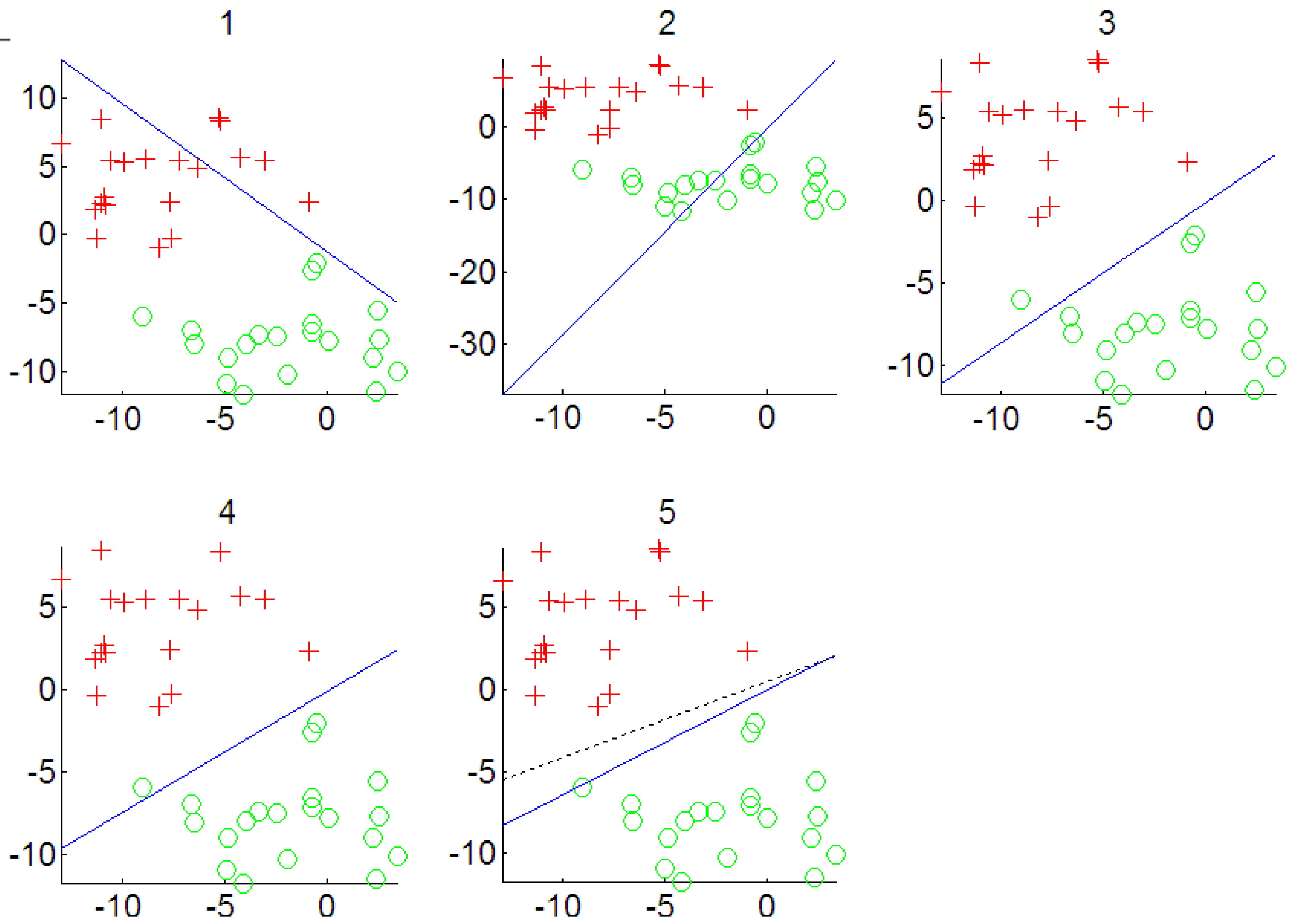
- The resulting hyperplane classifies correctly all the samples and the algorithm ends with:

$$1.42x_1 + 0.51x_2 - 0.5 = 0$$

- **Details** of the (basic) perceptron algorithm:
  - **(1) Example 2**

$\rho = 1.5$

w(0) =	+0.74	+0.67	+0.92
w(1) =	-218.29	+77.27	+23.42
w(2) =	-152.98	+179.38	+11.42
w(3) =	-139.43	+188.33	+9.92
w(4) =	-125.88	+197.29	+8.42



- **Details** of the (basic) perceptron algorithm:
  - **(1) Implementation**

$$w(t+1) = w(t) - \rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i$$

```
def perceptron_2D(X, y, rho, nit):

    N = X.shape[0]
    w = np.zeros(3)
    for t in range(nit):
        S = np.zeros(3); ic = 0
        for i in range(N):
            xs = [X[i,0], X[i,1], 1]
            if np.dot(w,xs) < 0 and y[i] == 1:     # 1 ≡ w1
                S = S + xs; ic += 1
            elif np.dot(w,xs) >= 0 and y[i] == 0: # 0 ≡ w2
                S = S - xs; ic += 1
        if ic == 0: # Y = empty set
            break
        else:
            w = w + rho * S

    w = w / sqrt(w[0] ** 2 + w[1] ** 2)
    return w
```

- **Details** of the (basic) perceptron algorithm:
  (2) The **pocket algorithm**: dealing with non-linearly separable datasets
    - Stops after a number of iterations (T), providing the best hyperplane which has been found along that number of iterations
    - Partially solves the **convergence problem** of the original perceptron algorithm when the classes are not linearly separable

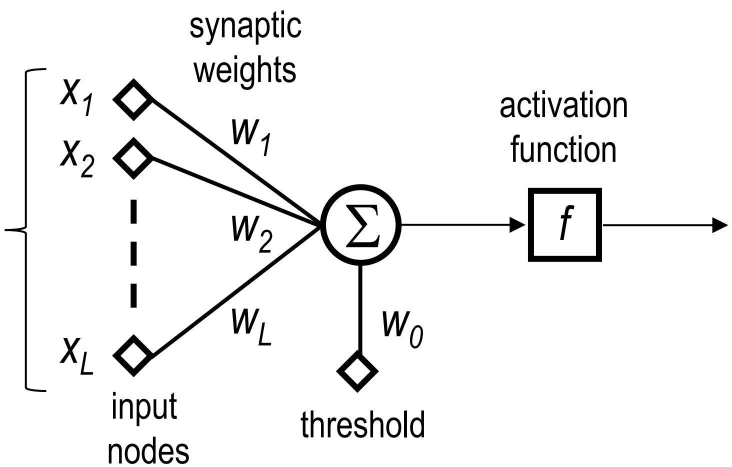
$$
\begin{array}{ll}
(1) & \text{Initialize } w(0) \text{ randomly} \\
(2) & w_s = w(0) \\
 & h_s = \text{no. samples correctly classified by } w(0) \\
(3) & \textbf{for } t = 0 \textbf{ to } T \\
 & (3.1)\; w(t+1) = w(t) - \rho_t \sum_{x_i \in \mathcal{Y}} \delta_{x_i} x_i \\
 & (3.2)\; h = \text{no. samples correctly classified by } w(t+1) \\
 & (3.3)\; \textbf{if } h > h_s \\
 & \qquad\quad \textbf{then } w_s = w(t+1), h_s = h \\
 & \qquad\quad \textbf{end if} \\
(4) & \textbf{end for}
\end{array}
$$

Linear discrimination functions and the perceptron algorithm

- **Classification device**:
  - Once the perceptron algorithm has converged with e.g. $w = (w_1, w_2, \ldots, w_L, w_0)$, then the following structure can be considered to implement the classification operation:

$x = (x_1, x_2, \ldots, x_L)$

perceptron or
[artificial] **neuron**
(McCulloch-Pitts, 1943)



$$w^T x + w_0 > 0, x \rightarrow \omega_1$$

$$w^T x + w_0 < 0, x \rightarrow \omega_2$$

e.g. hard limiter

$$f(u) = \begin{cases} -1 & u < 0 \\ +1 & u > 0 \end{cases}$$

  - The perceptron can be considered as the basic building element for more complex learning machines, e.g. **neural networks**

Lecture 3.1
Supervised learning:
Bayesian and linear classifiers

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

Alberto ORTIZ RODRÍGUEZ