

# Classificazione

- Classificatore di Bayes
  - Approccio parametrico (distribuzione Multinormale)
  - Approccio non parametrico (Parzen Window)
- Nearest Neighbor
  - $k$ -NN
  - Metriche
- SVM
  - Lineari: pattern linearmente separabili e non
  - Non lineari
  - Caso multiclasse
- Multi classificatori
  - Fusione a livello di decisione
  - Fusione a livello di confidenza
  - Random Forest (Bagging)
  - AdaBoost (Boosting), Gradient Boosting

# Support Vector Machines (SVM)

La **teoria** che governa i meccanismi di funzionamento di SVM è stata introdotta da **Vapnik** a partire dal 1965 (**statistical learning theory**), e perfezionata più recentemente (1995) dallo stesso Vapnik e altri. SVM è uno degli strumenti più **utilizzati** per la classificazione di pattern.

Invece di stimare le densità di probabilità delle classi, Vapnik suggerisce di risolvere direttamente il problema di interesse (che considera più semplice), ovvero **determinare le superfici decisionali tra le classi** (**classification boundaries**).

**andiamo per gradi ...**

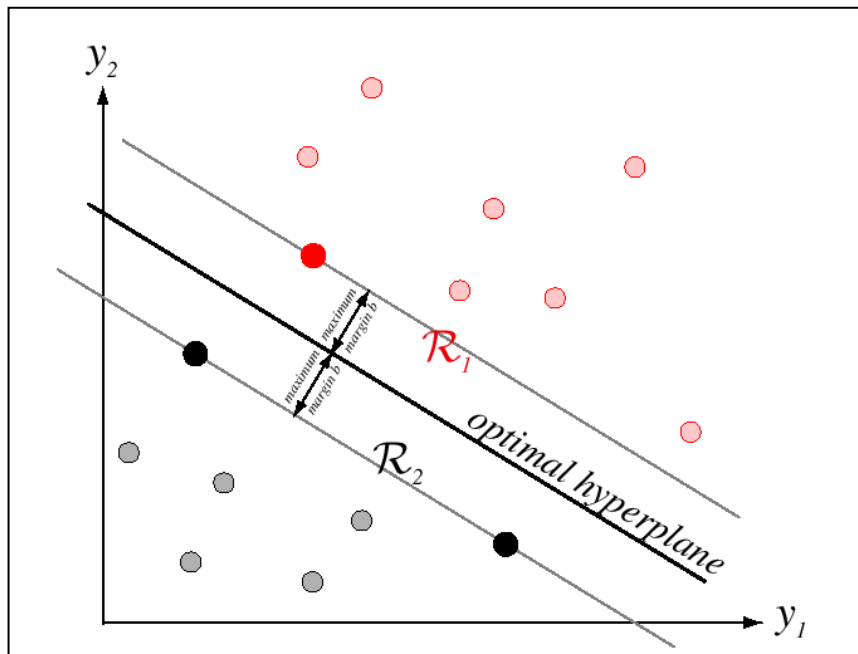
SVM nasce come **classificatore binario** (2 classi), estendibile a più classi. Affrontiamo la trattazione per gradi:

- SVM **lineare** (i.e., la superficie di separazione è un **iperpiano**) e pattern del training set **linearmente separabili** (i.e., esiste per ipotesi almeno un iperpiano in grado di separarli).
- SVM **lineare** e pattern **non linearmente separabili**. Ci saranno inevitabilmente errori di classificazione nel training set non esistendo alcun iperpiano in grado di separare i pattern.
- SVM **non lineare** (i.e., superficie di separazione **complessa**) senza ipotesi sulla separabilità dei pattern.
- Estensione **multiclasse**.

# SVM: l'idea

Date due classi di pattern multidimensionali linearmente separabili, tra tutti i possibili **iperpiani** di separazione, SVM determina quello in grado di **separare le classi** con il maggior **margin** possibile.

Il **margin** è la distanza minima di punti delle due classi nel training set dall'iperpiano individuato. *Definizione formale in seguito.*

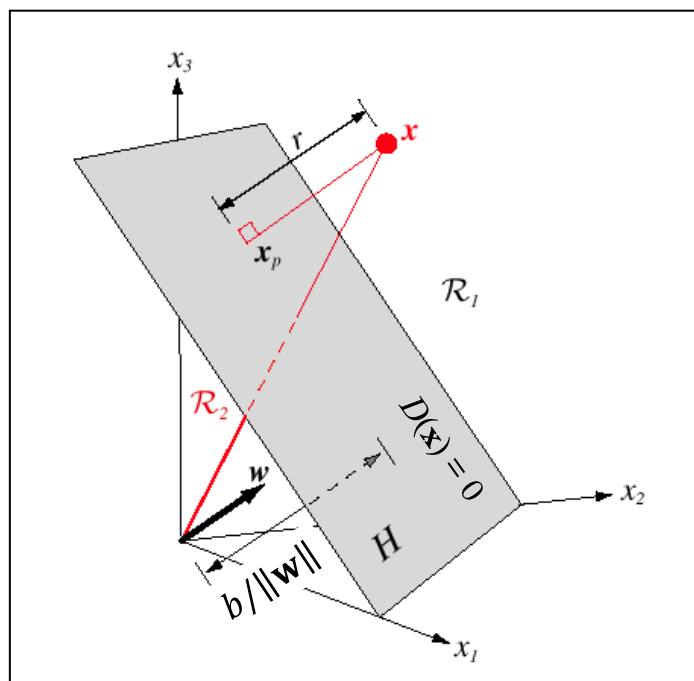


La massimizzazione del **margin** è legata alla **generalizzazione**. Se i pattern del training set sono classificati con ampio margine si può «sperare» che anche pattern del test set vicini al confine tra le classi siano gestiti correttamente.

Come vedremo il problema può essere impostato come una ottimizzazione convessa (convex optimization). Come tale ammette **un solo minimo globale** (vantaggio: non si rischia di essere intrappolati in minimi locali).

# SVM lineari: Pattern Separabili

Date due classi di pattern (**linearmente separabili**), e un training set **TS** contenente  $n$  campioni  $(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)$ , dove  $\mathbf{x}_i \in \mathbb{R}^d$  sono i **pattern multidimensionali** e  $y_i \in \{+1, -1\}$  le **etichette** delle due classi, esistono **diversi iperpiani** in grado di eseguire la separazione voluta. Un generico iperpiano è definito dai parametri  $(\mathbf{w}, b)$ :



**Iperpiano**

per semplicità  
omettiamo il trasposto  
nel prodotto scalare

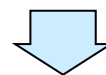
$$D(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

$\mathbf{w}$ : vettore normale all'iperpiano

$b/\|\mathbf{w}\|$ : distanza dall'origine

$D(\mathbf{x}) = 0$ : luogo dei vettori sul piano

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}, \quad D(\mathbf{x}_p) = 0$$



$$D(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = r \cdot \|\mathbf{w}\|$$

La **distanza** di un vettore  $\mathbf{x}$  dall'iperpiano vale pertanto:  $r = \frac{D(\mathbf{x})}{\|\mathbf{w}\|}$

Gli **iperpiani**  $(\mathbf{w}, b)$  che **separano i pattern** del TS, con distanza minima  $1/\|\mathbf{w}\|$  su ogni lato, soddisfano, per  $i = 1 \dots n$ , le equazioni:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 \quad \text{se } y_i = +1$$

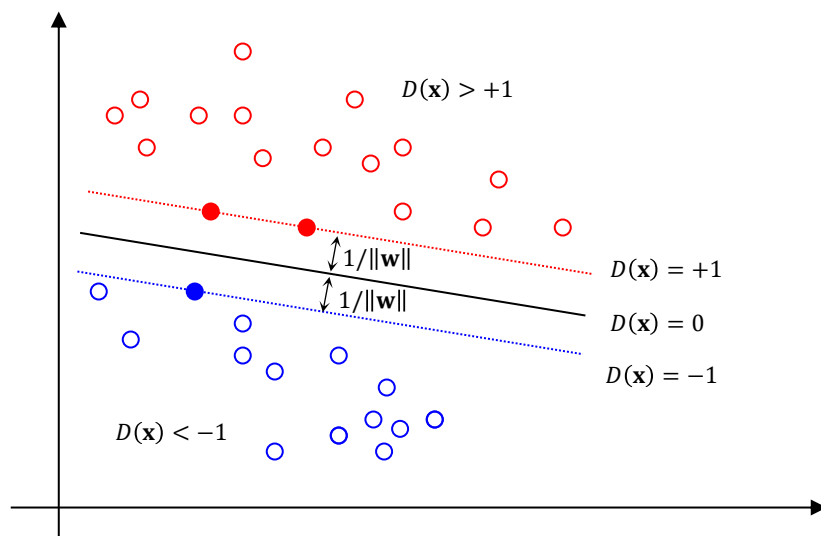
$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \quad \text{se } y_i = -1$$

o in modo più compatto:

$$y_i [\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 \quad \text{per } i = 1 \dots n$$

# SVM lineari: Pattern Separabili (2)

La **minima distanza** tra l'iperpiano di separazione e un pattern del training set è detta **margin** ( $\tau$ ).



La distanza dei punti che giacciono sull'iperpiano  $D(\mathbf{x}) = +1$  dall'iperpiano di separazione ( $D(\mathbf{x}) = 0$ ) è  $1/\|\mathbf{w}\|$ ; lo stesso vale per i punti sull'iperpiano  $D(\mathbf{x}) = -1$ .

Pertanto il **margin** è  $\tau = 2/\|\mathbf{w}\|$ .

L'iperpiano **ottimo** secondo SVM è quello soddisfa i vincoli di **separazione dei pattern** e **massimizza il margin**  $\tau$  (o alternativamente minimizza il suo inverso):

**Minimizza:**  $\|\mathbf{w}\|^2/2$

**Vincoli:**  $y_i[\mathbf{w} \cdot \mathbf{x}_i + b] - 1 \geq 0 \quad \text{per } i = 1 \dots n$

I pattern del training set che **giacciono sul margin** (cerchi pieni in figura) sono detti **support vector**. Tali pattern, che costituiscono i casi più complessi, **definiscono completamente** la soluzione del problema, che può essere espressa come **funzione di solo tali pattern**, indipendentemente dalla dimensionalità dello spazio  $d$  e dal numero  $n$  di elementi in TS.

# SVM lineari: Pattern Separabili (3)

Il **problema di ottimizzazione precedente**, può essere risolto passando innanzitutto a una **formulazione Lagrangiana** e successivamente a una **formulazione duale**.

La formulazione Lagrangiana prevede di introdurre un **moltiplicatore**  $\alpha_i$  ( $\alpha_i \geq 0$ ) per ogni vincolo nella forma **equazione**  $\geq 0$  e di sottrarre il vincolo moltiplicato per  $\alpha_i$  dalla **funzione obiettivo**:

$$Q(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^n \alpha_i (y_i [\mathbf{w} \cdot \mathbf{x}_i + b] - 1)$$

da **minimizzare** rispetto a  $\mathbf{w}$  e  $b$  e **massimizzare** rispetto a  $\alpha_i \geq 0$ .

Utilizzando le condizioni di **Karush-Kuhn-Tucker (KKT)**, il problema può essere posto in **forma duale** esprimendo i parametri  $\mathbf{w}$  e  $b$  in funzione dei moltiplicatori  $\alpha_i$ . Si elimina così la dipendenza diretta da  $\mathbf{w}$  e  $b$  e il problema può essere risolto **massimizzando la nuova funzione obiettivo** rispetto ai soli  $\alpha_i$ :

$$Q(\boldsymbol{\alpha}) = \sum_{i=1 \dots n} \alpha_i - \frac{1}{2} \sum_{i,j=1 \dots n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{con vincoli} \quad \sum_{i=1 \dots n} y_i \alpha_i = 0 \quad \text{e} \quad \alpha_i \geq 0 \quad \text{per } i = 1 \dots n$$

Si noti inoltre che i **pattern** compaiono ora **solo come prodotti scalari**.

Per approfondimenti e derivazione delle equazioni:

- S. Gunn, *Support Vector Machines for Classification and Regression*
- C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*
- <http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>

# SVM lineari: Pattern Separabili (4)

Il problema di ottimizzazione precedente può essere risolto attraverso un algoritmo di **programmazione quadratica** (disponibile in librerie numeriche).

La soluzione consiste nel derivare i valori ottimi  $\alpha_1^*, \alpha_2^* \dots \alpha_n^*$

Le condizioni KKT assicurano che  $\alpha_i^* = 0$  per tutti i vettori che non sono support vector.

L'iperpiano ottimo è dunque parametrizzato da:

$$\mathbf{w}^* = \sum_{i=1 \dots n} \alpha_i^* y_i \mathbf{x}_i$$

$$\text{e } b^* = y_s - \sum_{i=1 \dots n} \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}_s)$$

dove  $(\mathbf{x}_s, y_s)$  è uno dei **support vector**.

La funzione distanza dall'iperpiano è:

$$D(\mathbf{x}) = \mathbf{w}^* \cdot \mathbf{x} + b^* = \sum_{i=1 \dots n} \alpha_i^* y_i (\mathbf{x} \cdot \mathbf{x}_i) + b^*$$

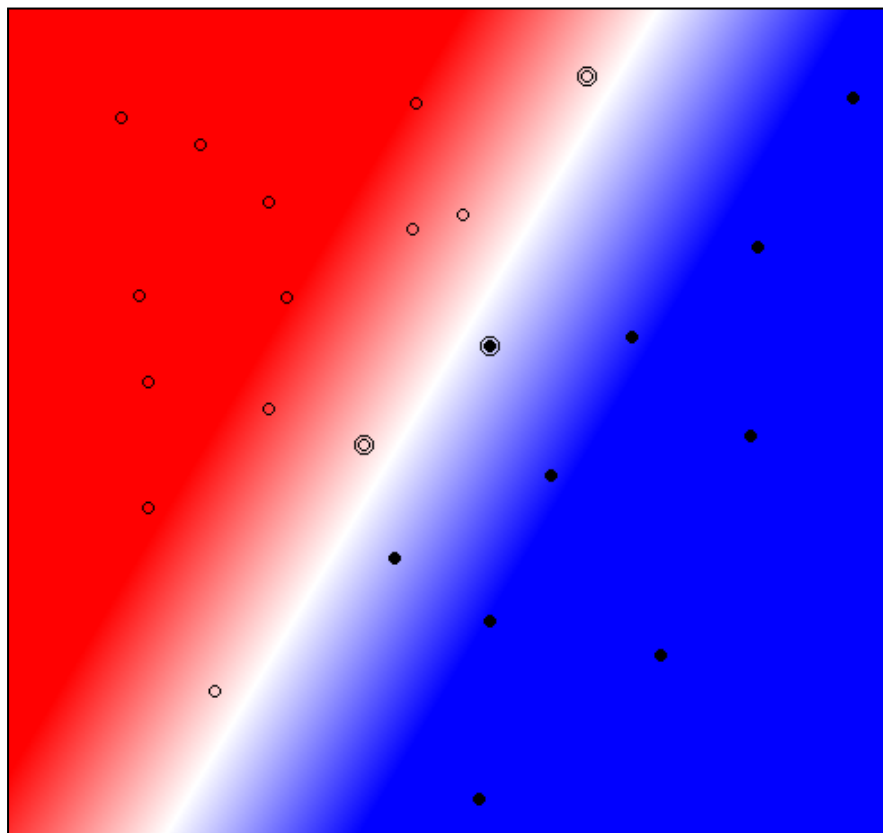
Si noti che:

- Il **segno** della funzione  $D(\mathbf{x})$  consente di classificare un generico pattern  $\mathbf{x}$ .
- Le sommatorie sono riducibili ai soli support vector.
- Nel caso lineare non è necessario, dopo aver calcolato  $\mathbf{w}^*$  e  $b^*$ , conservare/memorizzare i support vectors.

# SVM lineari: Pattern Separabili (4)

Vantaggi dell'approccio SVM:

- Definizione della soluzione sulla base di un numero ridotto di support vector (solitamente pochi).
- Il numero di support vector  $n_{sv}$  indica la complessità del problema e può essere dimostrato che l'errore medio (sui possibili training set) è limitato da  $n_{sv}/n$ .
- SVM «scala» molto bene rispetto alla dimensionalità  $d$  dello spazio delle feature (grazie ai prodotti scalari). La complessità computazionale nel training è quadratica rispetto al numero  $n$  di pattern in TS. In pratica il problema può essere risolto per  $d = 10^7$  e per  $n$  fino a  $10^4$ .



Esempio:  
i support vectors  
(cerchiati)  
definiscono la  
soluzione.



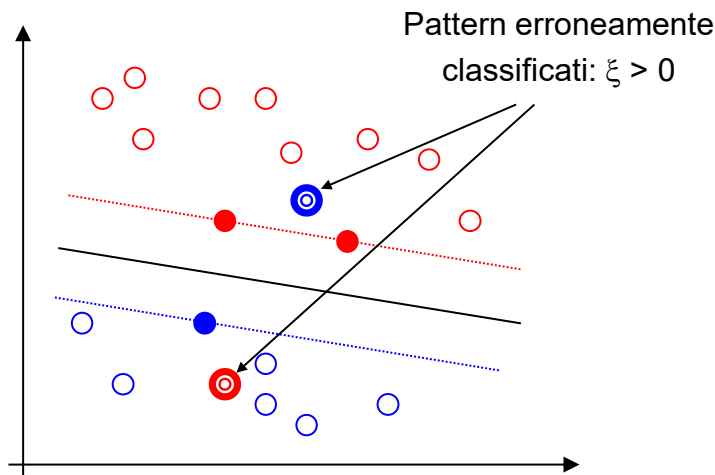
# SVM lineari: Pattern non Separabili

In questo caso **non tutti** i pattern possono essere separati da un iperpiano, ed è necessario **rilassare i vincoli di separazione**, per far sì che alcuni pattern (il minor numero possibile) possano valicare il confine della classe.

A tal fine si introducono  $n$  variabili di **slack** positive  $\xi_i$ ,  $i = 1 \dots n$  e si modificano i vincoli di separazione:

$$y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i \quad \text{per } i = 1 \dots n$$

Per ogni pattern  $\mathbf{x}_i$  del TS, la variabile  $\xi_i$  codifica la deviazione dal margine. Per i pattern **separabili** del TS le corrispondenti variabili di slack assumeranno **valore 0**.



L'**iperpiano ottimo** deve in questo caso **ancora massimizzare il margine**, ma allo stesso tempo **minimizzare il numero di elementi non correttamente classificati**. La funzione obiettivo, e di conseguenza il problema di ottimizzazione vengono così **modificati**:

**Minimizza:**  $\frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1 \dots n} \xi_i$

**Vincoli:**  $y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i \quad \text{per } i = 1 \dots n$

# SVM lineari: Pattern non Separabili (2)

Il coefficiente  $C$  nel problema di ottimizzazione precedente, indica l'importanza relativa degli errori di classificazione rispetto all'ampiezza del margine. Si tratta di uno dei pochi iperparametri che l'utente deve scegliere per il tuning di SVM.

Passando attraverso forma lagrangiana/duale otteniamo un risultato uguale al caso linearmente separabile, tranne che per l'introduzione del limite superiore ( $C$ ) per i valori dei moltiplicatori  $\alpha_i$ :

$$Q(\alpha) = \sum_{i=1 \dots n} \alpha_i - \frac{1}{2} \sum_{i,j=1 \dots n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

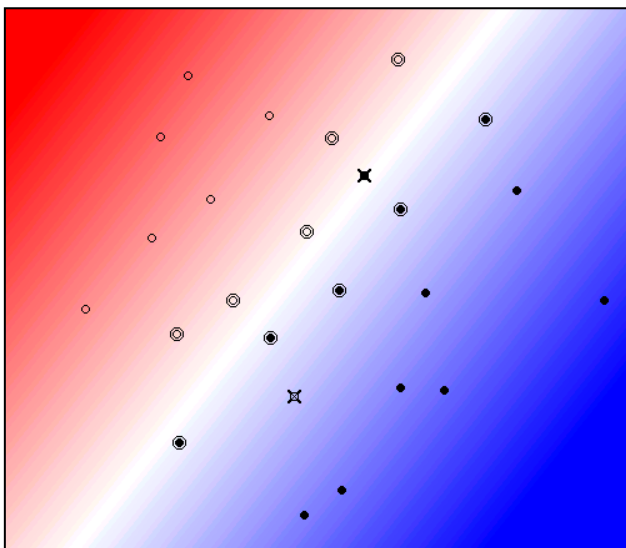
$$\text{con vincoli } \sum_{i=1 \dots n} y_i \alpha_i = 0 \quad \text{e} \quad 0 \leq \alpha_i \leq C \quad \text{per } i = 1 \dots n$$

Il metodo di soluzione (i.e. Progr. Quadratica) e il modo di derivare l'iperpiano dagli  $\alpha_i$  sono gli stessi del caso linearmente separabile.

Esempi:

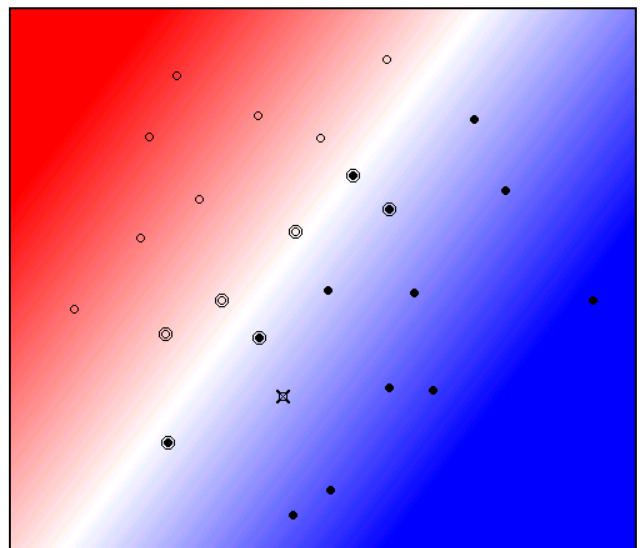
$C = 10$

2 errori, margine maggiore



$C = 200$

1 solo errore, margine minore



# SVM Non lineari

SVM prevede un'importante estensione della teoria inizialmente sviluppata per iperpiani, al caso (non lineare) di separazione dei pattern con superfici anche molto complesse. Tutto ciò avviene in modo molto semplice:

- Viene definito un mapping  $\Phi$  non lineare dei pattern dallo spazio di partenza  $\mathcal{R}^d$  verso uno spazio  $\mathcal{R}^m$  a più alta dimensionalità ( $m > d$ ):

$$\Phi: \mathcal{R}^d \rightarrow \mathcal{R}^m, \Phi(\mathbf{x}) = [g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_m(\mathbf{x})]$$

- Nello spazio  $\mathcal{R}^m$ , dove maggiori sono i gradi di libertà, i pattern  $\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), \dots, \Phi(\mathbf{x}_n)$  possono essere più facilmente separati da un iperpiano utilizzando la teoria nota. Ciò equivale a separare i pattern  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$  in  $\mathcal{R}^d$  con superfici arbitrariamente complesse.

Analizzando la formulazione del problema lagrangiano-duale, si nota che i vettori del training set appaiono solo in forma di prodotti scalari tra coppie di vettori. Questa proprietà (fondamentale) permette di evitare la manipolazione di vettori nello spazio  $\mathcal{R}^m$ .

Infatti, per opportuni mapping  $\Phi$  è possibile ricondurre il prodotto scalare di due pattern mappati nello spazio  $\mathcal{R}^m$  a una funzione  $K$  (detta Kernel) dei due pattern originali nello spazio  $\mathcal{R}^d$ .

$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = K(\mathbf{x}, \mathbf{x}')$$

Per il teorema di Mercer se la funzione Kernel  $K$  soddisfa alcuni requisiti (es. è continua e simmetrica rispetto agli argomenti) allora esiste un mapping  $\Phi$  verso uno spazio di dimensionalità maggiore che gode della suddetta proprietà. E' dunque possibile utilizzare  $K$  senza nemmeno conoscere il corrispondente  $\Phi$  (ad esempio nel caso di Kernel RBF,  $m = \text{infinito}$ ).

# SVM Non lineari: Kernel functions

## ■ Polinomio di grado $q$ (iperparametro):

Le componenti  $g_i(\mathbf{x}), i = 1..m$  sono ottenute come tutte le possibili combinazioni di elevamento a potenze  $\leq q$  delle componenti di  $\mathbf{x}$ . Ad esempio per  $d = 2, q = 2$ :

$$\Phi(\mathbf{x}) = \Phi([x_1, x_2]) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2, x_1^2 x_2, x_1 x_2^2, x_1^2 x_2^2]$$

e quindi  $m = 9$ .

Si dimostra che:

$$K(\mathbf{x}, \mathbf{x}') = [(\mathbf{x} \cdot \mathbf{x}') + 1]^q$$

## ■ Radial Basis Function (RBF) di ampiezza $\sigma$ (iperparametro):

$$K(\mathbf{x}, \mathbf{x}') = e^{\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)}$$

## ■ Sigmoid (meno utilizzato):

$$K(\mathbf{x}, \mathbf{x}') = \tanh(v(\mathbf{x} \cdot \mathbf{x}') + a)$$

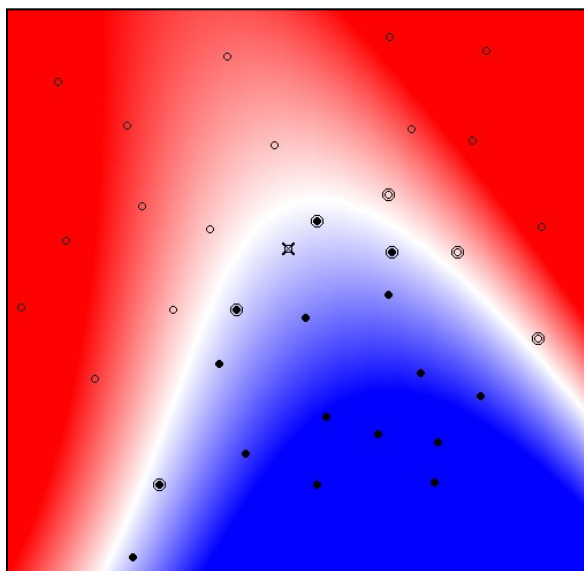
$v$  ed  $a$  (iperparametri) devono essere scelti opportunamente: una possibile scelta è:  $v = 1, a = 1$

Utilizzando il **kernel trick** si può **risolvere** il problema di ottimizzazione senza particolari complicazioni rispetto al caso lineare. Una volta determinati gli  $\alpha_i^*$ , la **superficie di separazione** (regola di classificazione) è esprimibile come:

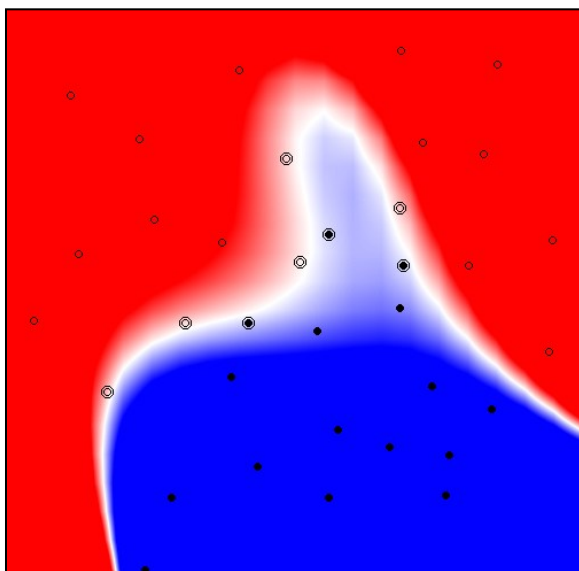
$$D(\mathbf{x}) = \sum_{i=1..n} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^*$$

# SVM Non lineari: Esempi

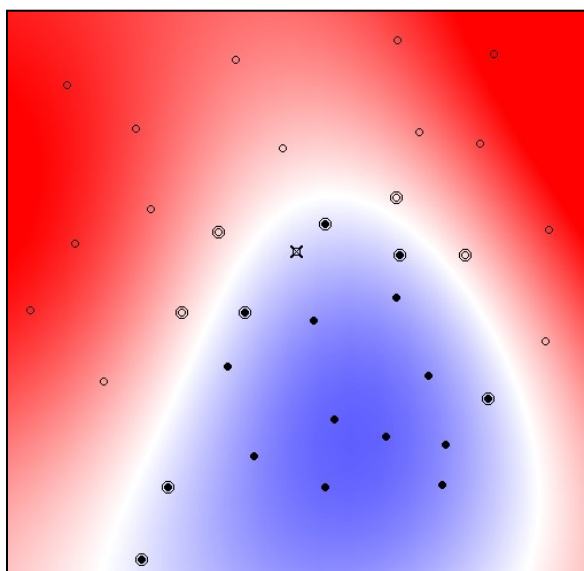
Polinomio  $q = 2$



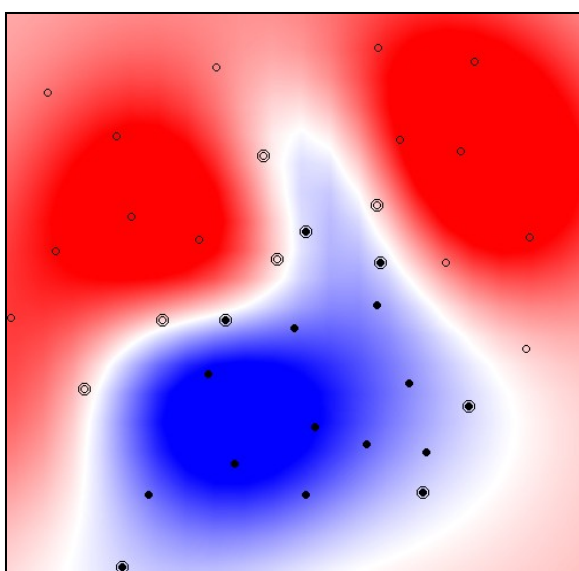
Polinomio  $q = 10$



RBF  $\sigma = 1$



RBF  $\sigma = 0.2$



# SVM: estensione multiclasse

SVM è in grado di determinare la **superficie di separazione** tra **2 classi** di pattern; come gestire allora i problemi con **più di 2 classi** ?

Si tratta di un problema ancora aperto anche se esistono diverse soluzioni; le più utilizzate sono:

- **One-Against-One**: che studieremo in seguito nell'ambito dei multi-classificatori.

- **One-Against-All**:

- Date **s classi**,  $w_1, w_2 \dots w_s$
- Per ogni classe  $w_k$ , si determina con SVM la **superficie di separazione** tra i pattern di  $w_k$  (**etichettati +1**) da una parte, e i pattern di tutte le rimanenti classi  $w_h$ ,  $h \neq k$  (**etichettati -1**) dall'altra, ottenendo la funzione  $D_k(\mathbf{x})$  che indica quanto  $\mathbf{x}$  è distante dalla superficie decisionale in direzione di  $w_k$ . Maggiore è  $D_k(\mathbf{x})$  più confidenti siamo dell'appartenenza di  $\mathbf{x}$  a  $w_k$ .
- Al termine del training, **si assegna il pattern**  $\mathbf{x}$  alla classe  $k^*$  per cui è massima la distanza dalla superficie decisionale:

$$k^* = \arg \max_k \{D_k(\mathbf{x})\}$$

Nota: È necessario eseguire **s** training SVM

# SVM: implementazione

Il training di SVM, richiede **algoritmi numerici non banali** in grado di risolvere un problema di programmazione quadratica. Alcune implementazioni sono **disponibili on-line**:

## ■ LIBSVM - <http://www.csie.ntu.edu.tw/~cjlin/libsvm>

- Attenzione i Kernel (RBF, ecc.) sono parametrizzati in modo diverso da quello comune (vedi *Readme.txt* di LibSvm e [1]). In particolare si fa uso del parametro gamma ( $\gamma$ ) per regolare la complessità della superficie decisionale. Aumentando  $\gamma$  la superficie può assumere forme più complesse.

- *N.B. Con kernel RBF  $\gamma$  opera in modo inverso rispetto a  $\sigma$ .*

- *Inserito  $\gamma$  anche nel kernel polinomiale (oltre al grado polinomio e Coef0)*

- Per la classificazione multiclasse utilizza internamente One-Against-One [2] (accurato ma inefficiente per molte classi).

- Wrapped da **Scikit-Learn** → `sklearn.svm.SVC`

[1] C.W. Hsu, C.C. Chang, and C.J. Lin, *A Practical Guide to Support Vector Classification*, disponibile sul sito web di LIBSVM

[2] C.C. Chang and C.J. Lin. *LIBSVM: a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011, disponibile sul sito web di LIBSVM

## ■ LIBLINEAR - <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>

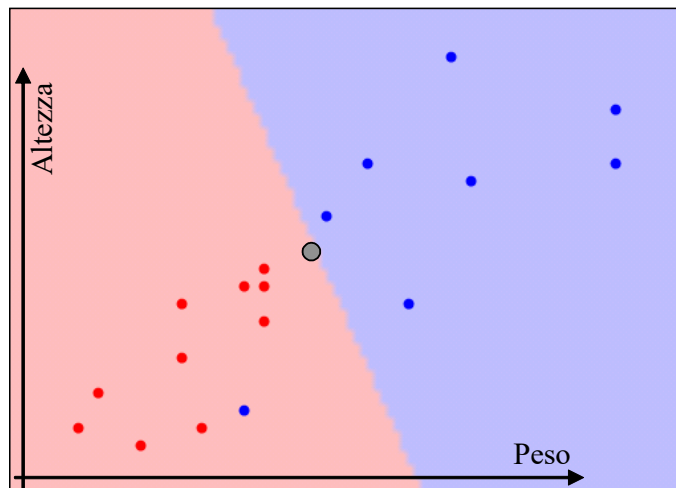
- Stessi autori di LIBSVM, consigliata nel caso lineare per elevata dimensionalità ed elevato numero di pattern.

- Wrapped da **Scikit-Learn** → `sklearn.svm.LinearSVC`

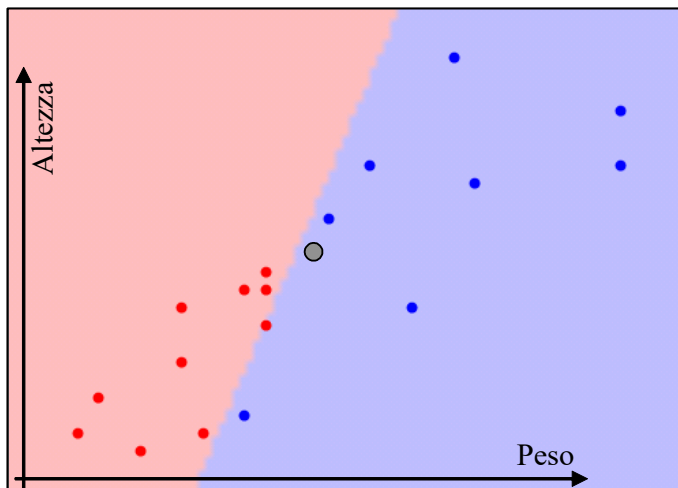
## ■ SVM-light - <http://svmlight.joachims.org>

# Esempi LibSvm (1)

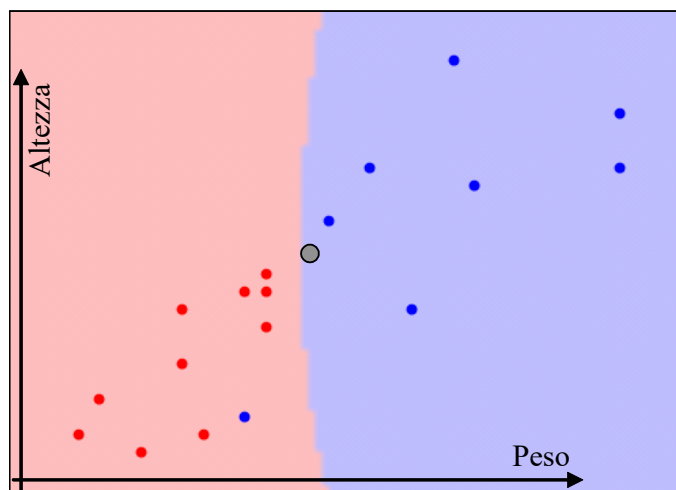
## «maschi-femmine»



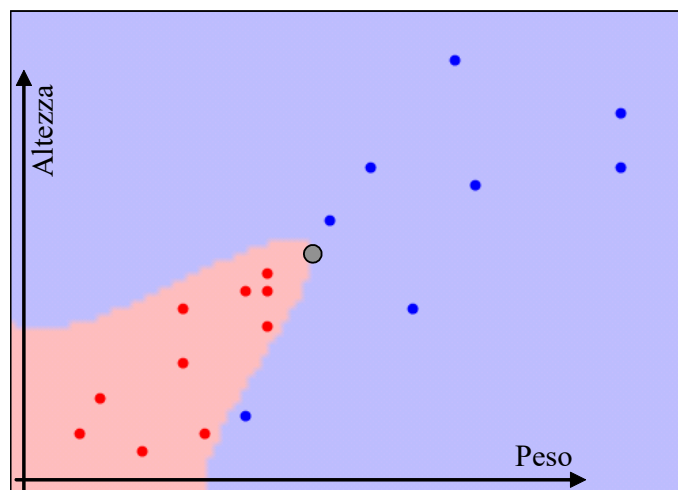
Lineare,  $C = 10$



Lineare,  $C = 500$



Polinomio  $q = 3$ ,  $C = 10$

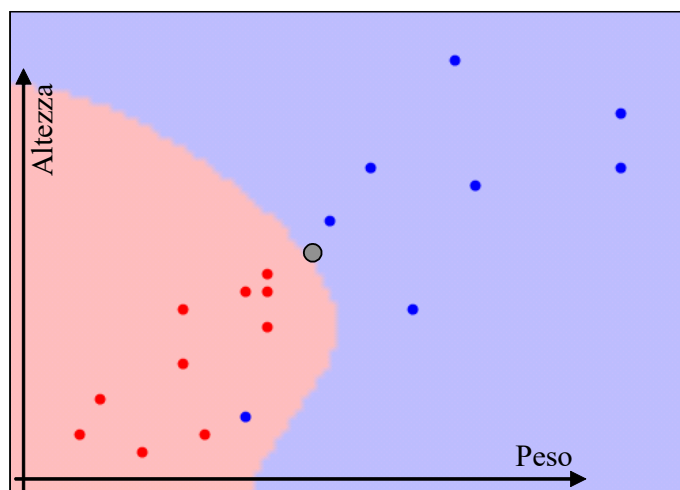


Polinomio  $q = 3$ ,  $C = 500$

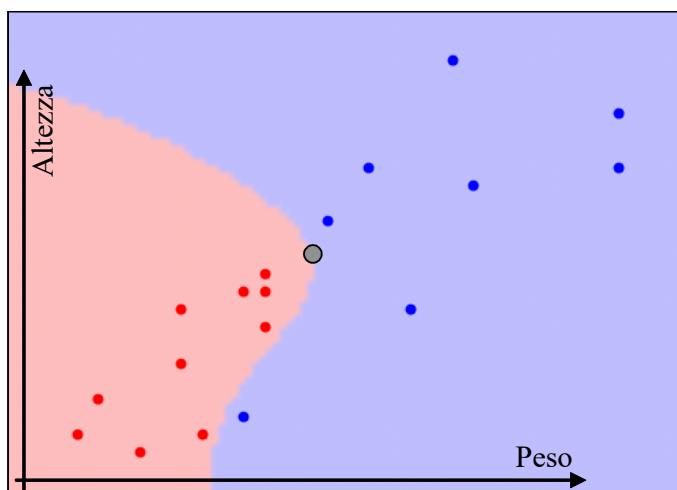


# Esempi LibSvm (2)

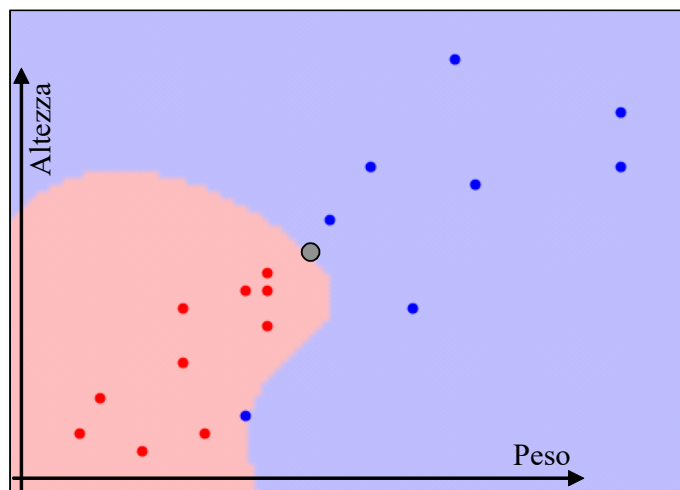
## «maschi-femmine»



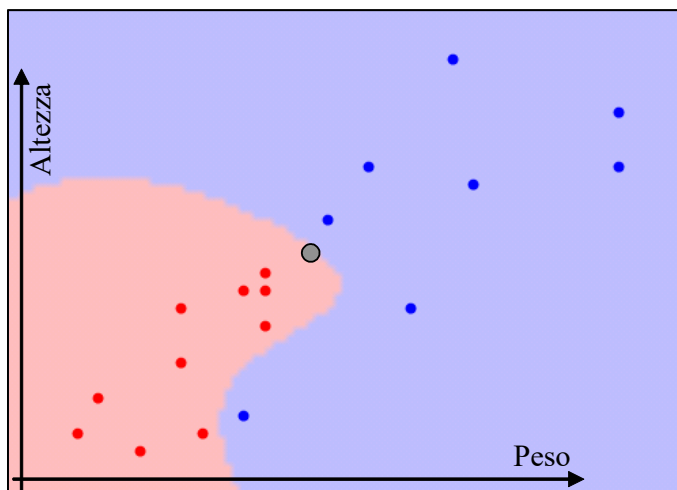
RBF,  $\gamma = 5, C = 10$



RBF,  $\gamma = 5, C = 500$



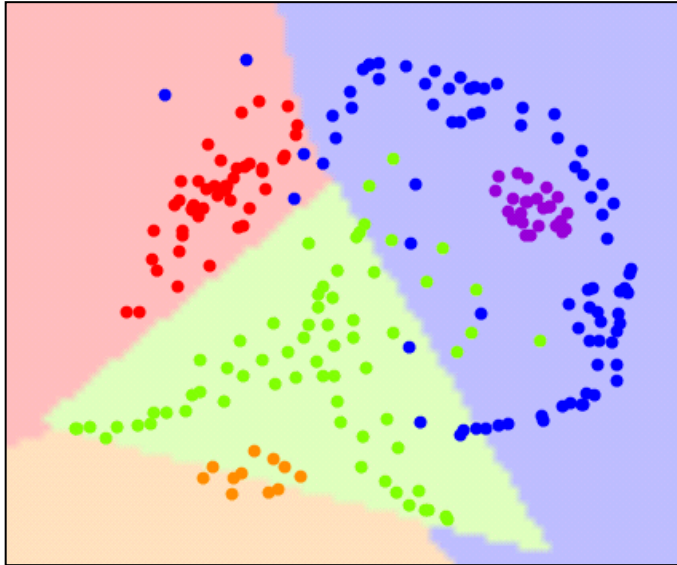
RBF,  $\gamma = 10, C = 10$



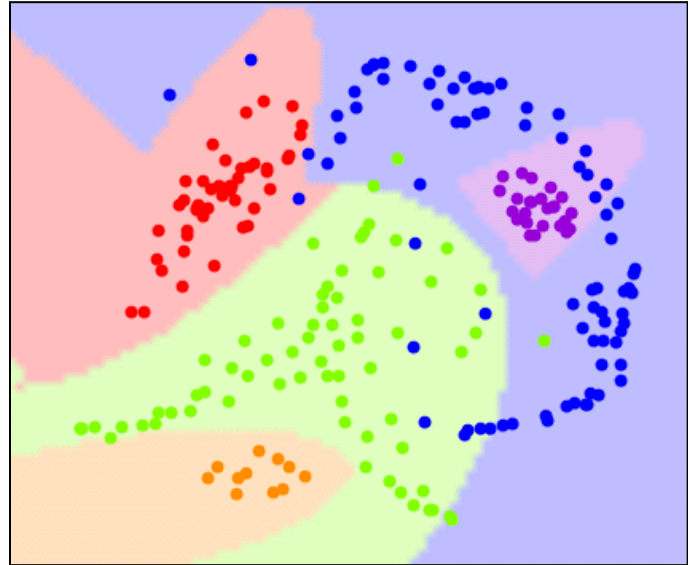
RBF,  $\gamma = 10, C = 500$

# Esempi LibSvm (3)

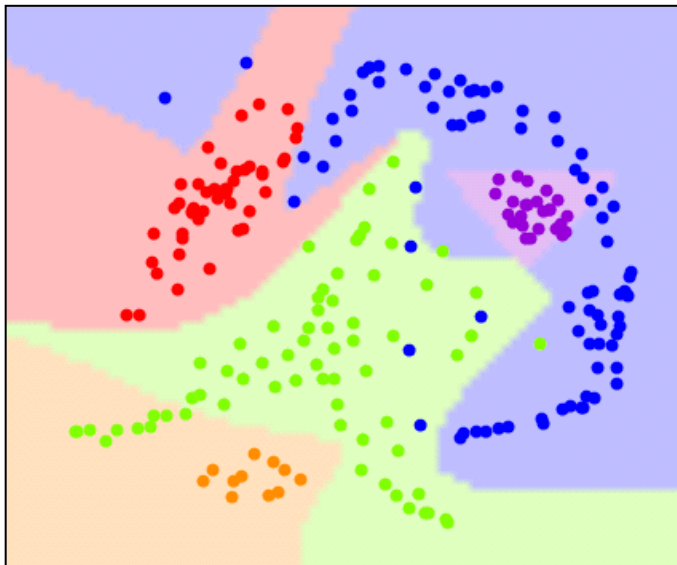
## multiclasse



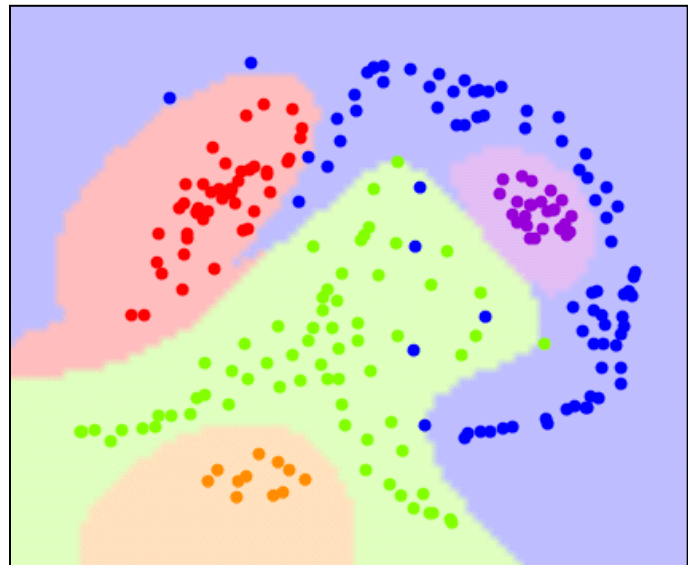
Lineare,  $C = 100$



Polinomio  $q = 2$ ,  $C = 100$



Polinomio  $q = 7$ ,  $C = 100$



RBF,  $\gamma = 5$ ,  $C = 100$

# SVM in pratica

## ■ Quando usare SVM?

- SVM permette di ottenere buoni risultati su dati numerici **omogenei** (anche di elevata dimensionalità), ma è meno performante di multi-classificatori basati su alberi decisionali (es. Random Forest) su dati **tabulari eterogenei**.

## ■ Lineare o Non-lineare?

- se la dimensionalità  $d$  dello spazio è molto **elevata** (es. 5000 feature) si utilizza generalmente SVM **lineare**. Infatti in uno spazio così grande i pattern sono tipicamente molto sparsi e anche «semplici» iperpiani sono in grado di separare le classi efficacemente. Il solo iperparametro da tarare è  $C$ .
- per **bassa** dimensionalità (es. 20 feature) la scelta primaria è **SVM non lineare** con **kernel RBF**. Gli iperparametri da tarare sono  $C$  e  $\sigma$  (o  $\gamma$  se si utilizza LIBSVM).
- Per media dimensionalità (es. 200 features) in genere si provano entrambe le tipologie (i.e., anche questa scelta diventa un iperparametro).
- Come sempre gli iperparametri si tarano su un validation set separato, oppure attraverso cross-validation sul training set.

## ■ Come gestire il caso multi-classe?

- Tipicamente ci si affida alla soluzione **disponibile** nella libreria utilizzata (**One-Against-One** per LIBSVM).
- Se però il numero di classi è molto elevato, il costo può diventare inaccettabile per certe applicazioni. In questo caso **One-Against-All** diventa la scelta obbligata.

# Multi-Classificatori

## (Ensemble methods)

Un **multi-classificatore** è un approccio dove **diversi classificatori** sono utilizzati (*normalmente in **parallelo**, ma talvolta anche in **cascata** o in **modo gerarchico***) per eseguire la classificazione dei pattern; le **decisioni** dei singoli classificatori sono **fuse** ad un certo livello del processo di classificazione.

È stato dimostrato (***teoricamente** ma soprattutto nella **pratica***) che l'utilizzo di **combinazioni di classificatori** (in inglese **multi-classifier**, **combination of classifiers**, **classifier fusion**, **ensemble learning**) può migliorare, anche molto, le **prestazioni**.

- **Siate pragmatici!** Nella pratica investire tempo nell'ottimizzazione «spinta» di un singolo classificatore è in genere meno conveniente rispetto all'affiancamento (al classificatore iniziale) di altri classificatori.
- Attenzione però: la **combinazione** è **efficace** solo quando i singoli classificatori sono (almeno parzialmente) **indipendenti tra loro**, ovvero non commettono gli stessi errori.
- L'**indipendenza** (o **diversità**) è normalmente ottenuta:
  - Utilizzando **feature diverse** (non correlate o poco correlate)
  - Utilizzando **algoritmi diversi** per l'estrazione delle **feature**
  - Utilizzando **diversi algoritmi di classificazione**
  - Addestrando lo **stesso algoritmo** di classificazione su **porzioni diverse** del training set (**bagging**)
  - **Insistendo** con l'addestramento sugli **errori** commessi dai predecessori (**boosting**)
- La **combinazione** può essere eseguita a **livello di decisione** o a **livello di confidenza**.

# Fusione a livello di decisione

Ogni **singolo classificatore** fornisce in output la propria **decisione** che consiste nella **classe** cui ha assegnato il pattern. Le **decisioni** possono essere tra loro **combinare** in diversi modi, tra cui:

- **Majority vote rule**: è uno dei più noti e semplici metodi di fusione; ogni classificatore **vota per una classe**, il pattern viene assegnato alla classe **maggiormente votata**.

Più formalmente:

Se  $\{C_1, C_2, \dots, C_{nc}\}$  è un insieme di **nc** classificatori, e

$$\theta_{ij} = \begin{cases} 1 & \text{se } i \text{ è la classe votata da } C_j \\ 0 & \text{altrimenti} \end{cases} \quad (1 \leq i \leq s, 1 \leq j \leq nc)$$

Allora il pattern è assegnato alla classe **t** tale che:

$$t = \arg \max_{i=1..s} \left\{ \sum_{j=1..nc} \theta_{ij} \right\}$$

- **Borda count**: ogni classificatore **invece** di una **singola classe**, produce una **classifica** o **ranking** delle classi (dalla prima all'ultima) a seconda della probabilità che a ciascuna di esse appartenga il pattern da classificare.

I ranking sono **convertiti** in **punteggi** e poi **sommati**; la classe con il più **elevato punteggio** finale è quella scelta dal multi-classificatore.

Rispetto a majority vote rule, considera anche i «non primi» di ogni classificatore.

# Prestazioni teoriche

- Dato un problema di classificazione **binario** e un **multi-classificatore** costituito da  $nc$  classificatori ( $nc$  dispari), la **majority vote rule** classifica il pattern come appartenente alla classe che ha ottenuto almeno  $k = (nc + 1)/2$  voti.
- Sia  $P$  la probabilità che ogni singolo classificatore esegua la classificazione correttamente, la probabilità  $P_{multi}(nc)$  che la **decisione del multi-classificatore** sia **corretta** è esprimibile attraverso la **distribuzione binomiale**:

$$P_{multi}(nc) = \sum_{m=k}^{nc} \binom{nc}{m} P^m (1 - P)^{nc-m}$$

*Ricorda, la distribuzione binomiale permette di calcolare con quale probabilità estraiamo  $m$  palline rosse da un'urna contenete  $nc$  palline, se  $P$  è la probabilità di estrarne una rossa in un singolo tentativo.*

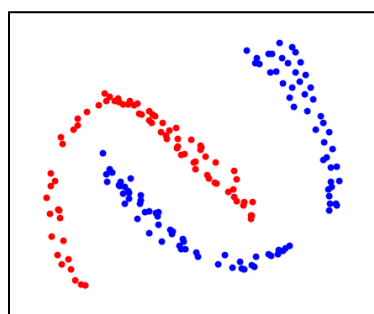
Nella formula di  $P_{multi}(nc)$  i casi favorevoli sono quelli in cui un numero di classificatori compreso tra  $k$  e  $nc$  hanno effettuato classificazione corretta.

- **Esempio:** se  $P = 0.8$  (ogni classificatore commette il **20%** di errore di classificazione) stando alla formula precedente per un multi-classificatore con  $nc$  (variabile) componenti:

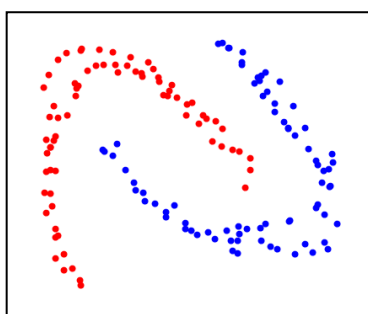
- ▶  $P_{multi}(3) = 0.896$  ( $k = 2$ )
- ▶  $P_{multi}(5) = 0.942$  ( $k = 3$ )
- ▶  $P_{multi}(7) = 0.966$  ( $k = 4$ )
- ▶  $P_{multi}(9) = 0.980$  ( $k = 5$ )
- ▶  $P_{multi}(15) = 0.995$  ( $k = 8$ )
- ▶  $P_{multi}(21) = 0.999$  ( $k = 11$ )      **Eccezionale !!!**

**Dov'è il trucco?** La formula precedente è basata sull'**indipendenza (totale) dei classificatori**, cosa difficilmente ottenibile in pratica!

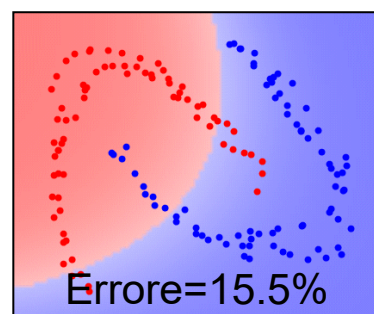
# Majority vote rule (esempio)



Training set

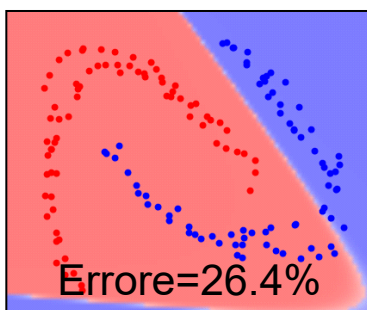
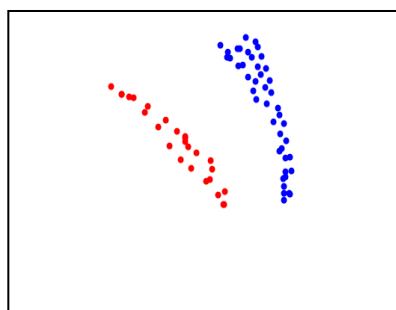
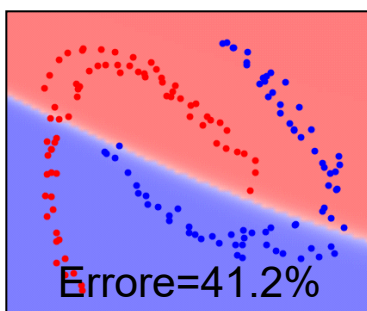
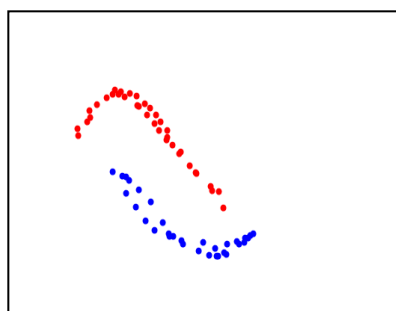
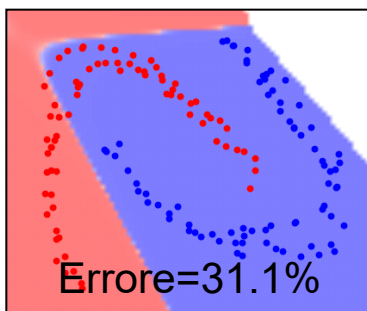
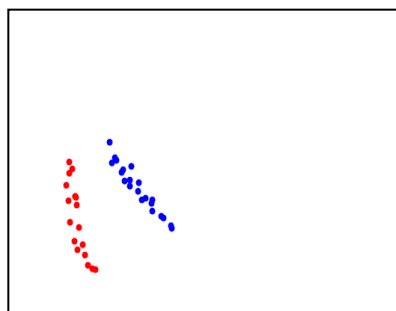


Test set

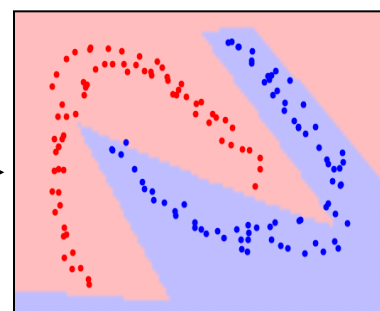


(Bayes – Distr. Normali)

Dividendo il training set in 3 parti e addestrando 3 classificatori (stesso tipo):



Majority vote rule:



Errore=0.7%

# One-Against-One

L'approccio **One-Against-One**, consente di risolvere un problema di classificazione **multi-classe**, attraverso classificatori **binari**.

È l'approccio adottato dalla libreria **LIBSVM** (usata in BioLab).

- Se  $s$  sono le classi del problema, si addestrano  $s \times (s - 1)/2$  classificatori binari: **tutte le possibili coppie, indipendentemente dall'ordine**.
- Durante la classificazione, il pattern  $x$  viene classificato da ogni classificatore binario, che assegna un voto alla classe (tra le due) più probabile.
- Al termine il pattern  $x$  è assegnato alla classe che ha ricevuto più voti (**majority vote rule**).

È in genere più accurato di **One-Against-All** (discusso in precedenza per SVM), anche se meno efficiente in quanto richiede l'addestramento di un numero maggiore di classificatori.



# Fusione a livello di confidenza (1)

Ogni **singolo classificatore**  $C_j$ ,  $j = 1..nc$  fornisce in output la **confidenza di classificazione** del pattern rispetto a ciascuna delle classi, ovvero un vettore  $\mathbf{conf}_j = [conf_{j1}, conf_{j2} \dots conf_{js}]$  in cui l'**i-esimo elemento** indica il grado di appartenenza del pattern alla **classe i-esima**.

Diversi metodi di fusione sono possibili tra cui: **somma** (sum), **prodotto** (prod), **massimo** (max) e **minimo** (min):

$$\blacksquare \text{ **sum** } = \sum_{j=1..nc} \mathbf{conf}_j \quad t = \arg \max_{i=1..s} \{ \text{sum}_i \}$$

$$\blacksquare \text{ **prod** } = \prod_{j=1..nc} \mathbf{conf}_j \quad t = \arg \max_{i=1..s} \{ \text{prod}_i \}$$

$$\blacksquare \text{ **max}_i = \max_{j=1..nc} \{ \text{conf}_{ji} \} \quad t = \arg \max_{i=1..s} \{ \text{max}_i \}**$$

$$\blacksquare \text{ **min}_i = \min_{j=1..nc} \{ \text{conf}_{ji} \} \quad t = \arg \max_{i=1..s} \{ \text{min}_i \}**$$

- Il criterio del minimo può sembrare illogico. *Attenzione scegliamo il massimo dei minimi ... quindi una classe che non ha ricevuto confidenza troppo bassa da nessun classificatore.*
- Il prodotto è il metodo «**probabilisticamente**» più corretto (la probabilità congiunta si calcola come prodotto di probabilità) ma solo nel caso di indipendenza statistica.
- Nella pratica la **somma è spesso preferibile al prodotto** in quanto più robusta. Infatti nel prodotto è sufficiente che un solo classificatore indichi confidenza zero per una classe per portare a zero la confidenza del multi-classificatore per quella classe.

# Fusione a livello di confidenza (2)

- Una **variante efficace**, è quella della **somma pesata**, dove la somma dei vettori confidenza è eseguita pesando i diversi classificatori in base al loro **grado di abilità**  $g_j$ .

$$\text{avgsum} = \sum_{j=1 \dots nc} g_j \cdot \text{conf}_j \quad t = \arg \max_{i=1 \dots s} \{ \text{avgsum}_i \}$$

I **gradi di abilità** possono essere **definiti** in base alle **singole prestazioni** dei classificatori, ad esempio in maniera **inversamente proporzionale** all'**errore** di classificazione (vedi AdaBoost).

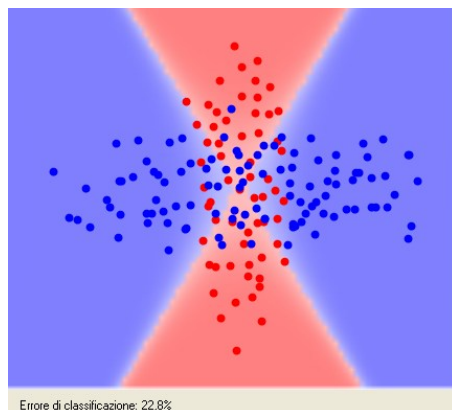
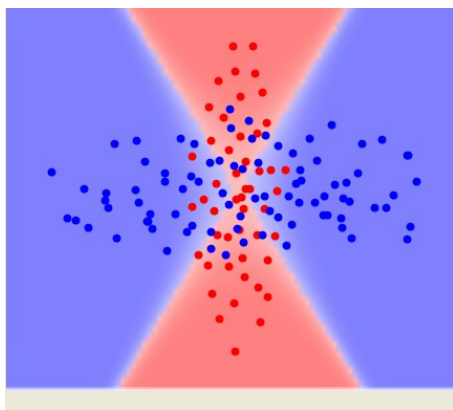
- Attenzione alla **normalizzazione** dei **vettori di confidenza** nel caso in cui essi **non siano probabilità** (ma ad esempio similarità). *Riferimento a quanto detto per la normalizzazione delle distanze.*

# Fusione a livello di confidenza: esempio

Training Set

Test Set

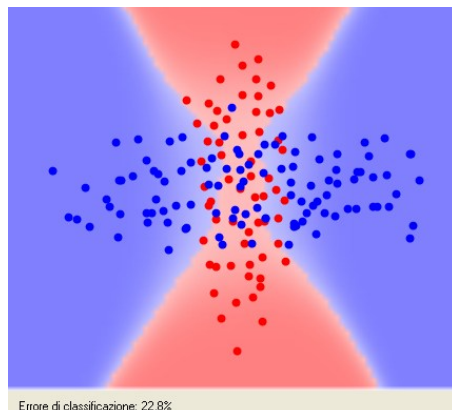
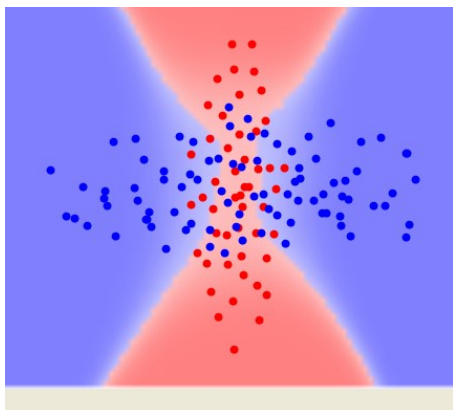
Bayes parametrico  
(normali)



Errore di classificazione: 22,8%

Errore=22.8%

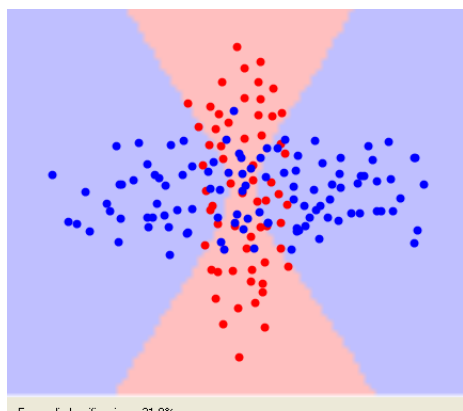
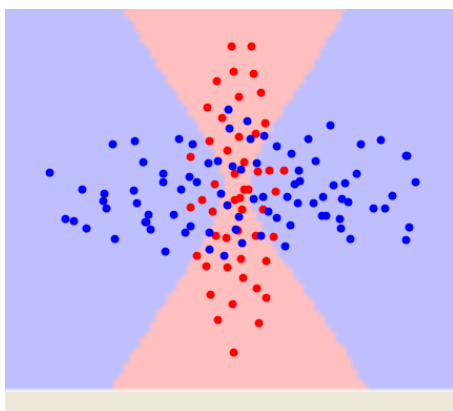
Bayes & Parzen  
(normali  $h=10$ )



Errore di classificazione: 22,8%

Errore=22.8%

Fusione a livello di  
confidenza  
(somma)



Errore di classificazione: 21,0%

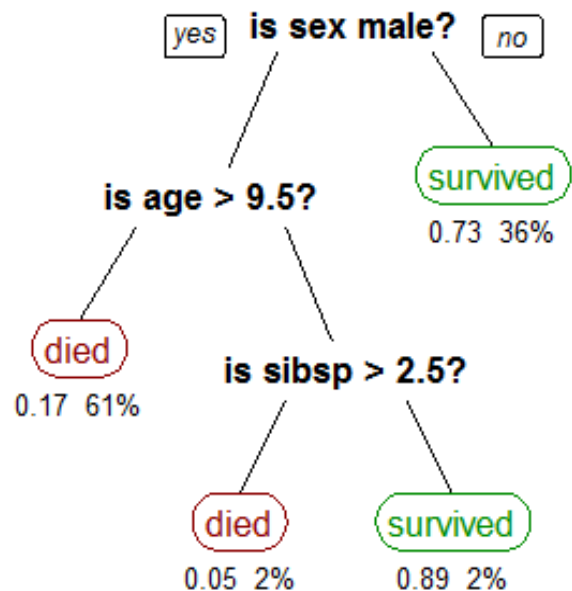
Errore=21.0%

# Random Forest

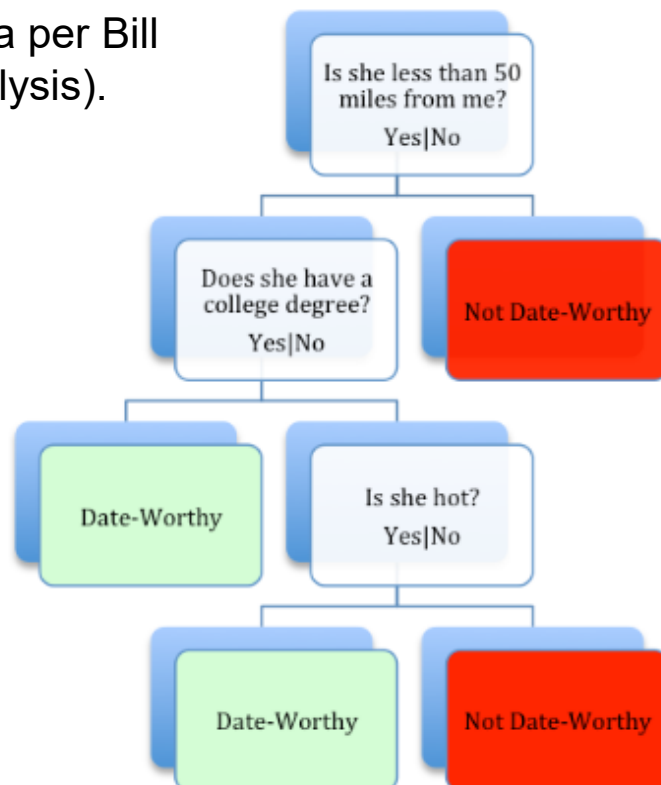
- Ideato da Leo Breiman nel 2001.
- Appartiene alla famiglia di metodi di Bagging (Bootstrap Aggregating):
  - Estrazione con re-imbussolamento di un sottoinsieme  $S_j$  di pattern dal training set (tipicamente i 2/3 del totale).
  - Addestramento del classificatore  $C_j$  sul sottoinsieme  $S_j$
  - Fusione dei classificatori (e.g., majority vote rule, somma)
- In Random Forest i singoli classificatori sono classification tree (alberi di classificazione):
  - I classification tree sono uno strumento molto utilizzato in applicazioni con pattern categorici o mixed (es. Data Mining). Per pattern numerici «puri» in genere non sono competitivi con SVM se considerati singolarmente.
  - Esistono diversi tipi di alberi, tra cui: ID3, C4.5, CART (default in Random Forest).
  - CART è un albero binario in cui ogni nodo divide i pattern del training set (assegnandoli ai due nodi figli). La suddivisione si basa sul confronto di una singola feature con una soglia.
  - Per la «crescita» dell'albero a partire da un training set, si sceglie (in modo greedy) ad ogni livello la coppia (feature, soglia di suddivisione) che meglio separa le classi. Vedi [A. Géron] per maggiori dettagli.
  - Per la classificazione di un nuovo pattern si visita l'albero e una volta giunti a una foglia, si classifica il pattern sulla base della classe più comune nel nodo (tra i pattern del training set): majority vote rule.

# Esempi di classification tree

- Sopravvivenza passeggeri del **Titanic** (fonte Wiki).
- Classi: **died** e **survived**
- **sibsp**: numero familiari a bordo.
- I numeri sotto le foglie indicano la probabilità di sopravvivenza (sinistra) e la percentuale di campioni nella foglia (destra).



- La partner giusta per Bill (fonte GormAnalysis).



# Random Forest (2)

- In Random Forest, per rendere maggiormente indipendenti i classificatori (tree):
  - per ogni nodo la scelta della feature migliore su cui partizionare non è fatta sull'intero insieme delle  $d$  feature (dimensionalità dei pattern), ma su un sottoinsieme random di  $d'$  feature. Valore tipico  $d' = \sqrt{d}$
  - In assenza di questo accorgimento (noto anche come **feature bagging**) molti tree sceglierebbero con elevata probabilità le stesse variabili (quelle più discriminanti).
  - Pertanto Random Forest opera simultaneamente **due tipi di bagging**: uno sui pattern del training set e uno sulle features.
- Il numero di tree (iperparametro **n\_estimators**) in una forest varia generalmente da alcune centinaia ad alcune migliaia. Aumentare **n\_estimators** oltre al valore ottimale in genere non produce overfitting (ma rende il sistema meno efficiente).
- Gli alberi possono essere fatti crescere fino a quando lo split dei nodi termina **naturalmente** (in quanto non porta a una migliore separazione delle classi) oppure imponendo una massima profondità (iperparametro **max\_depth**). Ridurre **max\_depth** ha effetto **regolarizzante** e riduce **overfitting**.
- Grazie al bagging, le prestazioni possono essere stimate con tecnica **Out-Of-Bag (OOB)** che non richiede validation set separato.
  - La out-of-bag decision function (per random forest) in **scikit-learn** calcola, per ogni pattern, la prediction media di tutti gli alberi che non includono tale pattern nel loro bagging set.

# AdaBoost

- Ideato da Freund and Schapire nel 1995.
- Appartiene alla famiglia dei metodi di boosting, dove:
  - più classificatori (tipicamente weak = deboli) sono combinati per realizzare un classificatore strong (forte).
  - a differenza del bagging, l'apprendimento è incrementale e prevede di aggiungere a ogni iterazione un classificatore efficace sui pattern critici.
  - ad ogni iterazione si assegna un peso a ciascun pattern del training set (aggiornando il peso precedente). Pattern classificati erroneamente hanno peso maggiore e contribuiscono maggiormente nella selezione del prossimo classificatore.
- Per semplicità nel seguito si introduce una versione binaria, ma esiste anche una versione multiclasse.
- Siano:
  - $(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)$  i pattern del training set (TS) e le rispettive etichette, con  $y_i \in \{+1, -1\}$
  - $w_i^t$  il peso del pattern  $\mathbf{x}_i$  all'iterazione  $t$
  - $h^t(\mathbf{x}) \rightarrow \{+1, -1\}$  il classificatore introdotto all'iterazione  $t$  e scelto tra i classificatori  $h_j^t(\mathbf{x})$  disponibili all'iterazione  $t$
  - $\alpha^t$  il peso del classificatore  $h^t(\mathbf{x})$  nell'ambito del multi-classificatore.

# AdaBoost: pseudocodice

$w_i^1 = \frac{1}{n}, i = 1..n$  (tutti i pattern hanno lo stesso peso iniziale)

For  $t = 1 \dots T$

Normalizza i pesi (somma 1):  $w_i^t / \sum_{j=1..n} w_j^t, i = 1..n$

Per ogni classificatore  $h_j^t(\mathbf{x})$  disponibile (per la scelta)

Calcola errore (pesato sull'importanza dei pattern)

$$\epsilon_j = \sum_{i=1..n} w_i^t \cdot I(h_j^t(\mathbf{x}_i) \neq y_i)$$

$$I(cond) = \begin{cases} 1 & cond = true \\ 0 & altrimenti \end{cases}$$

Scegli come  $h^t(\cdot)$  il classificatore  $h_j^t(\cdot)$  con minor  $\epsilon_j$

Calcola il peso del classificatore (vedi slide successiva)

$$\alpha^t = \frac{1}{2} \ln \left( \frac{1 - \epsilon^t}{\epsilon^t} \right), \quad \text{dove } \epsilon^t = \min \epsilon_j$$

Aggiorna i pesi (vedi slide successiva)

$$w_i^{t+1} = w_i^t \cdot e^{-\alpha^t y_i h^t(\mathbf{x}_i)}$$

End for

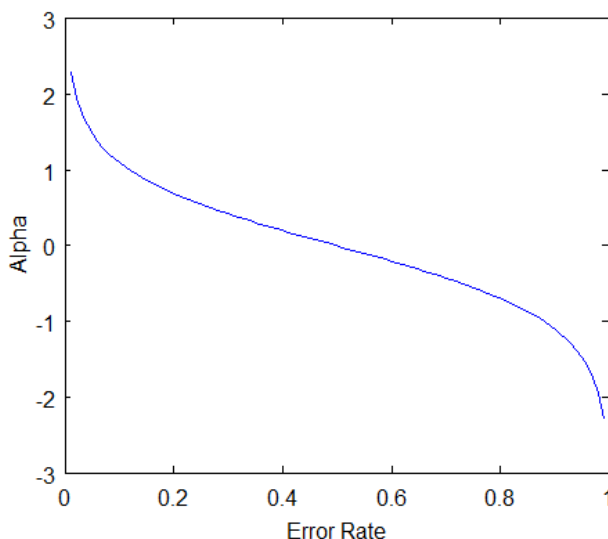
Al termine dell'addestramento, il **classificatore finale** (**strong**) è la media pesata dei weak classifiers:

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1..T} \alpha^t h^t(\mathbf{x}) \right)$$



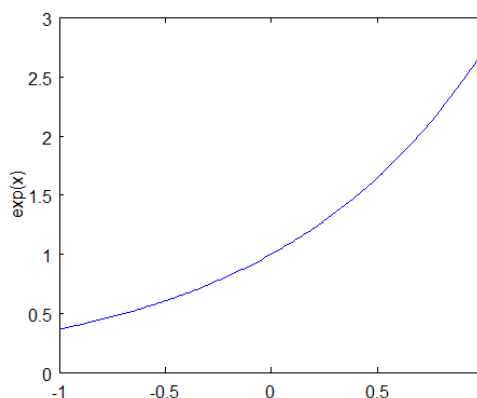
# AdaBoost: scelta dei pesi

$$\alpha^t = \frac{1}{2} \ln \left( \frac{1 - \epsilon^t}{\epsilon^t} \right)$$



Un classificatore binario con errore 0.5 ha peso 0 (neutro), il peso aumenta esponenzialmente quando l'errore si avvicina a 0 e diventa negativo se l'errore è superiore a 0.5 (*fai l'opposto di ciò che dice il classificatore!*).

$$w_i^{t+1} = w_i^t \cdot e^z \quad \text{dove}$$
$$z = -\alpha^t y_i h^t(\mathbf{x}_i)$$



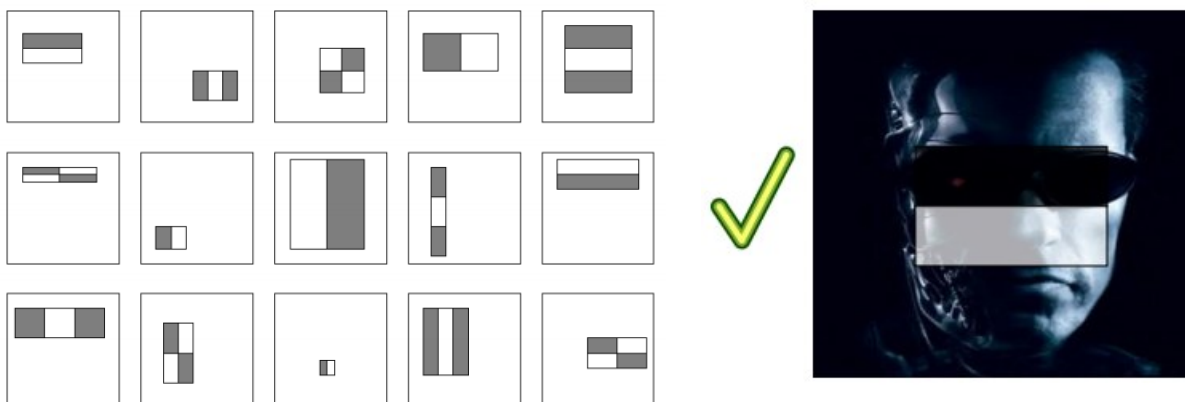
osservando che:

$$y_i h^t(\mathbf{x}_i) = \begin{cases} 1 & \text{se il pattern è correttamente classificato} \\ -1 & \text{altrimenti} \end{cases}$$

- $w_i^{t+1}$  aumenta se  $z > 0$ :  $\alpha^t$  positivo e classificazione errata, oppure  $\alpha^t$  negativo e classificazione corretta
- $w_i^{t+1}$  diminuisce se  $z < 0$ : nei 2 casi complementari.

# Viola & Jones Face detector (2001)

- È una delle più note applicazioni di Adaboost.
- I **weak classifiers**  $h_j$  sono scelti tra circa 160000 filtri rettangolari (**Haar features**) che possono essere calcolati in modo molto efficiente attraverso l'immagine integrale.



$$h_j(\mathbf{x}) = \begin{cases} 1 & \Sigma (\text{pixels in white area}) - \Sigma (\text{pixels in black area}) > \theta_j \\ -1 & \text{altrimenti} \end{cases}$$

- Per poter scartare velocemente non-face l'approccio di Viola & Jones utilizza una **cascata** di **strong** classifier (**ciascuno** ottenuto con **AdaBoost**) che operano sequenzialmente.
- Se uno dei classificatori (ad un qualsiasi livello della sequenza) classifica il pattern come (**non face**) questo viene scartato immediatamente.
- Gli strong classifier devono essere **scelti** e **tarati** per **minimizzare i falsi negativi** piuttosto che l'errore totale di classificazione.
- I classificatori all'inizio della sequenza sono semplici e veloci (*il primo utilizza solo due weak classifiers e scarta circa il 50% dei pattern non-face*).

# Gradient Boosting

- È una tecnica di boosting che invece di assegnare peso maggiore agli hard examples, cerca di addestrare i nuovi weak models sui **residuali** di errore (prodotti dai predecessori).
- L'implementazione più semplice (anche da capire) si applica al caso di **regressione con loss MSE**, maggiori dettagli sono forniti nelle slide sulla di regressione.
- L'implementazione ottimizzata **XGBoost** ha interfaccia fit/predict standard di scikit-learn e può essere facilmente aggiunta al toolkit dei modelli da considerare per la risoluzione di un problema. Vedi [link](#).