



SAPIENZA
UNIVERSITÀ DI ROMA

METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING IN FISICA

S. Giagu - AA 2019/2020
Lezione 7: 18.3.2020

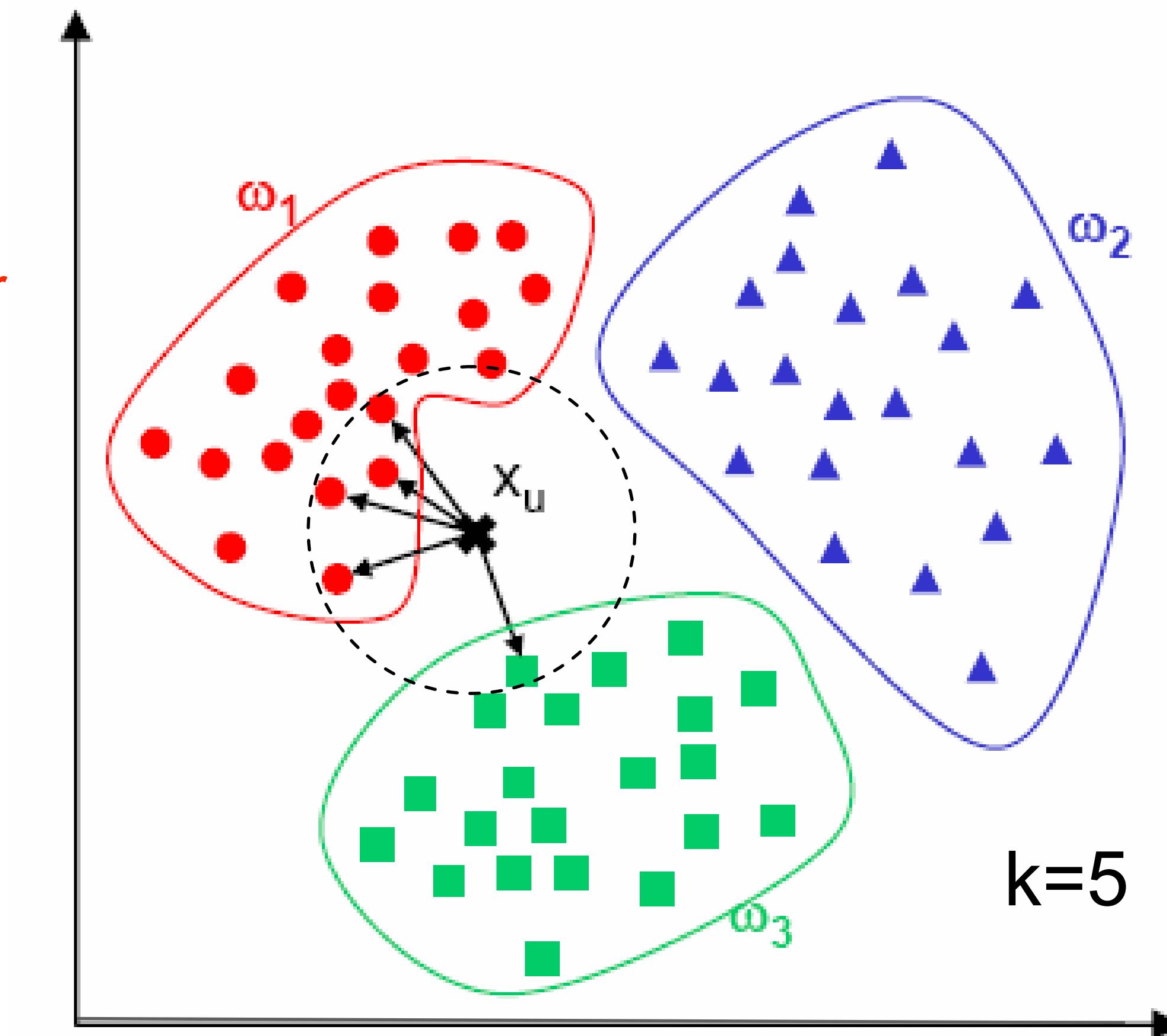
CLASSIFICATORE K-NN CLASSIFIER

- algoritmo k-NN di tipo lazy basato su stima non parametrica delle pdf, che classifica eventi sulla base della loro similitudine con gli eventi di un training set T

- Parametri:

- k: numero di primi vicini (nearest-neighbour)
- distanza nello spazio delle feature:

$$D(a, b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$



ESEMPIO CLASSIFICATORE K-NN IN SCIKIT-LEARN

https://www.dropbox.com/s/2n3ge1ykjdj1g31s/kNN_classification_example.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

IRIS dataset disponibile in scikit-learn

```
# import example data: iris dataset
# consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepa
1
# length, stored in a 150x4 numpy.ndarray
# rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Pe
tal Width.
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

print (X[0])
print(y[0])
print (X.size)
```

[5.1 3.5]
0
300



è il parametro k

```
n_neighbors = 15 ←  
h = .02 # step size in the mesh  
  
# Create color maps  
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])  
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])  
  
for weights in ['uniform', 'distance']:  
    # we create an instance of Neighbours Classifier and fit the data.  
    # weight = uniform --> All points in each neighborhood are weighted equally  
    # weight = distance --> weight points by the inverse of their distance (closer neighbors o  
f a query point will have a greater influence)  
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)  
    clf.fit(X, y)
```



```

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

#np.arange: Return evenly spaced values within a given interval
# np.meshgrid: Return coordinate matrices from coordinate vector
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# np.c_ : translates slice objects to concatenation along the second axis
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

print(Z.shape)

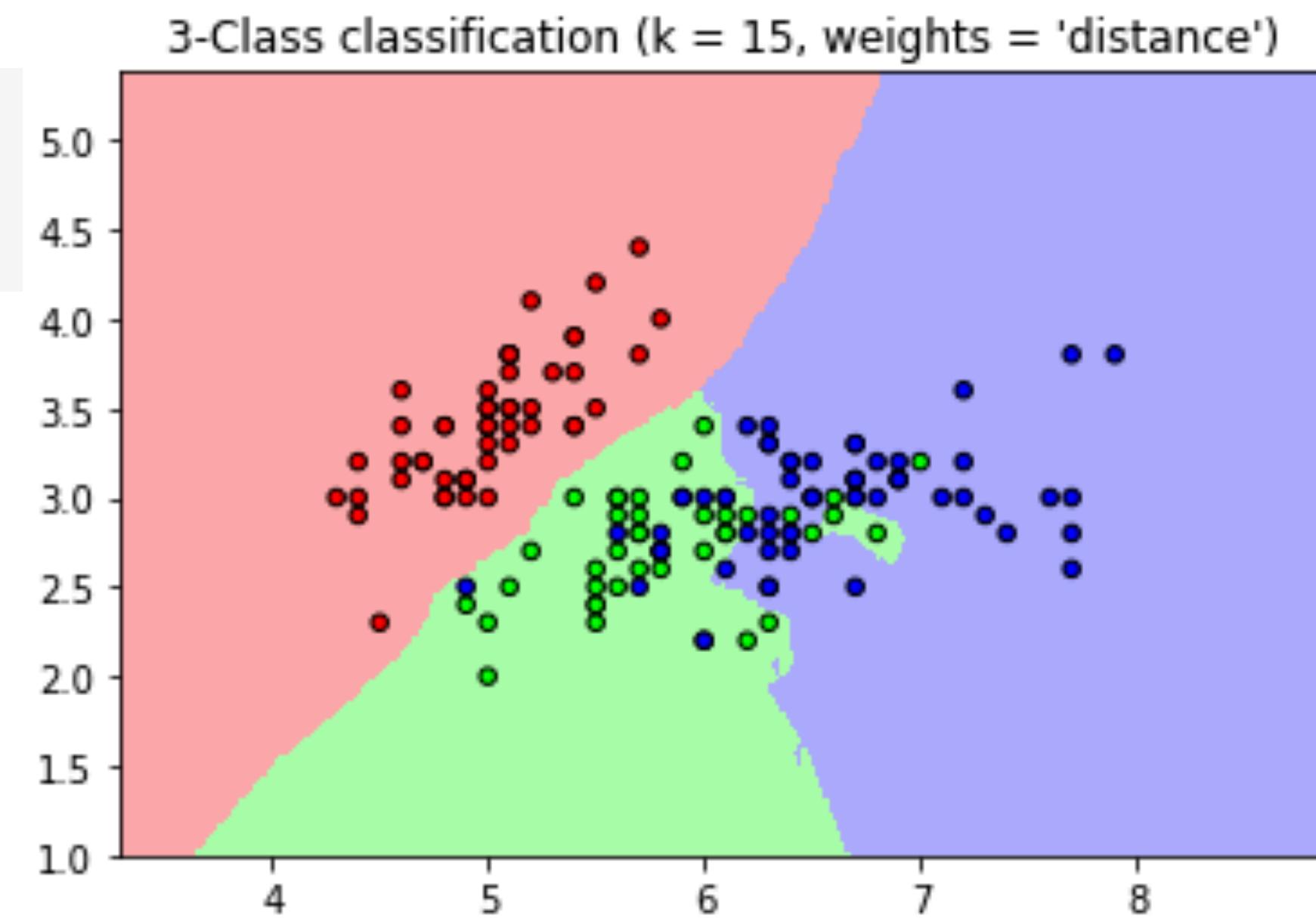
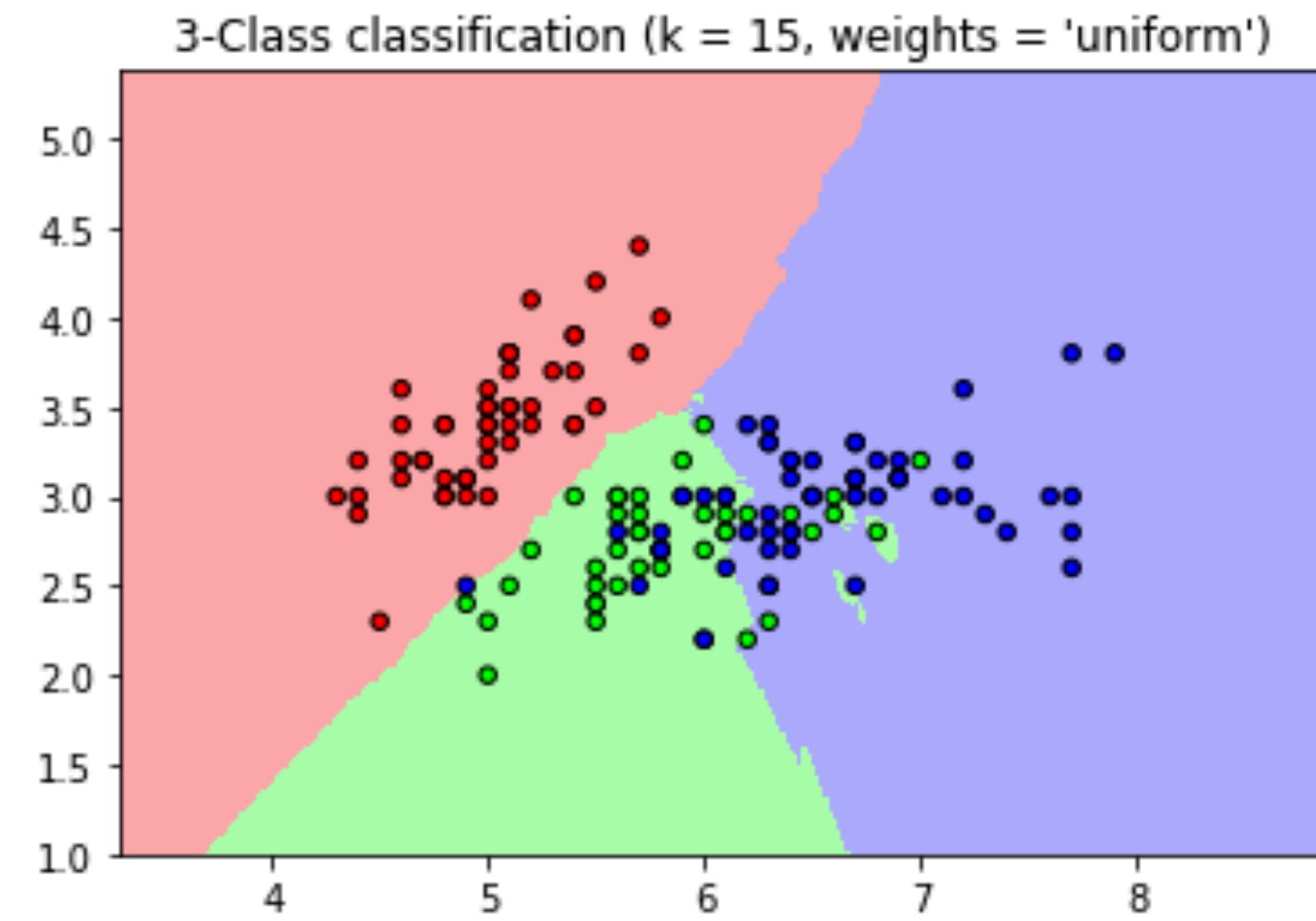
# Put the result into a color plot
Z = Z.reshape(xx.shape)
print(Z.shape)

plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i, weights = '%s')"
          % (n_neighbors, weights))

plt.show()

```



```
# predict output
n_neighbors = 2
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')
clf.fit(X, y)

predicted = clf.predict(X)

print('predicted: ', predicted, ' vs target:', y)
```

```
score = clf.score(X, y)
print('accuracy: ', score)
```

accuracy: 0.8733333333333333



FUNZIONI DISCRIMINANTI

- Una funzione discriminante è una funzione $g(x)$ che prende in input un dato vettore di feature x e lo assegna ad una tra C classi di appartenenza
- in questo modo un classificatore a C -classi può essere descritto come un sistema (algoritmo) che calcola C funzioni discriminanti $g_i(x)$ $\{i=1,\dots,C\}$, e assegna x alla classe con il valore più grande su tale set di funzioni

$$x \in \omega_i \text{ se } g_i(x) > g_j(x) \quad \forall j \neq i$$

- esempio: classificatore bayesiano $g_i(x) = -R(\alpha_i|x)$ oppure $g_i(x) = P(\omega_i|x)$
- per problemi a 2 classi serve una sola funzione discriminante:

$$g(\mathbf{x}) \equiv g_1(\mathbf{x}) - g_2(\mathbf{x}) \begin{cases} > 0 \\ < 0 \end{cases}$$

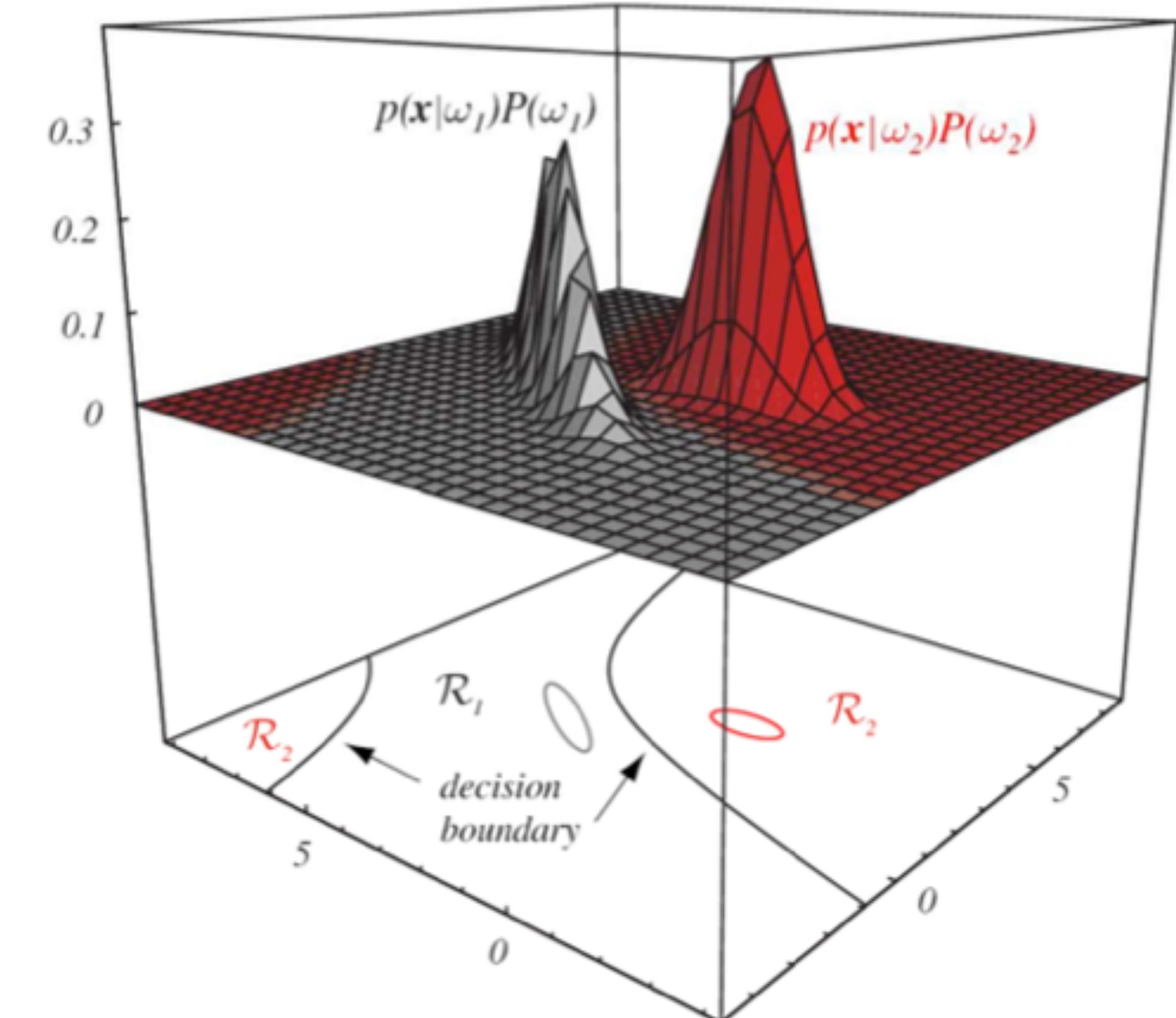


FUNZIONI DISCRIMINANTI E REGIONI DI DECISIONE

- la funzione discriminante immediatamente definisce le **regioni di decisione** e i **confini (decision boundaries)** tra le classi ω_i e ω_j :

$$R_i(\mathbf{x}) = \{\mathbf{x} | g_i(\mathbf{x}) > g_j(\mathbf{x}) \forall j \neq i\}$$

$$\Gamma_{ij}(\mathbf{x}) = \{\mathbf{x} | g_i(\mathbf{x}) = g_j(\mathbf{x}) \ j \neq i\}$$



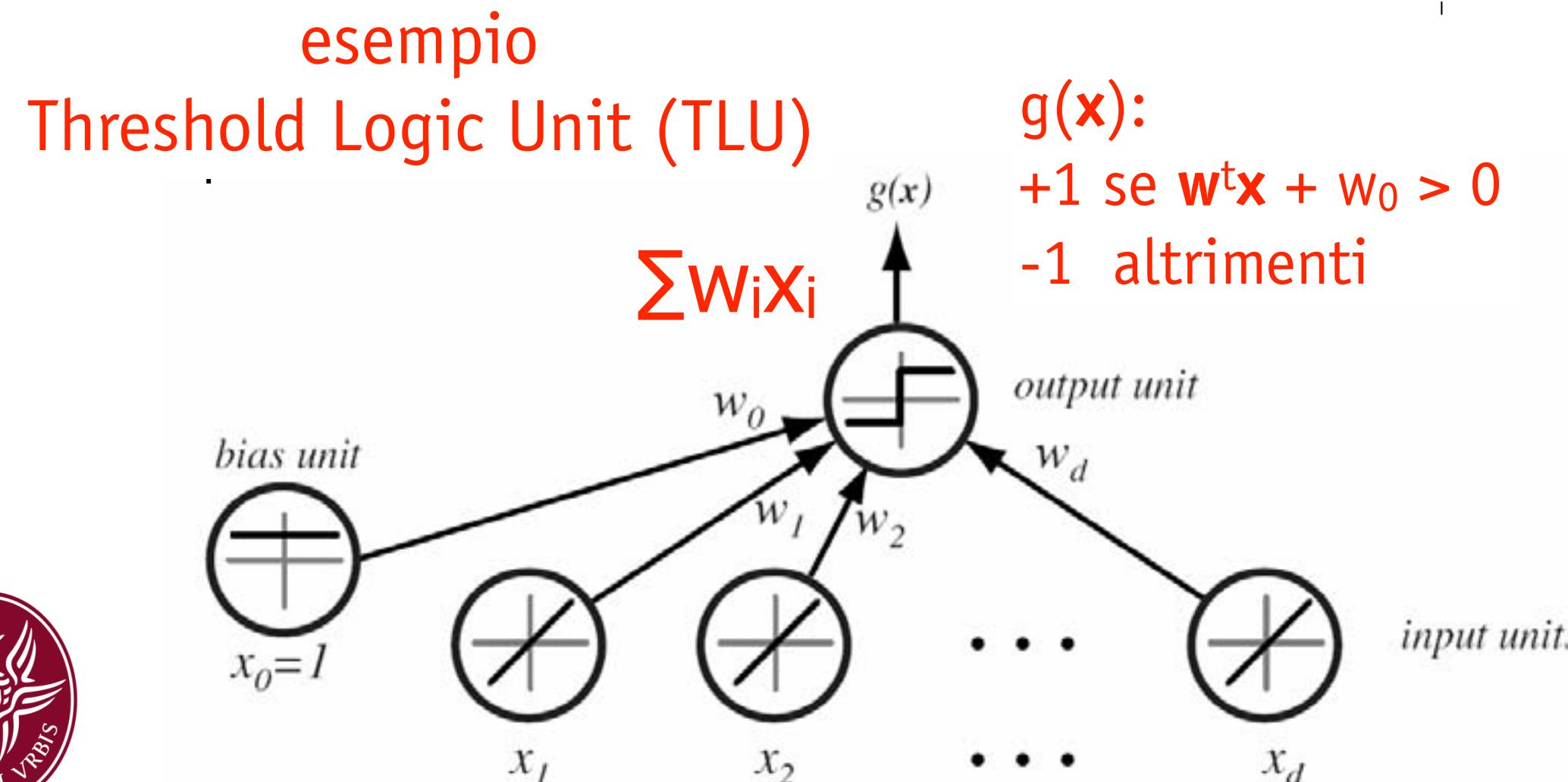
TRAINING DI UN ALGORITMO BASATO SU FUNZIONI DISCRIMINANTI

- il training degli algoritmi discussi nella precedente lezione (Naive Bayes, k-NN, ...) consisteva nel calcolo di una approssimazione locale delle distribuzioni di densità di probabilità sulla base del campione di training T, e nell'applicare a tali distribuzioni il test del LR di bayes
- con le funzioni discriminanti invece si stimano, sulla base del training set T, direttamente le funzioni $g(x)$
- l'approccio è parametrico:
 - si sceglie un **modello rappresentativo** per la funzione discriminante: $g(x) = g(x, w)$ con w set di iper-parametri (pesi) che descrivono la forma funzionale di g
 - il problema dell'apprendimento è quindi posto come un problema di minimizzazione di una opportuno criterio, rappresentato tramite i valori di una **funzione obiettivo (loss function)**, rispetto agli eventi del training set T
 - esempi: errore di classificazione, accuracy, MSE, cross-entropia, ...
- NOTA IMPORTANTE:
 - l'obiettivo finale è sempre quello di riuscire a classificare con la massima accuratezza possibile eventi che non fanno parte di T e che il classificatore non ha mai visto in precedenza
 - Un piccolo errore su T non implica necessariamente un piccolo errore di generalizzazione ...



MODELLI LINEARI

- Il più semplice modello rappresentativo per una funzione discriminante è quello della **funzione discriminante lineare (FDL)**
- supponiamo per semplicità che il problema sia a 2 classi, la generica FDL è data da :
 - $g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^t \mathbf{x} + w_0$
 - con \mathbf{w} vettore dei pesi e w_0 chiamato bias ($-w_0$ viene anche chiamato threshold weight)
- la regola di classificazione è fatta sulla base del segno di $g(\mathbf{x})$: $g(\mathbf{x}) > 0$ equivalente a $\mathbf{w}^t \mathbf{x} > -w_0$
- training: minimizzazione rispetto al vettore \mathbf{w} e sul training set $T=\{\mathbf{x}_i, y_i\}$, della **Loss function** $L(g(\mathbf{x}, \mathbf{w}), y)$ che quantifica le prestazioni del classificatore in funzione dei valori del vettore \mathbf{w}



NOTA: una conveniente ridefinizione spesso usata:

augmented feature vector

$\mathbf{x} \rightarrow \mathbf{z} = [1, \mathbf{x}]$ e $\mathbf{w} \rightarrow \mathbf{w} = [w_0, \mathbf{w}]$

da cui:

$g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^t \mathbf{x} + w_0 \rightarrow g(\mathbf{z}, \mathbf{w}) = \mathbf{w}^t \mathbf{z}$

ESEMPIO: FDL CON FEATURE BIDIMENSIONALI

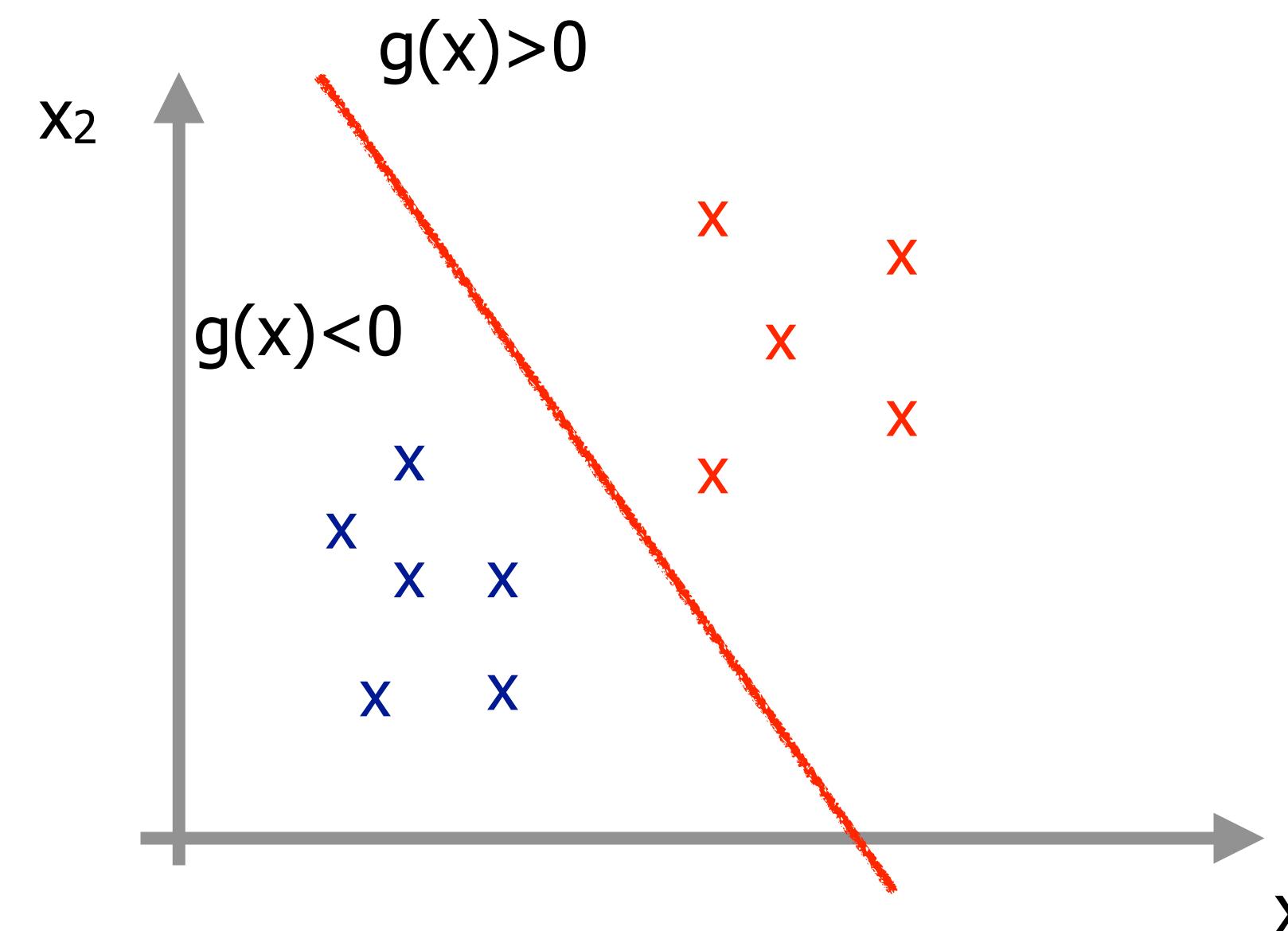
- problema 2 classi con vettore di feature bidimensionale: $\mathbf{x} = (x_1, x_2)$:

$$g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^t \mathbf{x} + w_0 = x_1 \cdot w_1 + x_2 \cdot w_2 + w_0 \quad \mathbf{w} = (w_1, w_2)$$

oppure:

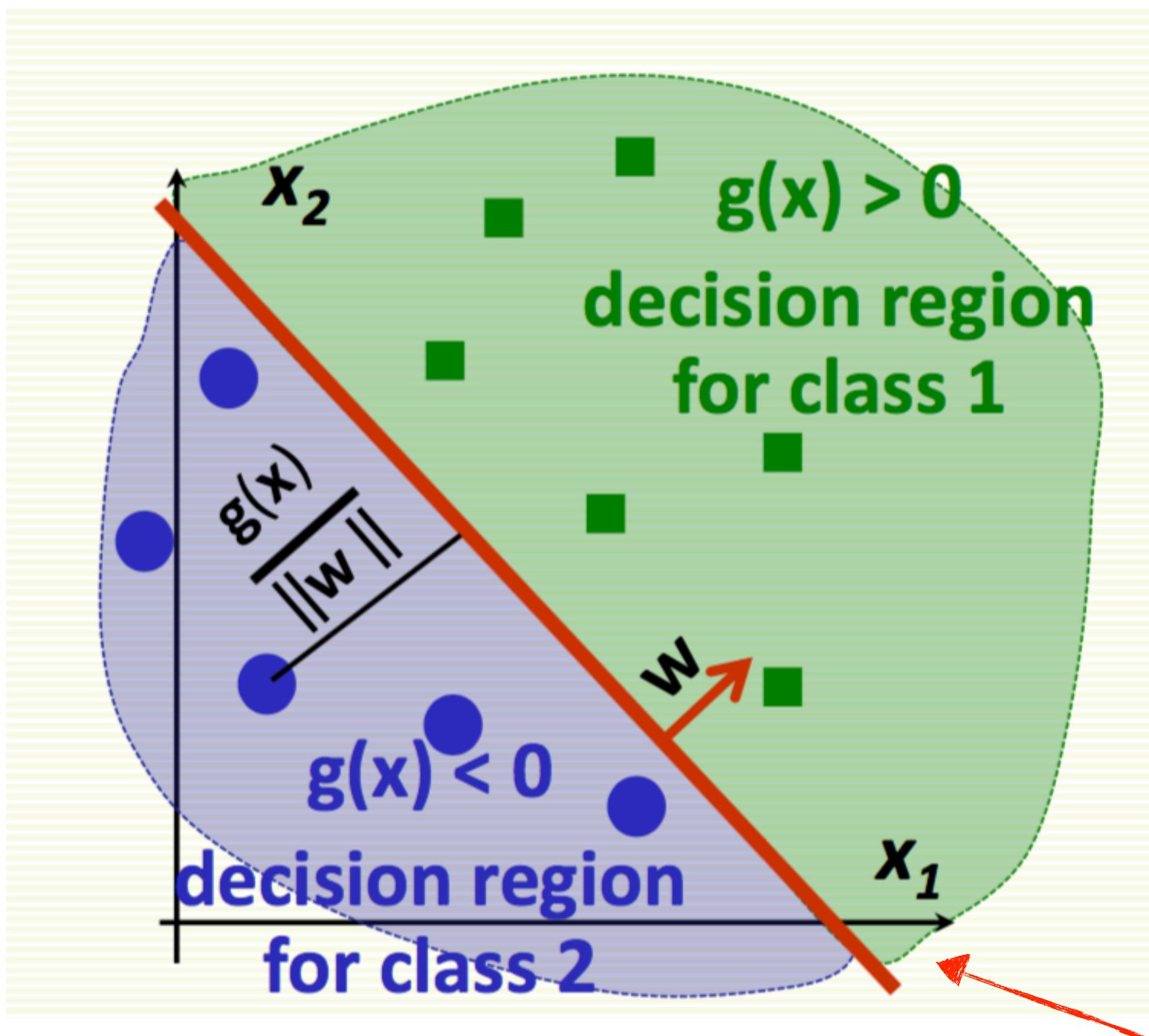
$$g(\mathbf{z}, \mathbf{w}) = \mathbf{w}^t \mathbf{z}$$

$$\begin{aligned}\mathbf{z} &= (1, x_1, x_2) \\ \mathbf{w} &= (w_0, w_1, w_2)\end{aligned}$$



$g(\mathbf{x}) > 0 \rightarrow$ classe ω_1
 $g(\mathbf{x}) < 0 \rightarrow$ classe ω_2

FDL: INTERPRETAZIONE GEOMETRICA



$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

$$g(\mathbf{x}) = 0 = w_0 + w_1 x_1 + w_2 x_2$$

w determina la direzione della retta di decisione
w₀ determina la posizione della retta di decisione

NOTA:
iperpiano ≡
punto in 1D
retta in 2D
piano in 3D
...

iperpiano (retta) di decisione: $g(\mathbf{x}) = 0$

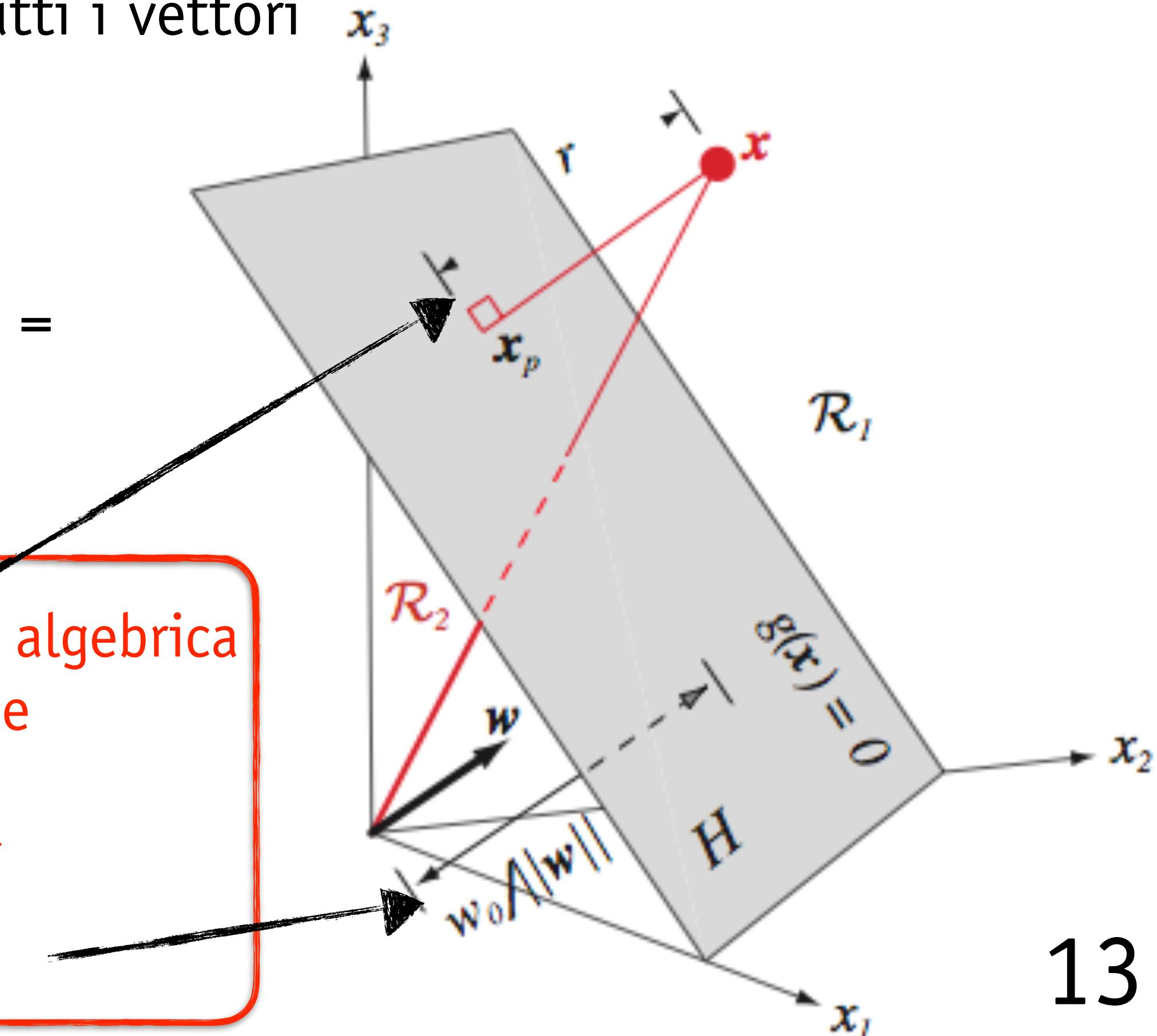
l'iperpiano di decisione rappresenta la migliore separazione tra due classi con confine lineare

FDL COME METRICA

- $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = 0 \Rightarrow$ geometricamente definisce un iperpiano nello spazio vettoriale delle feature chiamata **superficie di decisione** o **iperpiano di separazione**
- se prendiamo due vettori \mathbf{x}_1 e \mathbf{x}_2 che appartengono all'iperpiano di decisione $g(\mathbf{x}_1)=g(\mathbf{x}_2)=0 \Rightarrow$
- $\mathbf{w}^t \mathbf{x}_1 + w_0 = \mathbf{w}^t \mathbf{x}_2 + w_0 \Rightarrow \mathbf{w}^t (\mathbf{x}_1 - \mathbf{x}_2) = 0 \Rightarrow \mathbf{w}$ è ortogonale a tutti i vettori dell'iperpiano (cioè ne determina l'orientazione)
- un qualsiasi vettore \mathbf{x} può quindi essere espresso come:

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \Rightarrow g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = \mathbf{w}^t \mathbf{x}_p + w_0 + r \frac{\mathbf{w}^t \mathbf{w}}{\|\mathbf{w}\|} = \\ = g(\mathbf{x}_p) + r \|\mathbf{w}\| = r \|\mathbf{w}\| \Rightarrow$$

- $r = g(\mathbf{x})/\|\mathbf{w}\|$: la funzione discriminante fornisce una misura algebrica (una metrica) della distanza di \mathbf{x} dall'iperpiano di separazione
- se \mathbf{x} giace sull'iperpiano $\rightarrow r=0 \rightarrow \mathbf{w}^t \mathbf{x}/\|\mathbf{w}\| = -w_0/\|\mathbf{w}\| \rightarrow$
 $\rightarrow w_0$ definisce la posizione nello spazio dell'iperpiano



VANTAGGI DELLE FDL

- semplici da trattare analiticamente
- hanno basso costo computazionale
- possono risultare ottimali se le distribuzioni sottostanti sono simili (pdf normali con eguale covarianza) e per problemi linearmente separabili
- sulla base di particolari funzioni discriminanti lineari si costruiscono classificatori di architettura più complessa (non lineari) e più efficienti (SVM, NN, ...)



LOSS FUNCTION

- la loss function $L(\mathbf{w})$ e il metodo di ottimizzazione definiscono **l'algoritmo di apprendimento** utilizzato per ottenere il vettore dei pesi soluzione che definisce il modello (la funzione discriminante)
- **in pratica:**
 - L viene scelta in modo tale da restituire piccoli valori quando \mathbf{w} è buono, e grandi valori quando \mathbf{w} è cattivo
 - si minimizza L con un opportuno metodo di ottimizzazione (tipicamente una delle variazioni del SGD)
- La costruzione e il calcolo di L avviene in due step:
 - calcolo di una loss per evento: $L(g(\mathbf{x}_i, \mathbf{w}), t_i)$ per cui eventi classificati erroneamente ($g(\mathbf{x}_i, \mathbf{w}) \cdot t_i < 0$) sono penalizzati
 - calcolo total loss: $L(\mathbf{w})$ sommando le loss per evento sull'intero training set T :

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N L(g(\mathbf{x}_i, \mathbf{w}), t_i)$$

NOTA: dataset $T=\{\mathbf{x}_i, t_i\}$
con N eventi con label
 $t_i = \{-1, +1\} = \{\omega_2, \omega_1\}$



LOSS FUNCTION E ALGORITMI DI APPRENDIMENTO

- Esempio:

$$L(g(\mathbf{x}_i, \mathbf{w}), t_i) = \begin{cases} 0 & \text{se } g(\mathbf{x}_i, \mathbf{w})t_i > 0 \\ 1 & \text{altrimenti} \end{cases}$$

$$L(\mathbf{w}) = \frac{1}{2} \sum L(g(\mathbf{x}_i, \mathbf{w}), t_i)$$

conta il numero di errori fatti nella classificazione del training sample, cioè massimizza l'accuracy

- problema: è una funzione di \mathbf{w} con discontinuità ogni volta in cui un cambio in w porta il decision boundary a muoversi attraverso uno dei vettori x del campione. Il gradiente risulta nullo quasi da per tutto e non può essere minimizzata con il metodo della discesa lungo il gradiente
- si ottimizza un limite superiore (una approssimazione) dell'accuracy che risulti differenziabile e per il quale si possa applicare la tecnica di discesa lungo il gradiente
- diverse approssimazioni all'accuracy loss danno origine a diversi algoritmi di apprendimento::
 - algoritmo del Perceptron
 - Fisher
 - Minimum Squared Error (MSE)
 - Logistic Regression
 - ...
 - algoritmo a vettori di supporto (SVM)



ALGORITMO DEL PERCEPTRON

- per una data configurazione di pesi la loss perceptron è definita come:

$$L_{perc} = \sum_{j \in T_m} -\mathbf{w}^t \mathbf{z}_j t_j \quad \text{se } \mathbf{z}_j \text{ è mal classificato allora } -\mathbf{w}^t \mathbf{z}_j t_j > 0$$

- in cui T_m è il sottoinsieme di T in cui gli eventi sono classificati erroneamente: $T_m = \{j \mid \mathbf{w}^t \mathbf{z}_j t_j < 0\}$ per cui $L_{perc}(\mathbf{w}) \geq 0$
- interpretazione geometrica: il criterio del perceptron rappresenta la somma delle distanze algebriche tra l'iperpiano definito da \mathbf{w} e i campioni di T classificati in modo erroneo
- il contributo alla L associato ad un evento mal classificato è una funzione lineare di \mathbf{w} nelle regioni dello spazio w in cui x è mal classificato e zero nelle regioni in cui è classificato correttamente. La L totale è quindi lineare a tratti e può essere minimizzata con tecniche SGD
- l'algoritmo è iterativo in cui SGD è usato per modificare il vettore dei pesi: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla L_{perc}(\mathbf{w}) = \mathbf{w}^{(k)} + \eta z t$
 - il processo si ferma quando viene raggiunta una data condizione di stop:
 - $L(\mathbf{w}(k)) \leq L_{min}$
 - numero di iterazioni $\geq max_iter$
 - classi linearmente separabili: è garantita la convergenza ad una soluzione
 - classi non linearmente separabili: l'algoritmo non converge e la condizione di termine è obbligatoria



ALGORITMO DEL PERCEPTRON

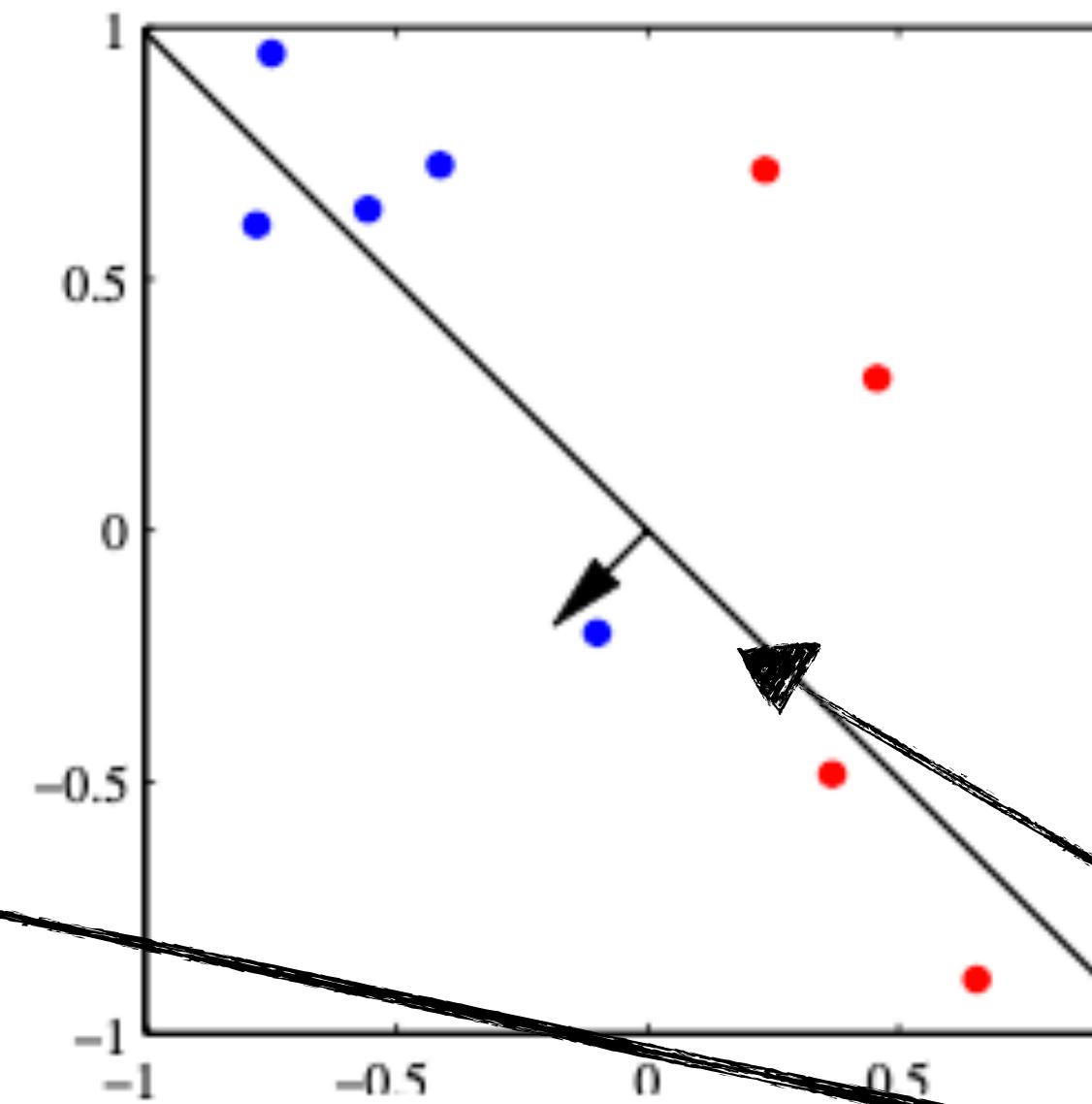
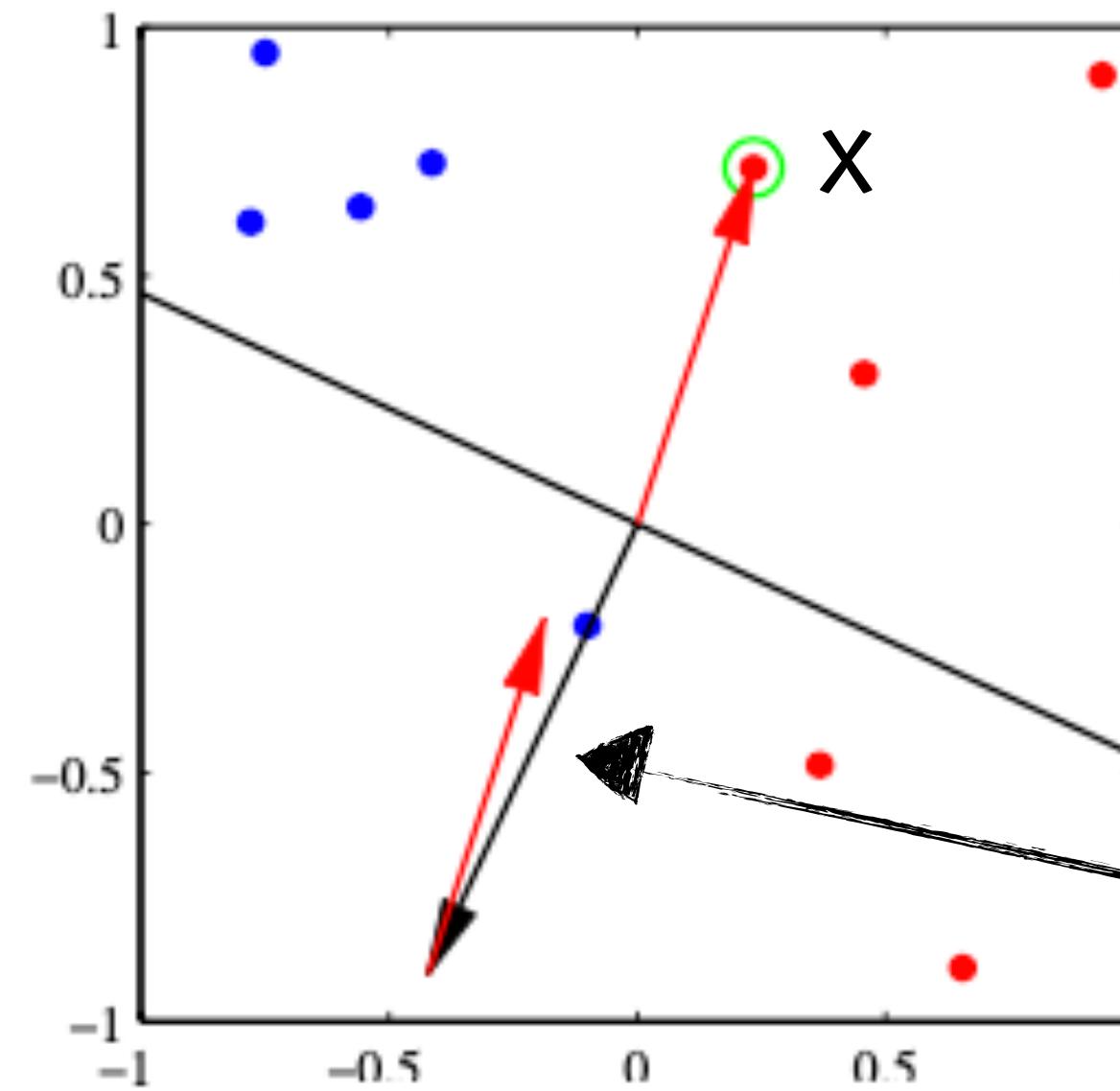
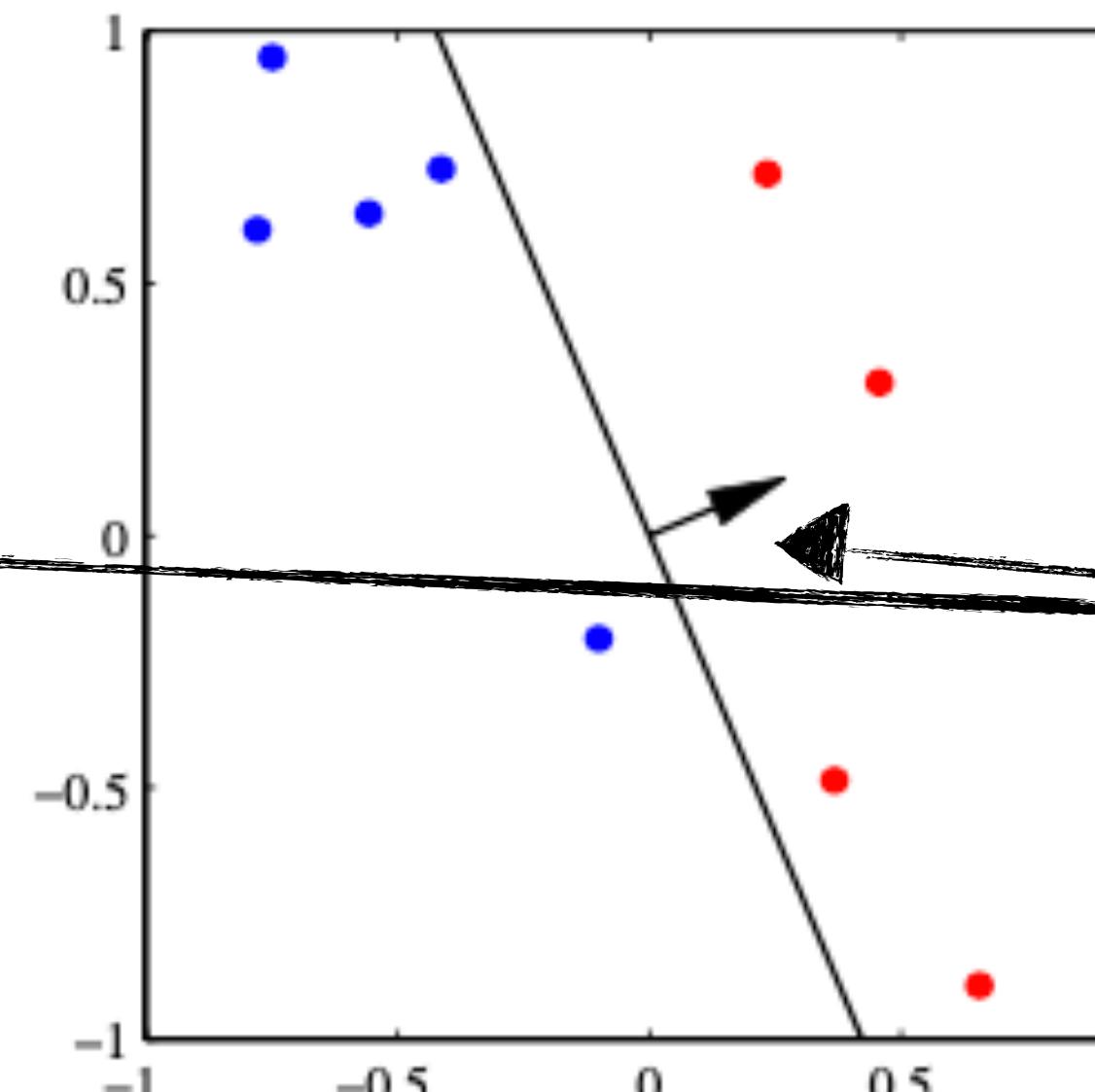
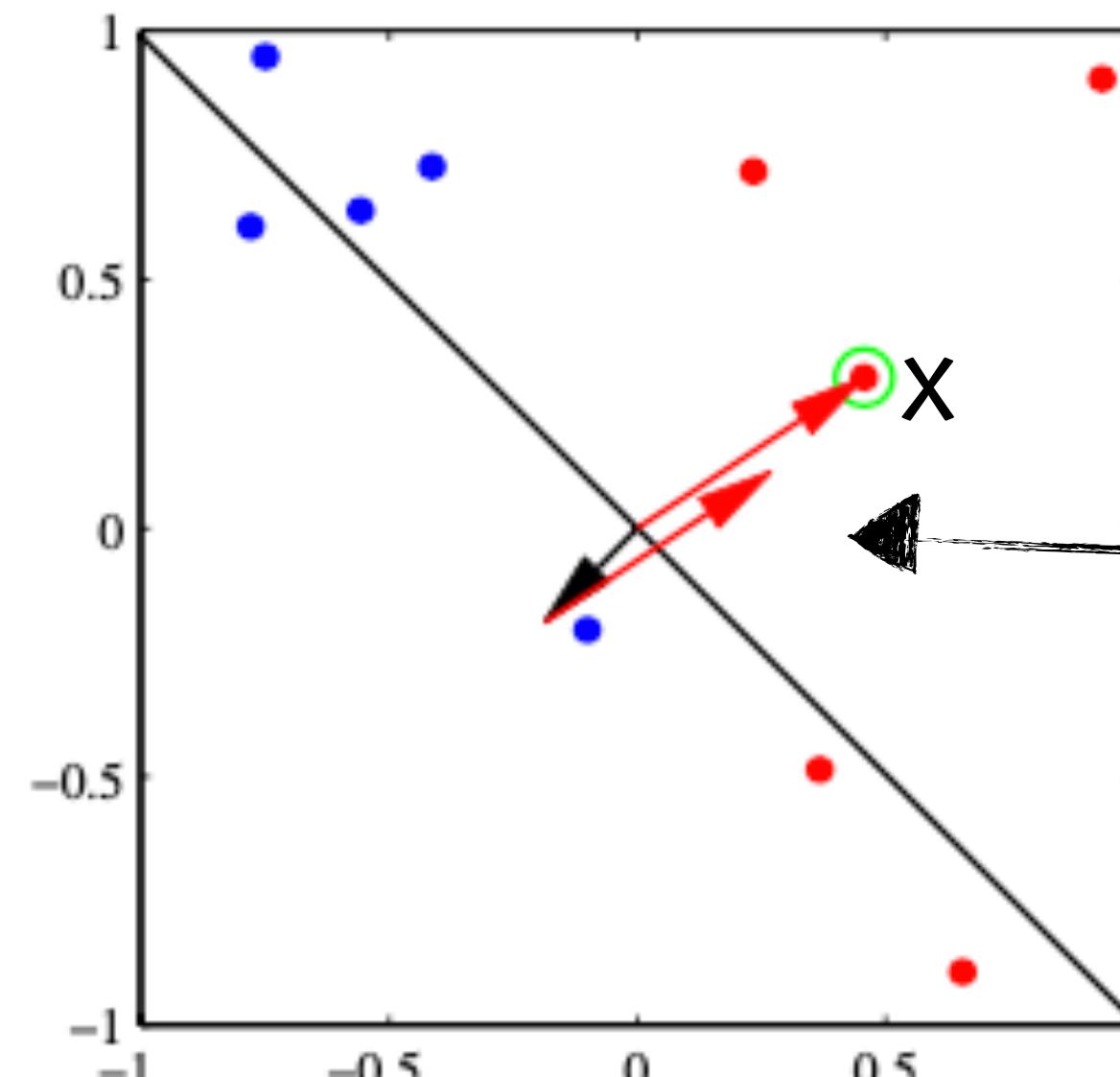


Illustrazione di come l'algoritmo del perceptron converge alla soluzione

2 classi (rossi e blu)
spazio feature 2D

la freccia nera indica la direzione in cui il classificatore classifica classe rossi



$$w^{(1)} = w^{(0)} + \eta xt$$

$$w^{(2)} = w^{(1)} + \eta xt$$

ALGORITMO MSE

- l'algoritmo del perceptron è di fatto una **procedura per la correzione degli errori**, genera infatti una modifica del vettore dei pesi solo quando incontra un errore (evento mal classificato)
- l'algoritmo **MSE (Minimum Squared Error)** invece tiene conto di tutti i campioni del training set
- L'idea è quella di sostituire la ricerca di un w tale che $\mathbf{w}^t \mathbf{z} > 0$ su tutti gli eventi mal misurati del training set, la ricerca di un w tale che: $\mathbf{w}^t \mathbf{z} = b$ con b costante arbitraria positiva
- e trasformare il tutto nel problema, meglio trattabile, di trovare la soluzione di un insieme di equazioni lineari (che rappresentano l'iperpiano di separazione) del tipo:

$z^{(i)}$ evento i -esimo del training set $\xrightarrow{\hspace{1cm}}$

$$\begin{bmatrix} z_0^{(1)} & z_1^{(1)} & \dots & z_d^{(1)} \\ z_0^{(2)} & z_1^{(2)} & \dots & z_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ z_0^{(N)} & z_1^{(N)} & \dots & z_d^{(N)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_d \end{bmatrix} \Rightarrow \mathbf{Zw} = \mathbf{b}$$



ALGORITMO MSE / WIDROW-HOFF

- il numero delle equazioni (campioni) è di fatto sempre maggiore delle incognite (numero dei pesi), per cui non è possibile usare la consueta regola di soluzione
- si cerca allora di determinare un \mathbf{w} che minimizzi una funzione dell'errore tra l'uscita della FDL (\mathbf{Zw}) e l'uscita attesa (target) (\mathbf{y})
- Esempio. l'errore quadratico:

$$L_{MSE}(\mathbf{w}) = \| \mathbf{Zw} - \mathbf{b} \|^2 = \frac{1}{2} \text{Tr}[(\mathbf{Zw} - \mathbf{b})^t (\mathbf{Zw} - \mathbf{b})]$$

- con soluzione analitica ben nota:

$$\nabla L_{MSE}(\mathbf{w}) = 0 \rightarrow \mathbf{Z}^t(\mathbf{Zw} - \mathbf{b}) = 0 \rightarrow \mathbf{Z}^t \mathbf{Zw} = \mathbf{Z}^t \mathbf{b} \rightarrow \mathbf{w} = (\mathbf{Z}^t \mathbf{Z})^{-1} \mathbf{Z}^t \mathbf{b} = \mathbf{Z}^\dagger \mathbf{b}$$

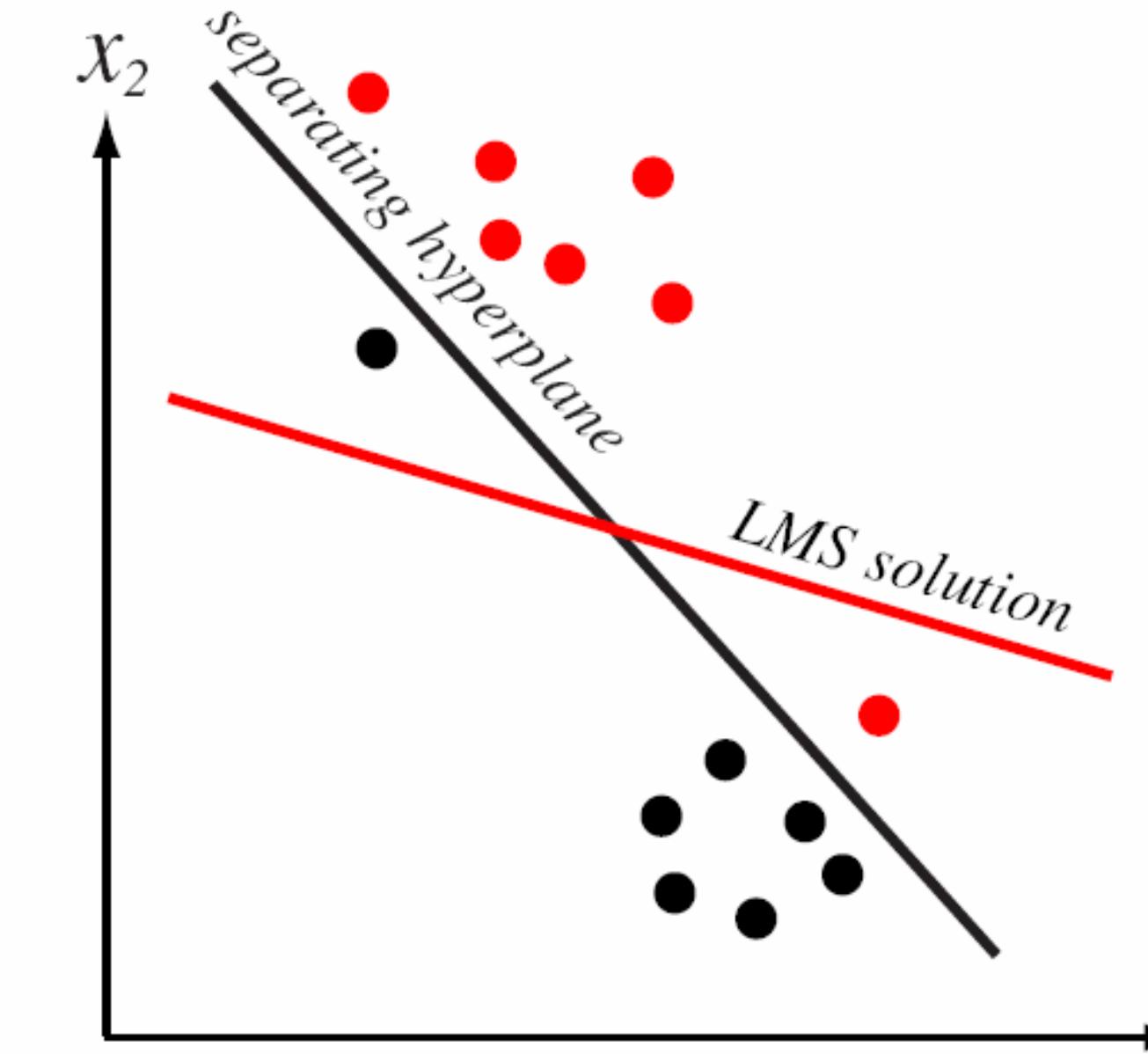
con \mathbf{Z}^\dagger pseudoinversa di \mathbf{Z} (= \mathbf{Z}^{-1} se quadrata e non singolare)

- il minimo del criterio $L_{MSE}(\mathbf{w})$ può anche essere trovato tramite un algoritmo di discesa lungo il gradiente che risulta più efficiente nel caso di campioni di grande statistica (metodo Widrow-Hoff (o LMS))



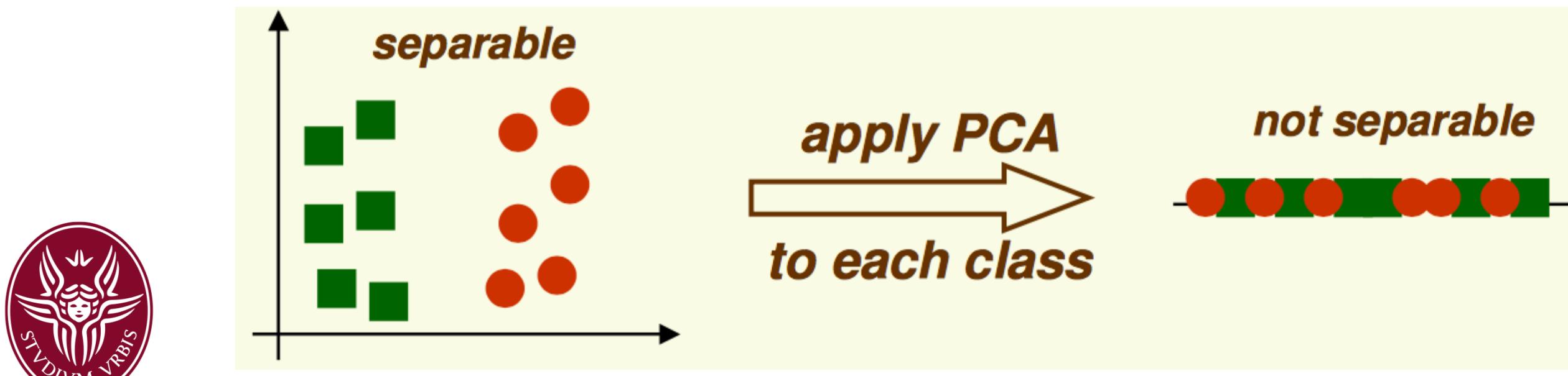
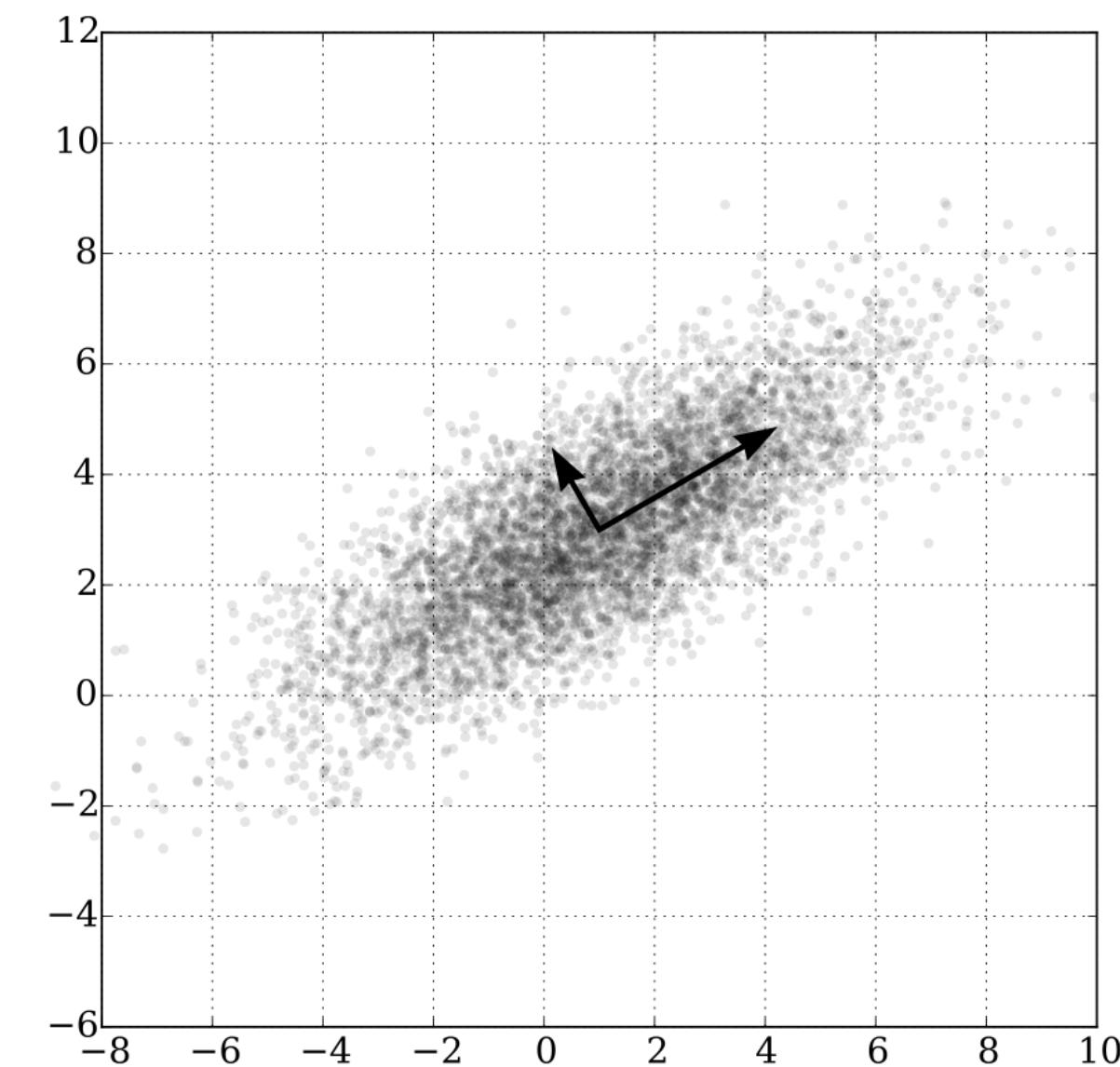
CONFRONTO TRA I 2 METODI

- **Algoritmo del Perceptron:**
 - trova sempre una soluzione se le classi sono linearmente separabili
 - non converge se non lo sono
- **Algoritmi MSE**
 - sono sicuramente convergenti, ma possono NON trovare l'iperpiano di separazione tra le classi anche se queste sono linearmente separabili (a causa dell'arbitrarietà di \mathbf{b})
 - di fatto gli algoritmi MSE cercano l'iperpiano che minimizza la somma dei quadrati delle distanze esistenti tra i campioni di training e l'iperpiano stesso e non cercano il vero l'iperpiano separatore
 - Se si sceglie \mathbf{b} tale da avere lo stesso valore per tutti i campioni della stessa classe si dimostra che la soluzione MSE per w corrisponde a quella del **discriminante lineare di Fisher**



ANALISI A COMPONENTI PRINCIPALI E FDL DI FISHER

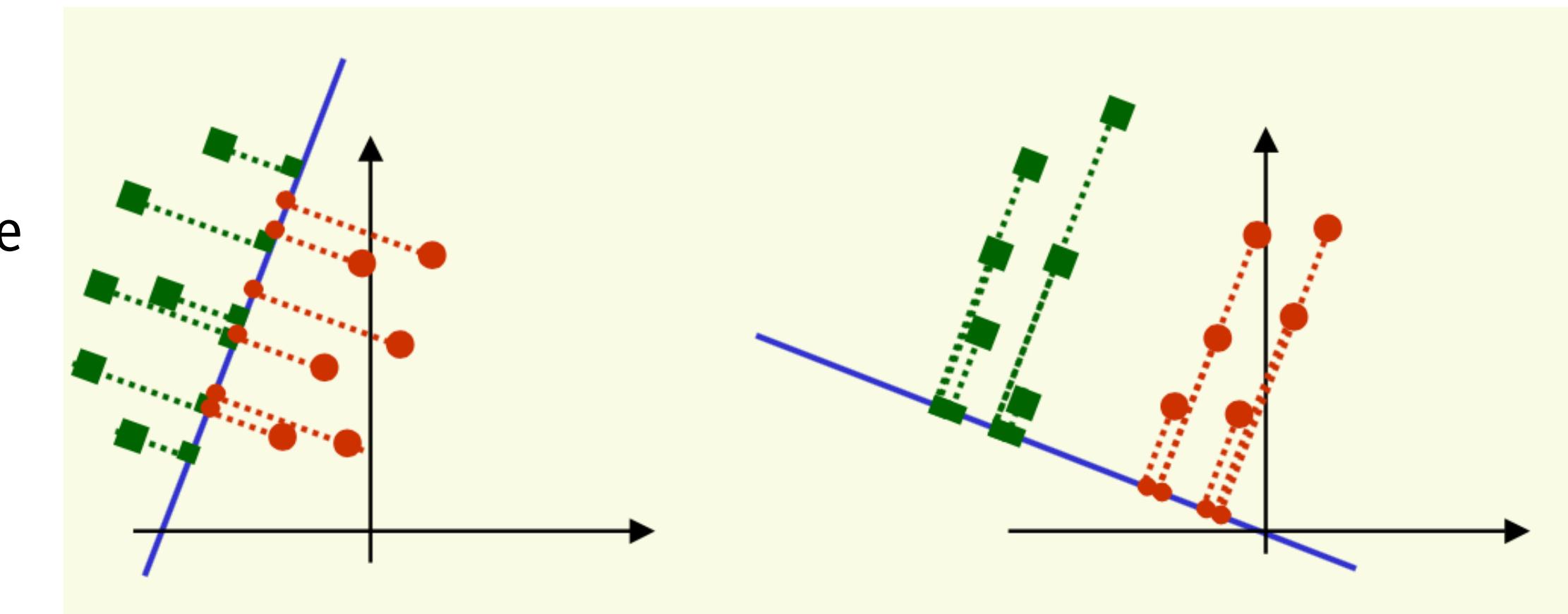
- l'analisi a componenti principali (PCA) è un noto metodo di riduzione dimensionale per determinare la rappresentazione più accurata di più campioni D-dimensionalni in uno spazio a dimensione ridotta d<D
 - l'idea è quella di combinare le feature $x_1 \dots x_D$ in un numero inferiore di **feature latenti limitando il più possibile la perdita di informazioni**
 - ad esempio potremmo scegliere m tale che la somma delle distanze tra m e le varie x_k sia la più piccola possibile:
 - $m = \langle x \rangle = 1/N \sum x_i \rightarrow$ rappresentazione 0-dimensionale (la media è sicuramente un parametro utile ma chiaramente si sta scartando troppa informazione)
 - per mantenere informazione anche sullo spread dei campioni si proiettano i dati su un nuovo sistema cartesiano centrato nel punto m e in cui gli assi rappresentano le direzioni di massima varianza, seconda massima varianza, etc... del campione e si sceglie di rappresentare il campione con le prime d di tali variabili
- La rappresentazione nello spazio latente PCA spesso risulta inutile o anche dannosa in un problema di classificazione:



Il discriminante lineare di Fisher rappresenta una estensione della tecnica PCA in cui le direzioni di proiezione vengono scelte in modo tale da massimizzare il potere di classificazione tra le due classi

FDL DI FISHER

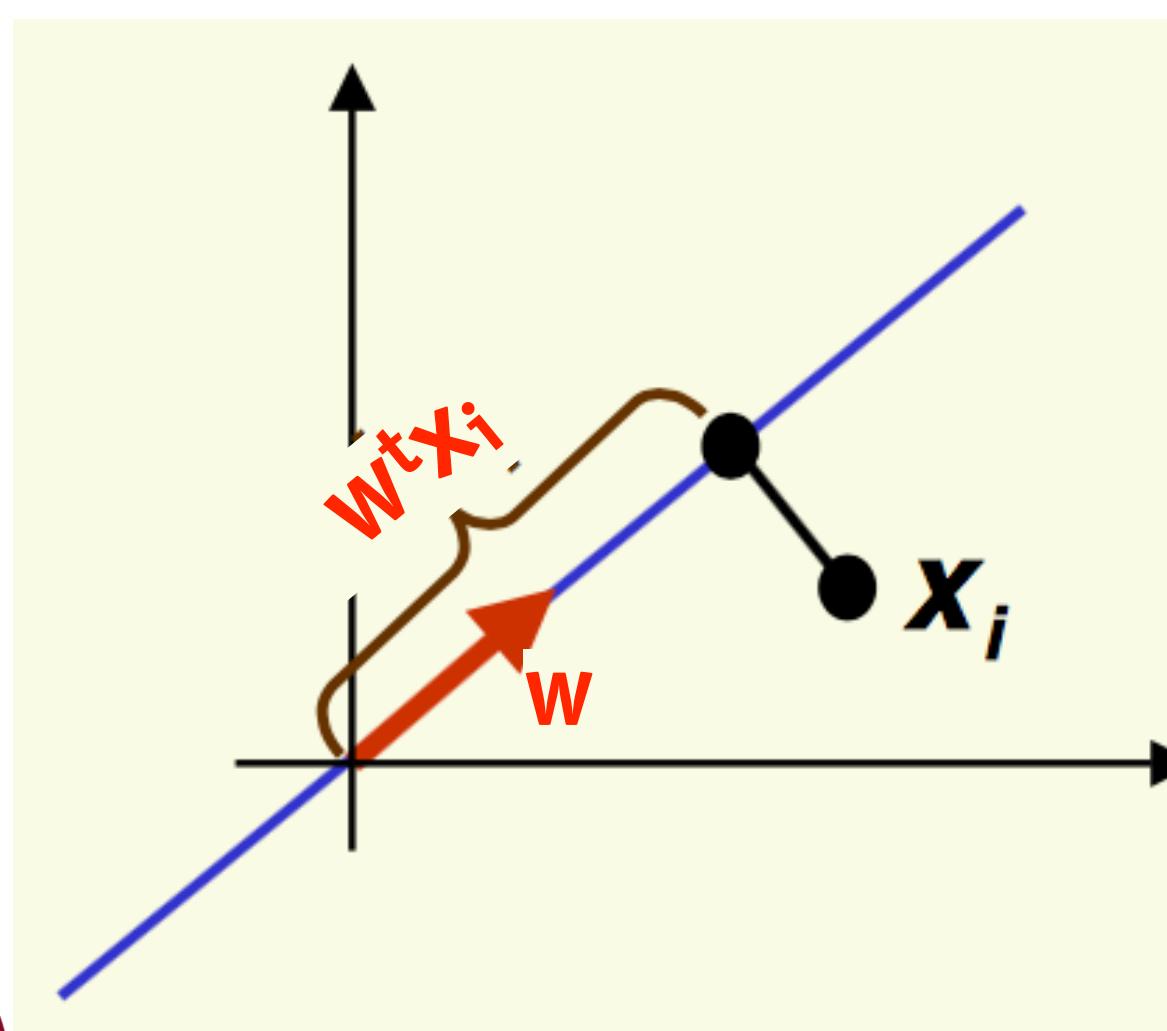
- idea principale: trovare la direzione w per cui i campioni proiettati su tale direzione sono ben separati



mal separati

ben separati

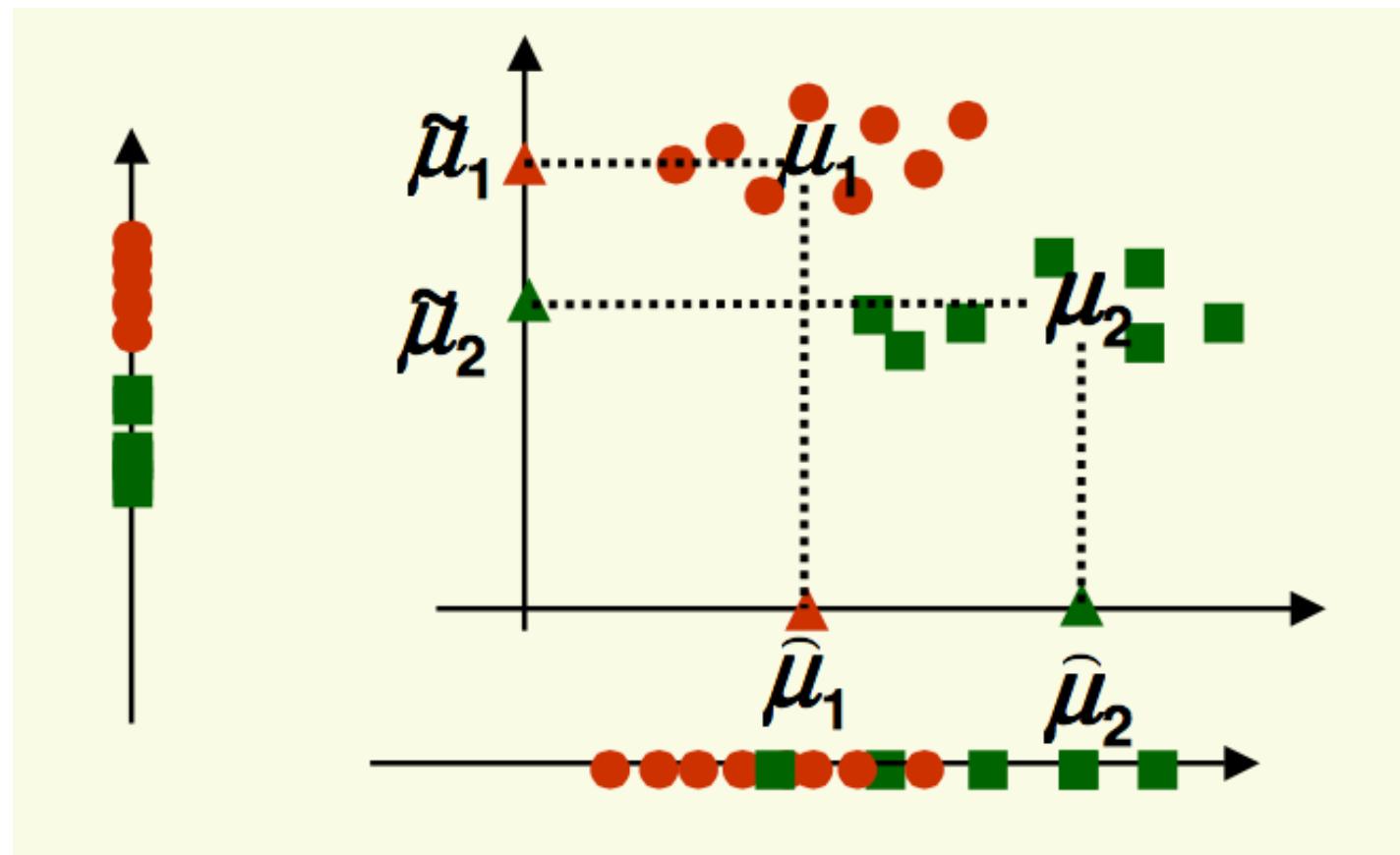
- supponiamo di avere 2 classi d-dimensionalili e $n=n_1$ (classe 1) + n_2 (classe 2) eventi $x_1 \dots x_n$
- consideriamo la proiezione sulla retta definita dal versore w



- il prodotto (scalare) $y_i = w^T x_i$ rappresenta la proiezione di x_i dall'origine
- $w^T x_i$ rappresenta quindi la proiezione di x_i in un sotto-spazio a dimensione ridotta ($d-1$)
- siano m_1 e m_2 i valori medi delle distribuzioni delle due classi nello spazio originale → i valori proiettati saranno:
 - $\mu_i = 1/n_i \sum y_i = 1/n_i \sum w^T x_i = w^T (1/n_i \sum x_i) = w^T m_i$
 - μ_i è quindi semplicemente dato dalla proiezione di m_i
 - possiamo usare $|\mu_1 - \mu_2|$ come misura della separazione tra i campioni per trovare la direzione w

FDL DI FISHER

- quanto è buono $|\mu_1 - \mu_2|$ come criterio per la stima della separazione?

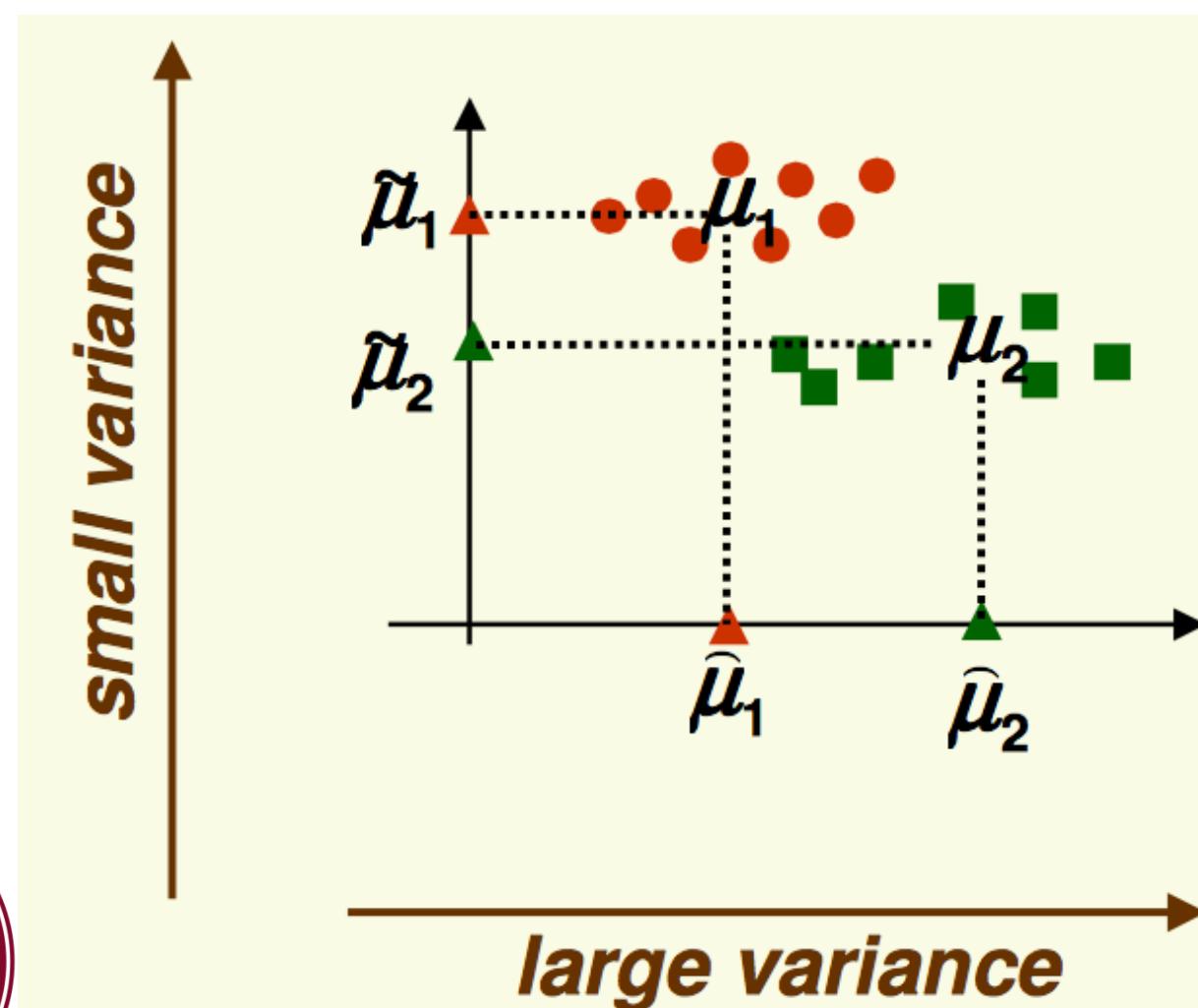


più grande è $|\mu_1 - \mu_2|$ maggiore è la separazione → OK

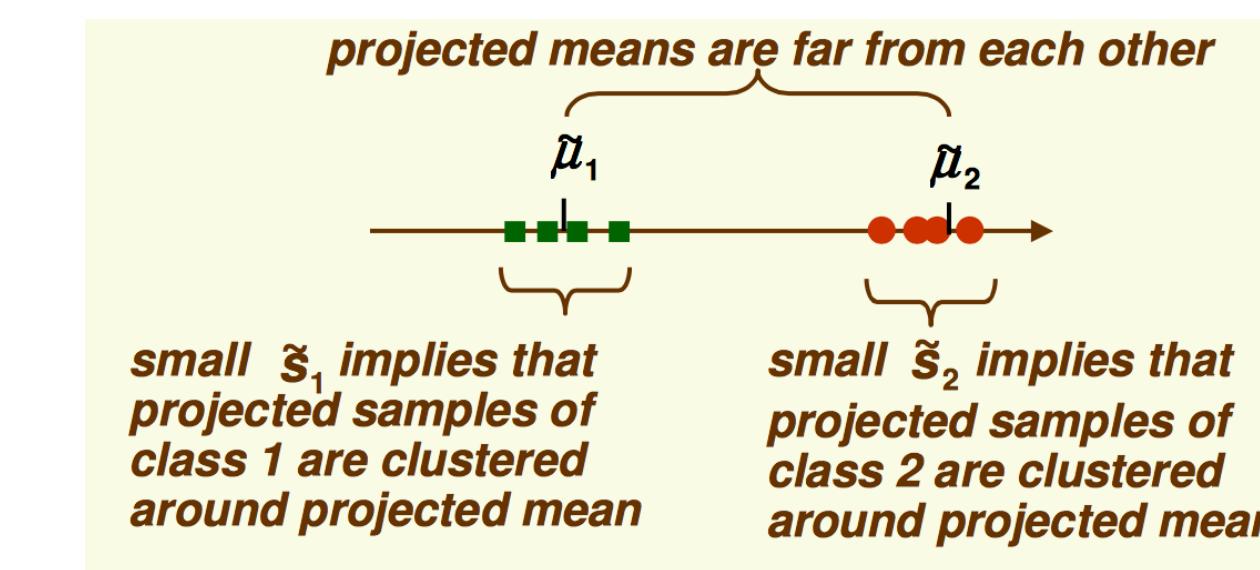
tuttavia nell'esempio:

- l'asse verticale separa meglio di quello orizzontale
- ma $|\tilde{\mu}_1 - \tilde{\mu}_2| < |\hat{\mu}_1 - \hat{\mu}_2|$

- il problema stà nel fatto che $|\mu_1 - \mu_2|$ non tiene in conto la varianza delle due classi nelle due direzioni



- si risolve il problema normalizzando $|\mu_1 - \mu_2|$ a qualche cosa che sia proporzionale alle varianze di ciascuna classe



- $s_i^2 = \sum (x_i - m_i)^2 \rightarrow$ nello spazio trasformato: $\sigma_i^2 = \sum (y_i - \mu_i)^2$
- Definiamo la varianza per classe totale del campione come: $\sigma^2 = \sigma_1^2 + \sigma_2^2$

FDL DI FISHER

- Criterio di Fisher:

- nella procedura di ottimizzazione viene massimizzata la distanza tra le medie delle due classi e minimizzata contemporaneamente la varianza all'interno di ciascuna classe:

$$J(\mathbf{w}) = \frac{|\mu_1 - \mu_2|^2}{\sigma^2} = \frac{|\mathbf{w}^t(\mathbf{m}_1 - \mathbf{m}_2)|^2}{\mathbf{w}^t \mathbf{S}_w \mathbf{w}}$$

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{x \in T_i} \mathbf{x}$$
$$\mathbf{S}_w = \sum_{i=1}^2 \sum_{x \in T_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$$

\mathbf{S}_w somma delle matrici di covarianza in ciascuna classe

- \mathbf{w} che massimizza J garantisce prestazioni ottimali in separazione tra le due classi per variabili gaussiane con correlazioni lineari
- NOTA: se una variabile ha stesso valore medio nelle due classi \rightarrow nessuna separazione \rightarrow spesso conviene usare $|\text{var}|$

Calcolando il gradiente di J rispetto a \mathbf{w} per trovare il massimo si ottiene:

$$\mathbf{w} = \frac{(\mathbf{m}_1 - \mathbf{m}_2)}{\mathbf{S}_w}$$

FDL di Fisher



FISHER E CRITERIO MSE

- Si dimostra che Fisher è un caso speciale del criterio MSE (Duda et Al.): 2 classi, n_1 eventi T_1 e n_2 eventi T_2 $n=n_1+n_2$, si definiscono:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \frac{n}{n_1} \mathbf{1}_1 \\ \frac{n}{n_2} \mathbf{1}_2 \end{bmatrix} \quad \mathbf{m}_i = \frac{1}{n_i} \sum_{x \in T_i} \mathbf{x} \quad \mathbf{S}_w = \sum_{i=1}^2 \sum_{x \in T_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$$

$$\mathbf{Z}\mathbf{w} = \mathbf{b} \rightarrow \mathbf{Z}^t \mathbf{Z}\mathbf{w} = \mathbf{Z}^t \mathbf{b} \quad \rightarrow \quad \begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \frac{n}{n_1} \mathbf{1}_1 \\ \frac{n}{n_2} \mathbf{1}_2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} n & (n_1 \mathbf{m}_1 + n_2 \mathbf{m}_2)^t \\ (n_1 \mathbf{m}_1 + n_2 \mathbf{m}_2) & \mathbf{S}_w + n_1 \mathbf{m}_1 \mathbf{m}_1^t + n_2 \mathbf{m}_2 \mathbf{m}_2^t \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} 0 \\ n(\mathbf{m}_1 - \mathbf{m}_2) \end{bmatrix} \quad \text{sono due equazioni:}$$

prima riga: (\mathbf{m} = media di tutti i campioni) $\rightarrow w_0 = -\mathbf{m}^t \mathbf{w}$ \rightarrow sostituendo nella seconda riga + algebra:

$$\left[\frac{1}{n} \mathbf{S}_w + \frac{n_1 n_2}{n^2} (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \right] \mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)$$

osservando che $(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \mathbf{w}$ è diretto come $(\mathbf{m}_1 - \mathbf{m}_2) \forall \mathbf{w}$ \rightarrow

$$\frac{n_1 n_2}{n^2} (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \mathbf{w} = (1 - \alpha)(\mathbf{m}_1 - \mathbf{m}_2) \rightarrow \mathbf{w} = \alpha n \frac{(\mathbf{m}_1 - \mathbf{m}_2)}{\mathbf{S}_w}$$

con α opportuno scalare

che a meno di fattori di scala è la soluzione della FDL di fisher



ESEMPI CLASIFICATORI LINEARI IN SCIKIT-LEARN

https://www.dropbox.com/s/hy79h33vvhd7tkd/Perceptron_and_FDL_classification.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import Perceptron
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

```
# IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, :2] # take only the first two features
y = iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=.5, random_state=1111, shuffle=True)
```

```
usePerc = False
```

```
if usePerc == True:
```

```
    # preceptron
    # stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol)
    logreg = Perceptron(tol=1e-7, random_state=0)
```

```
else:
```

```
    #Fisher LDA
    logreg = LinearDiscriminantAnalysis()
```

IRIS dataset

perceptron

LDA (Fisher)

```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X_train, y_train)
```



ESEMPI CLASIFICATORI LINEARI IN SCIKIT-LEARN

perceptron

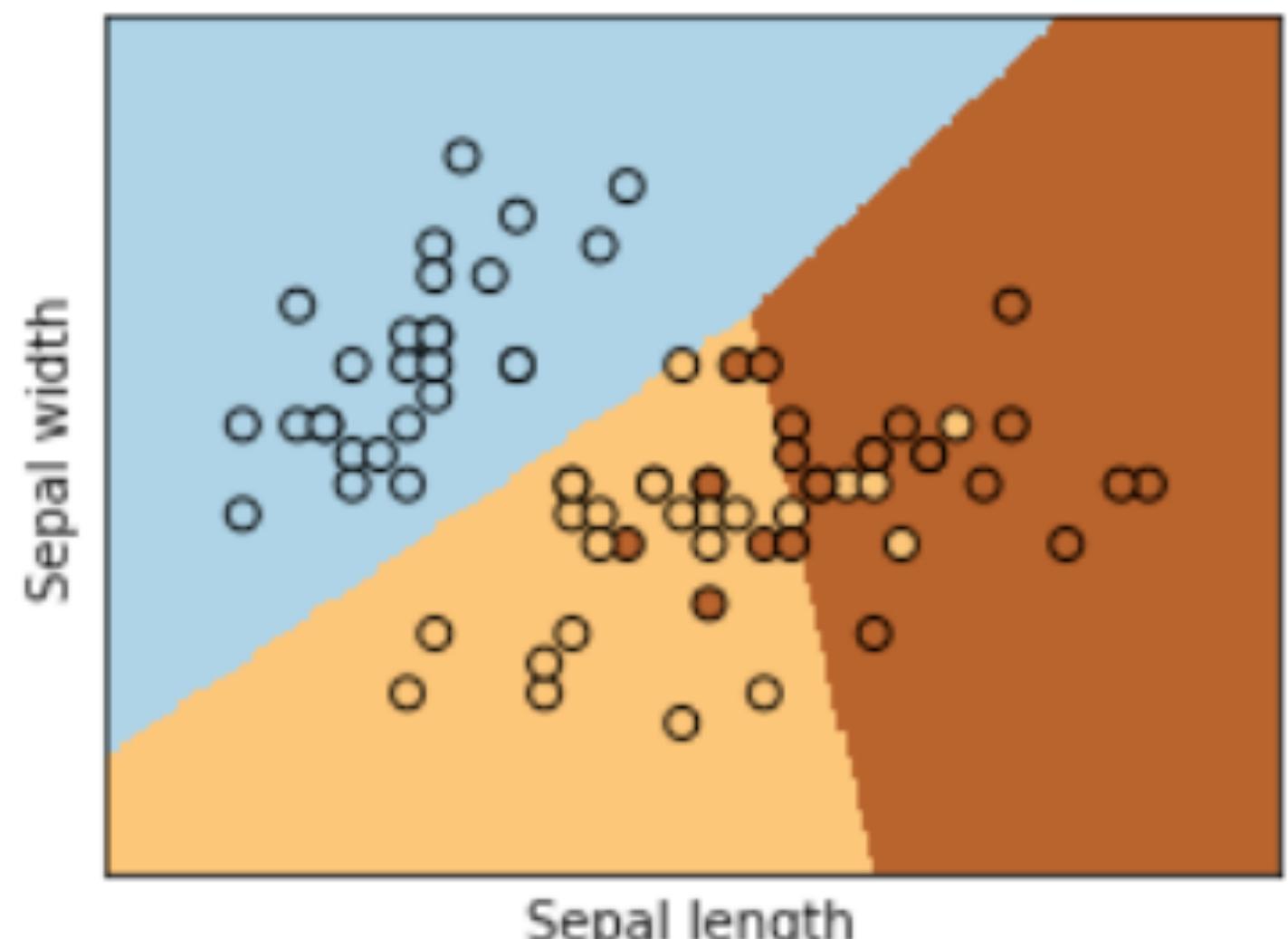
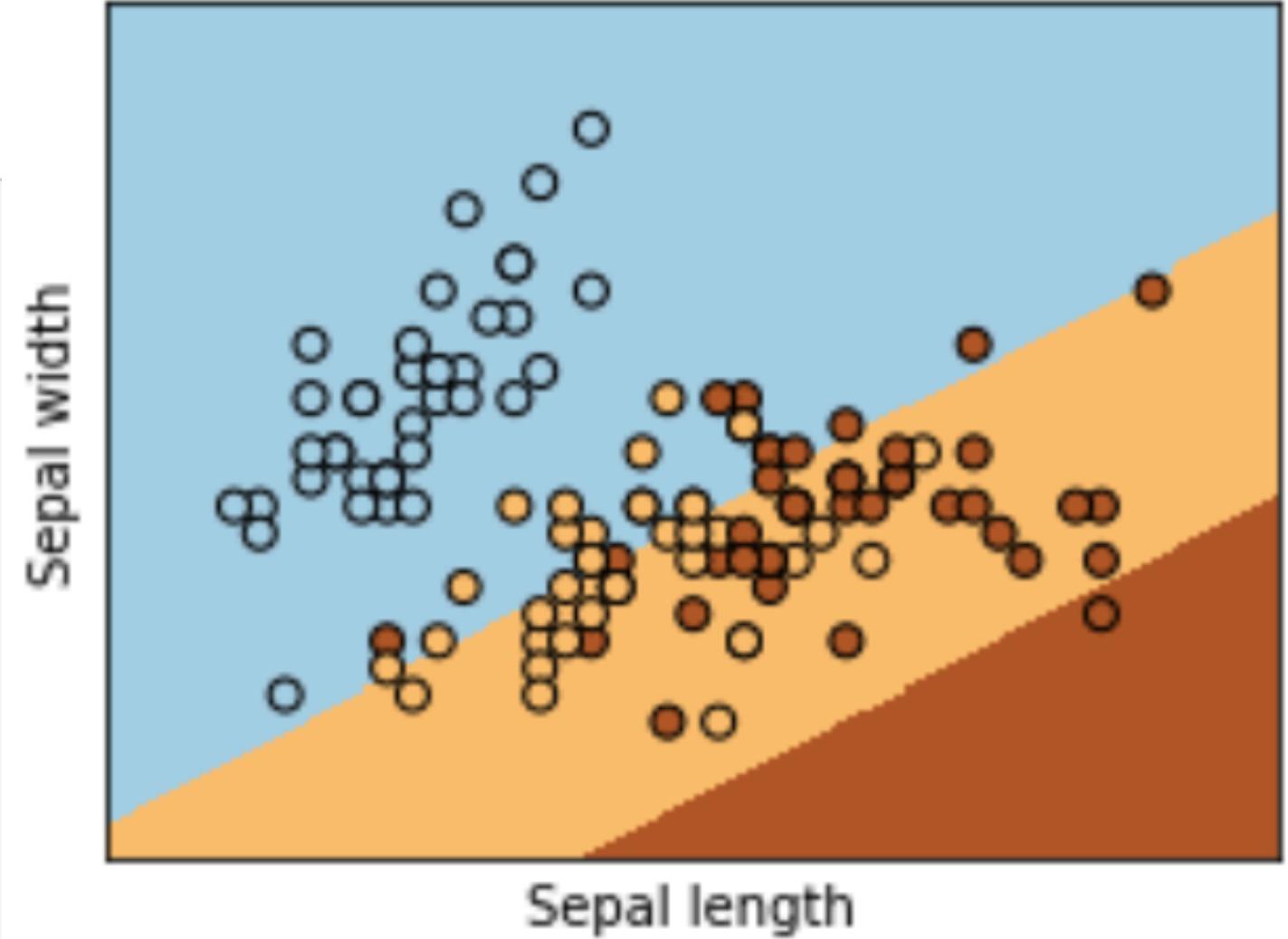
```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X_train[:, 0].min() - .5, X_train[:, 0].max() + .5
y_min, y_max = X_train[:, 1].min() - .5, X_train[:, 1].max() + .5
h = .02 # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



LDA

