

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 multclasse
- 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 ...

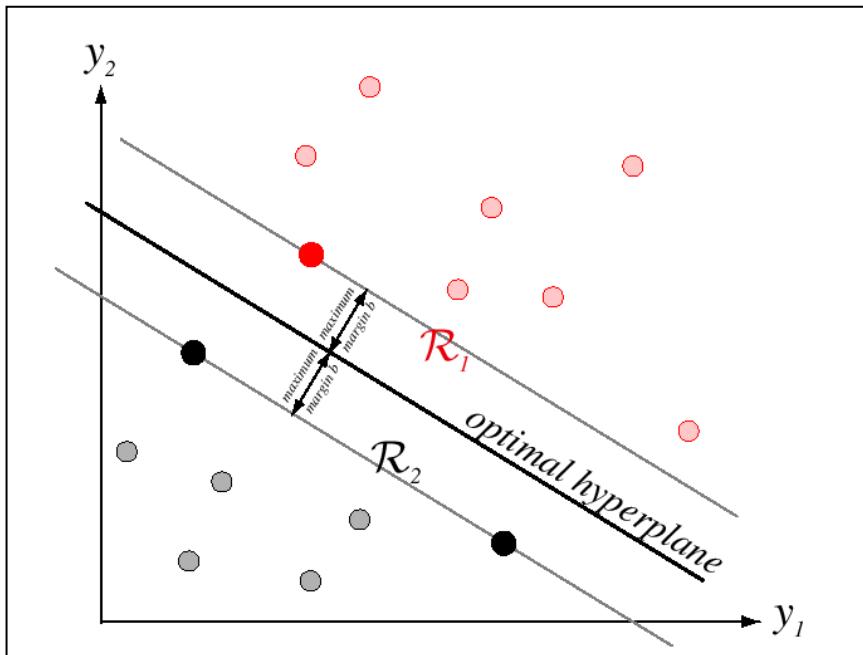
SVM nasce come classificatore binario (2 classi), estendibile a più classi. Affrontiamo la trattazione per gradi:

- 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 multclasse.

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 *margine* possibile.

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



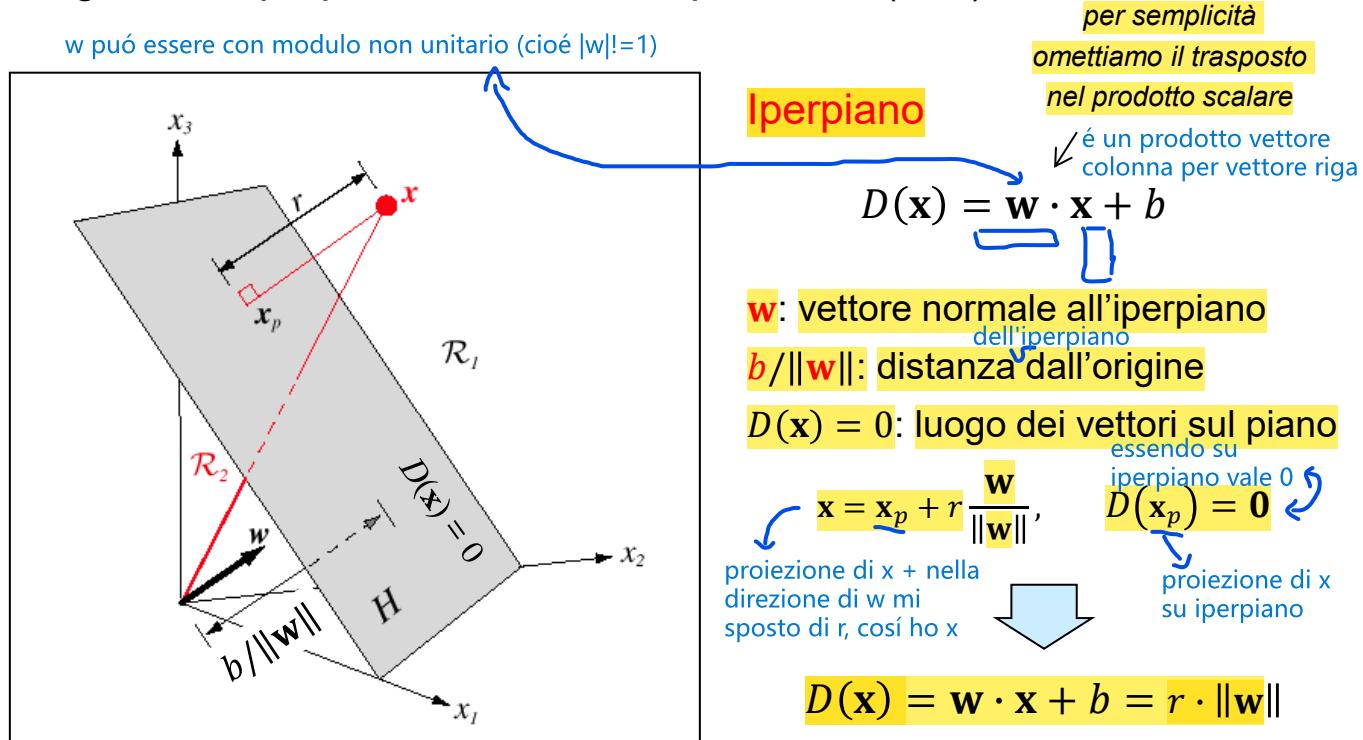
La massimizzazione del margine è legata alla *generalizzazione*. Se i pattern del training set sono classificati con ampio margine si può «sperare» che anche pattern del test set vicini al confine tra le classi siano gestiti correttamente.

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).

ottimizzazione vincolata: separazione totale con margine più alto
(ottimizzazione quadratica)

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) :



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:

É $>= +1$ e $<= -1$ perché devo considerare il margine (tutti i punti delle due classi devono stare sopra e sotto il margine)

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

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

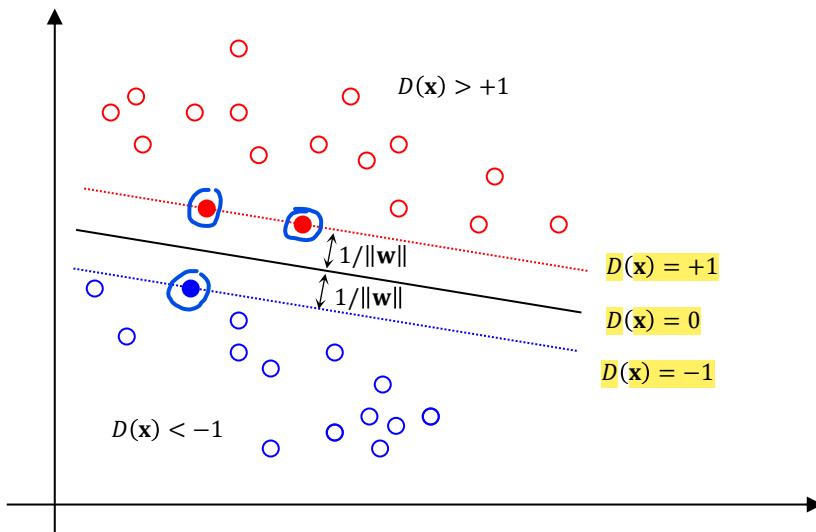
o in modo più compatto:

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

SVM lineari: Pattern Separabili (2)

La minima distanza tra l'iperpiano di separazione e un pattern del training set è detta **margine** (τ). Il margine è la distanza dei support vectors dal decision boundary

I punti che mi interessano nella definizione del margine e decision boundary sono i support vectors



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 margine è $\tau = 2/\|\mathbf{w}\|$.

che

L'iperpiano ottimo secondo SVM è quello che soddisfa i vincoli di separazione dei pattern e massimizza il margine τ (o alternativamente minimizza il suo inverso):

Questi vincoli assicurano che ogni punto di addestramento (\mathbf{x}_i, y_i) sia classificato correttamente e che si trovi ad una distanza minima (il margine) dall'iperpiano di separazione.

Minimizza: $\|\mathbf{w}\|^2/2$ Questo equivale a massimizzare il margine

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 margine (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.

Gli alpha_i (moltiplicatori di Lagrange) pesano le penalizzazioni in caso di violazione dei vincoli (se rompo il vincolo, aumenta l'output della funzione obiettivo, ma la funzione deve essere minimizzata, quindi si tende a ridurre il numero di vincoli violati)

SVM lineari: Pattern Separabili (3)

Formalmente, le sommatorie nella funzione obiettivo duale e nella determinazione di w^* e b^* scorrono su tutti gli n punti di addestramento. Tuttavia, in pratica, queste sommatorie sono riconducibili solo ai Support Vectors.

(chiamato problema primale)

Questo perché solo gli alpha_i dei SV sono diversi da zero

Il problema di ottimizzazione precedente, può essere risolto passando innanzitutto a una ¹ formulazione Lagrangiana e successivamente a una ² formulazione duale.

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

n = numero di training point del TS

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

Vincolo per ogni training point
vincolo primale

Questi moltiplicatori quantificano "quanto è importante" un vincolo.

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

2 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 α_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 α_i :

1 Stazionarietà (Eliminazione di \mathbf{w} e b) → Si impongono le derivate parziali della funzione Lagrangiana $Q(\mathbf{w}, b, \alpha)$ rispetto alle variabili primali (\mathbf{w} e b) pari a zero.

Sostituzione e 2
Formulazione Duale: si eliminano le variabili \mathbf{w} e b , ottenendo la funzione $Q(\alpha)$ che dipende solo dai moltiplicatori alpha.

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

prodotto scalare riga-colonna

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

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

w* e b* sono i valori ottimi

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

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:

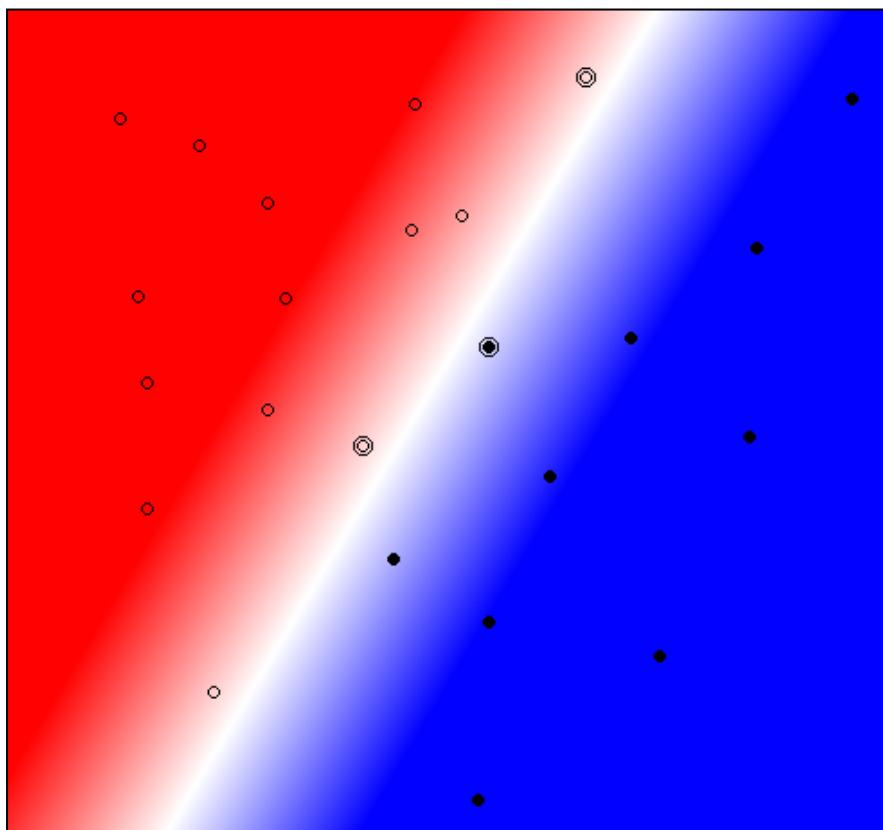
- Il **segno** della funzione $D(\mathbf{x})$ consente di classificare un generico pattern \mathbf{x} . (risultato binario, cioè o classe 1 o classe 2)
- Le sommatorie sono riducibili ai soli support vector.
- Nel caso lineare non è necessario, dopo aver calcolato \mathbf{w}^* e b^* , conservare/memorizzare i support vectors.

SVM lineari: Pattern Separabili (4)

Teoria dell'Apprendimento (Vapnik-Chervonenkis): La relazione $(n_{sv})/n$ (dove n è il numero totale di pattern in TS) non è casuale. Nella teoria statistica dell'apprendimento, si può dimostrare che l'errore atteso di generalizzazione (l'errore su dati mai visti) è limitato superiormente da una funzione della capacità del modello. Per le SVM, questa capacità (la dimensione VC) è direttamente correlata al numero di SV.

Vantaggi dell'approccio SVM:

- Definizione della soluzione sulla base di un numero ridotto di support vector (solitamente pochi).
Più è difficile separare le classi (cioè più i punti sono vicini al margine), maggiore sarà il numero di SV.
- 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 . "Curse of Dimensionality" Evitata: Molti algoritmi soffrono della CoD (l'aumento della complessità esponenziale con d). Le SVM, invece, ne sono in gran parte immuni, soprattutto quando si usa il Kernel Trick.



Esempio:

i support vectors
(cerchiati)
definiscono la
soluzione.

SVM lineari: Pattern non Separabili

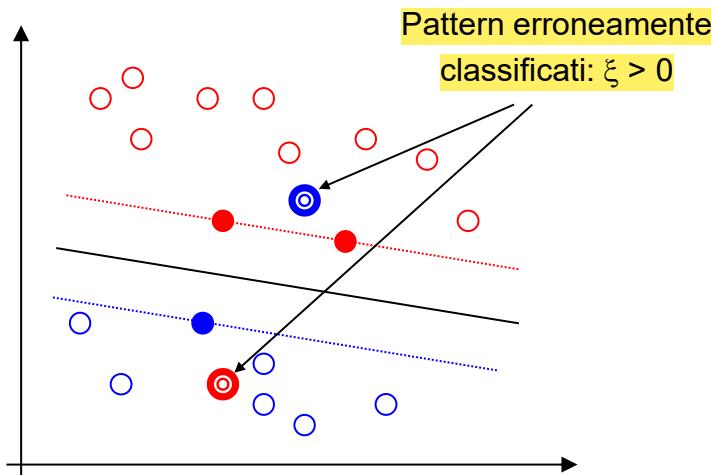
In questo caso **non tutti** i pattern possono essere separati da un iperpiano, ed è necessario rilassare i vincoli di separazione, per far sì che **alcuni** pattern (il minor numero possibile) possano valicare il confine della classe.

A tal fine si introducono n variabili di **slack** positive ξ_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 ξ_i codifica la deviazione dal margine. Per i pattern **separabili** del TS le corrispondenti variabili di slack assumeranno valore 0.

Con le variabili slack, posso "bypassare" il vincolo riguardante alla correttezza della classificazione di tutti i punti del TS



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:

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

parametro di regolarizzazione del numero di elementi erroneamente classificati

importanza relativa del rispetto dei vincoli in confronto al margine

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

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 α_i :

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

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

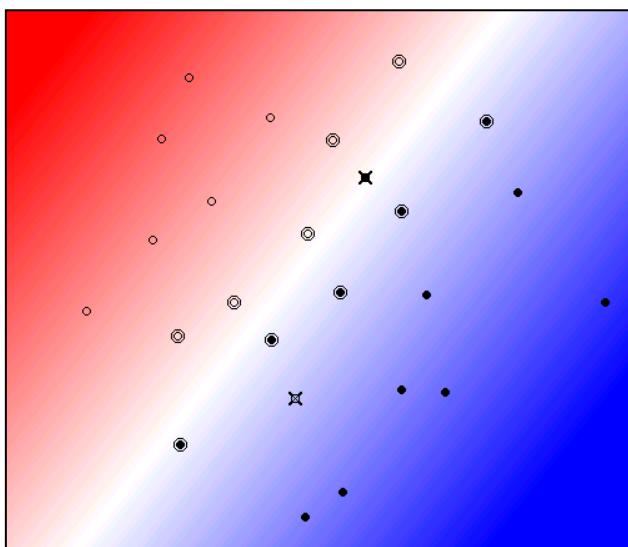
Le variabili slack durante le trasformazioni (forma lagrangiana/duale) vengono perse e si aggiunge il vincolo di C

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

Esempi:

$$\underline{C = 10}$$

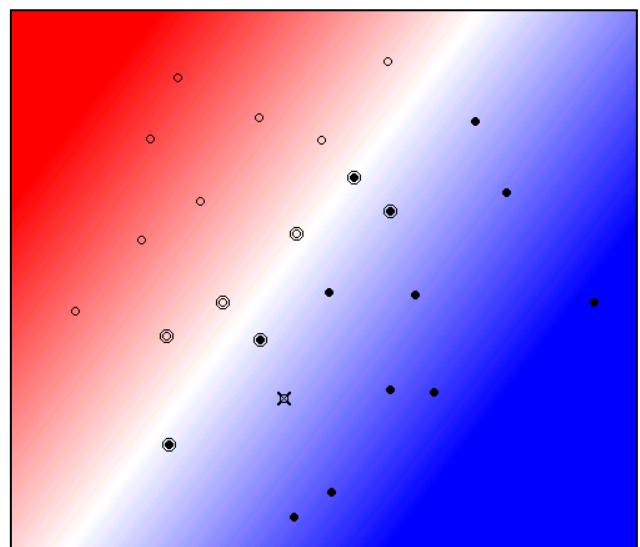
2 errori, margine maggiore



Maggior Overfitting

$$\underline{C = 200}$$

1 solo errore, margine minore



è più facile trovare un iperpiano lineare in uno spazio di dimensione più alta che una superficie curva in uno spazio di dimensione inferiore.

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* Φ non lineare dei pattern dallo spazio di partenza \mathcal{R}^d verso uno spazio \mathcal{R}^m a più alta dimensionalità ($m > d$): Questa trasformazione porta un'esplosione di dimensionalità (curse of dimensionality)

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

Trovare un iperpiano lineare nello spazio \mathcal{R}^m equivale a trovare una superficie di separazione arbitrariamente complessa nello spazio originale \mathcal{R}^d .

■ 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 Φ 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 .

Questo approccio ci permette di allenare un SVM non lineare con la stessa complessità computazionale di un SVM lineare, ma con la potenza discriminatoria della non linearità.

La Funzione Kernel è una funzione che calcola il prodotto scalare nello spazio ad alta dimensione senza dover mai calcolare esplicitamente le coordinate $\Phi(\mathbf{x})$ e $\Phi(\mathbf{x}')$.

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

Se dovessimo calcolare esplicitamente il mapping $\Phi(\mathbf{x})$ per ogni punto e poi applicare l'SVM standard, l'operazione sarebbe insostenibile perché: Il calcolo esplicito di $\Phi(\mathbf{x})$ in uno spazio ad alta dimensione è computazionalmente proibitivo. Inoltre, La complessità del training ($O(n^2 * m)$) aumenterebbe enormemente.

SVM Non lineari: Kernel functions

In m, ho un iperpiano. In d, ho un ipersuperficie (una figura complessa). Ciò avviene retropropagando da m in d

■ 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 $d = 2$, $q = 2$:

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

per un grado q , include tutte le possibili combinazioni di monomi fino al grado q . e quindi $m = 9$.

Si dimostra che:

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

Aumentando q , aumenta la complessità della superficie di separazione e il rischio di overfitting.

■ Radial Basis Function (RBF) di ampiezza σ (iperparametro):

Questo valore misura la somiglianza tra \mathbf{x} e \mathbf{x}' . È massimo (vicino a 1) se i due punti sono vicini e si avvicina a 0 se sono lontani.

Mapping in uno spazio di feature di dimensionalità infinita

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

sigma = ampiezza gaussiana

■ Sigmoid (meno utilizzato):

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

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

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

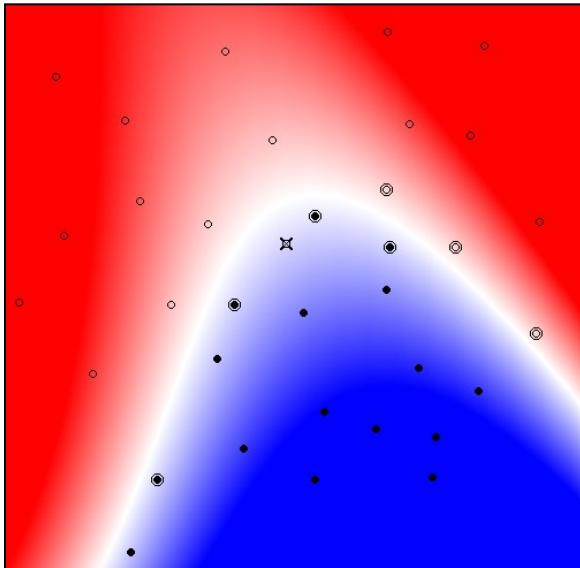
$$D(\mathbf{x}) = \sum_{i=1 \dots n} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^*$$

SVM non ritorna probabilità, ma $D(X)$ (distanza).
Ci sono tecniche di calibrazione che usano parte del dataset per convertire distanza in probabilità

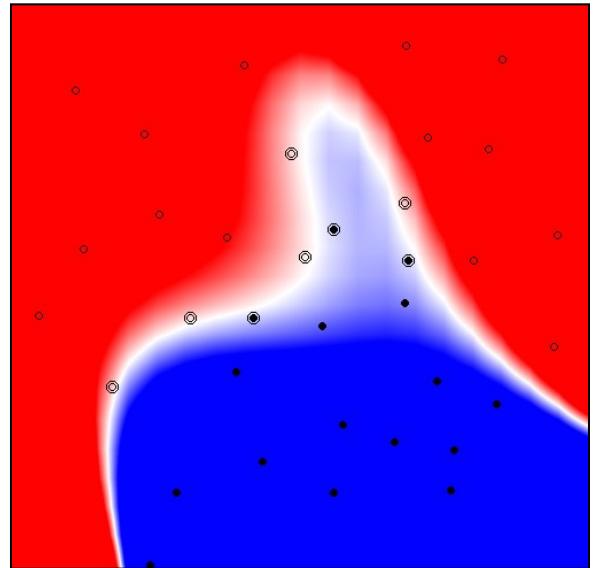
SVM Non lineari: Esempi

RBF sono più regolari, stabili numericamente e più efficaci nell'isolare le classi tra loro rispetto a Kernel Polinomiale

Polinomio $q = 2$



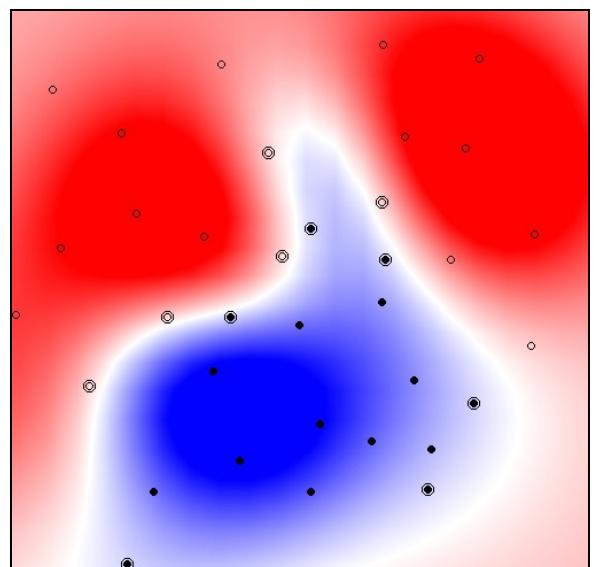
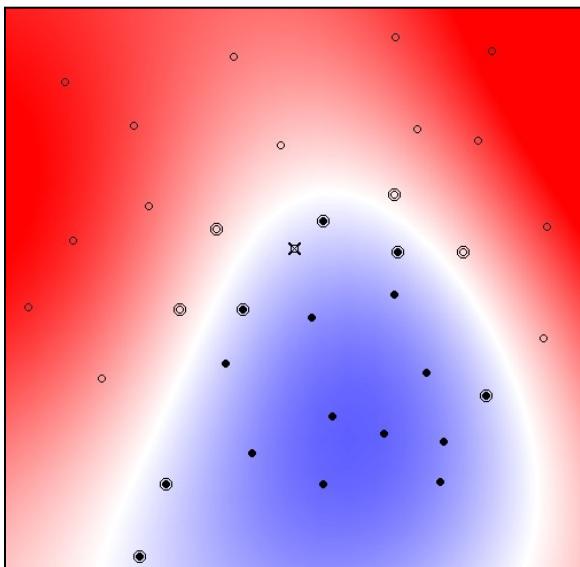
Polinomio $q = 10$



RBF $\sigma = 1$

gamma e sigma sono inversamente proporzionali
 $\text{gamma} = 1/2 * (\sigma^2)$

RBF $\sigma = 0.2$



sigma Grande -> gamma Piccolo: Se sigma è grande, la campana Gaussiana è larga (ampia). Il raggio di influenza di un Support Vector è esteso. Questo porta a confini di decisione più lisci (tendenza all'Underfitting).

sigma Piccolo -> gamma Grande: Se sigma è piccolo, la campana Gaussiana è stretta (concentrata). Il raggio di influenza è limitato. Questo porta a confini di decisione molto fratturati e sensibili ai singoli punti (tendenza all'Overfitting).

SVM: estensione multclasse

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. Ogni combinazione Classe-Classe con tutte le classi (1vs1 fatto per tutte le classi)
Si usa OvO quando il numero di classi è piccolo ($k^*(k-1)$ SVM)

- **One-Against-All:**

k = classe
 n = num classi

- 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 (γ) per regolare la complessità della superficie decisionale. Aumentando γ la superficie può assumere forme più complesse.
 - *N.B. Con kernel RBF γ opera in modo inverso rispetto a σ .*
 - *Inserito γ 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

→ Più ottimizzata rispetto a LIBSVM per quanto riguarda il training di SVM lineari

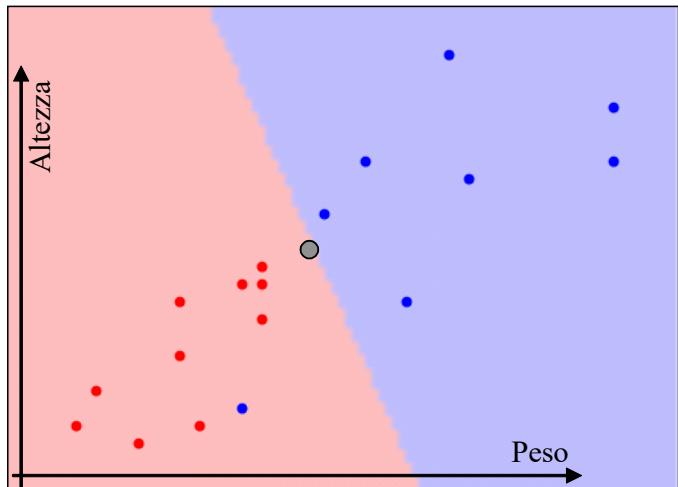
■ LIBLINEAR - <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>

- Stessi autori di LIBSVM, consigliata nel caso lineare per elevata dimensionalità ed elevato numero di pattern.
- Wrapped da Scikit-Learn → `sklearn.svm.LinearSVC`

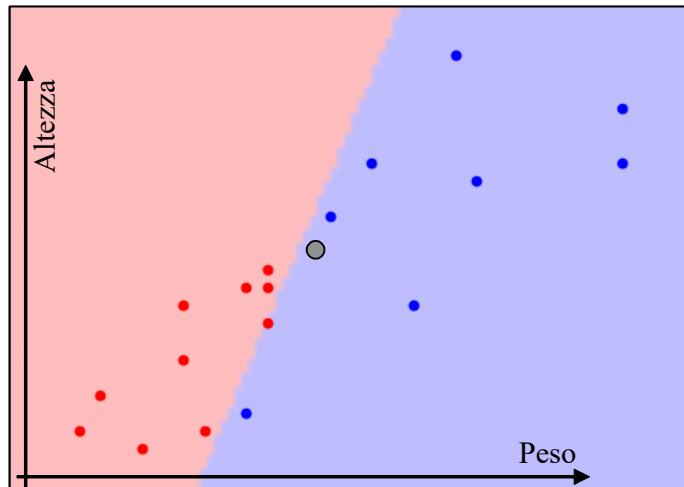
■ SVM-light - <http://svmlight.joachims.org>

Esempi LibSvm (1)

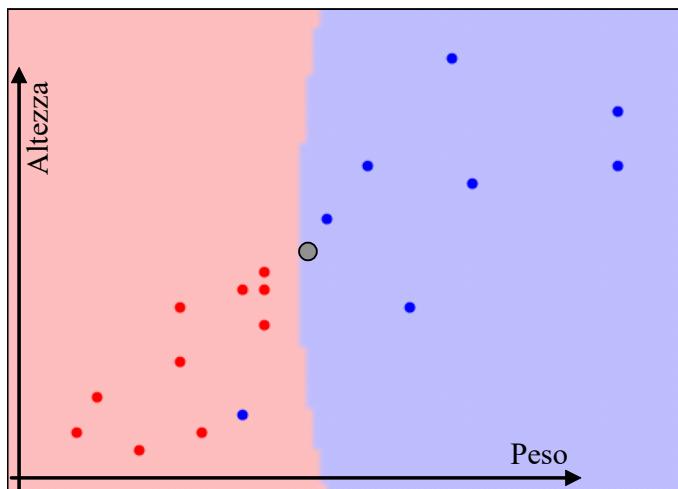
«maschi-femmine»



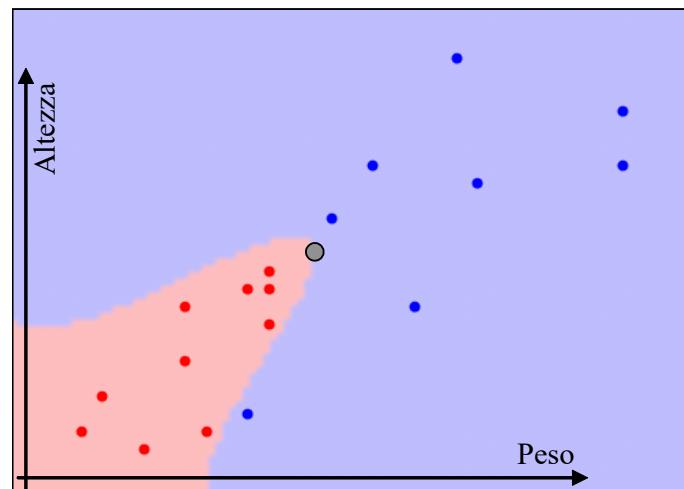
Lineare, $C = 10$



Lineare, $C = 500$



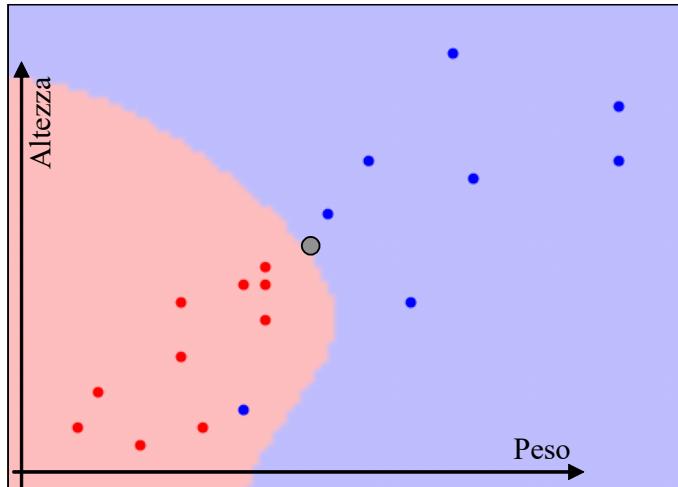
Polinomio $q = 3, C = 10$



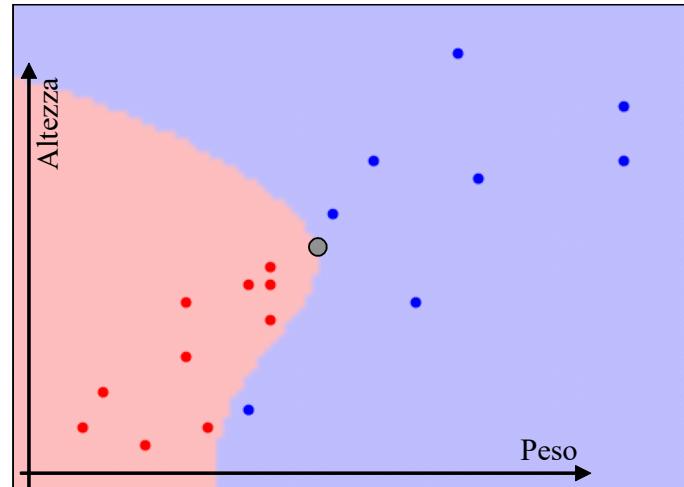
Polinomio $q = 3, C = 500$

Esempi LibSvm (2)

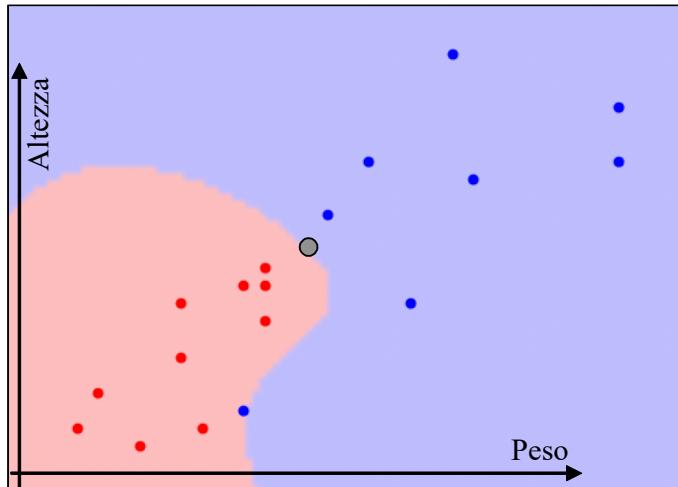
«maschi-femmine»



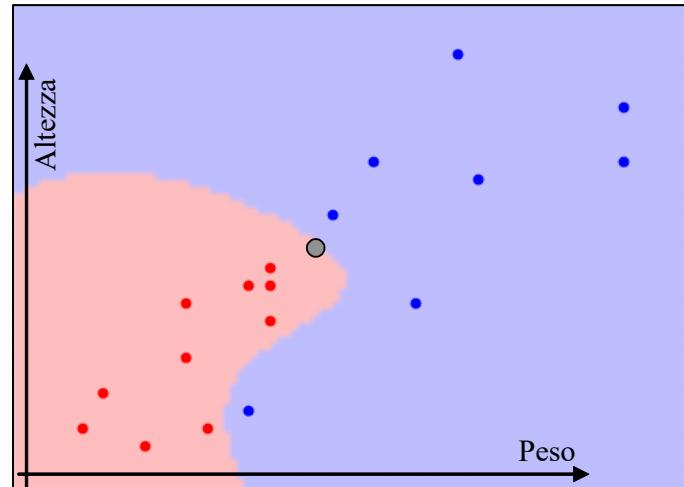
RBF, $\gamma = 5, C = 10$



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



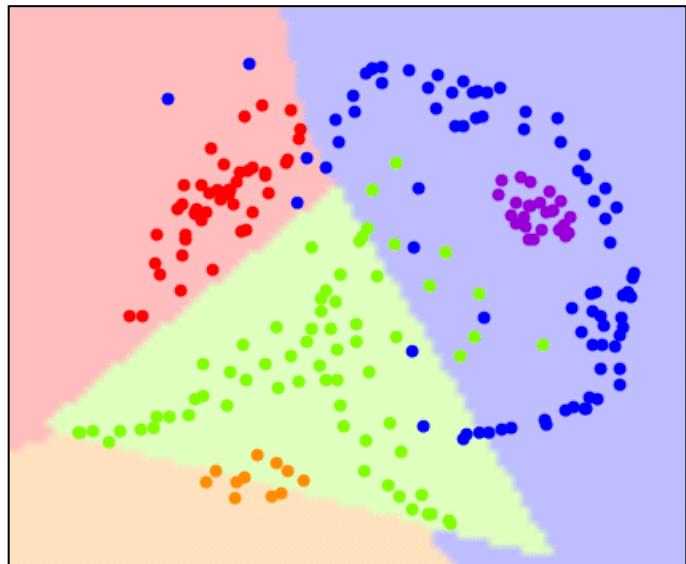
RBF, $\gamma = 10, C = 10$



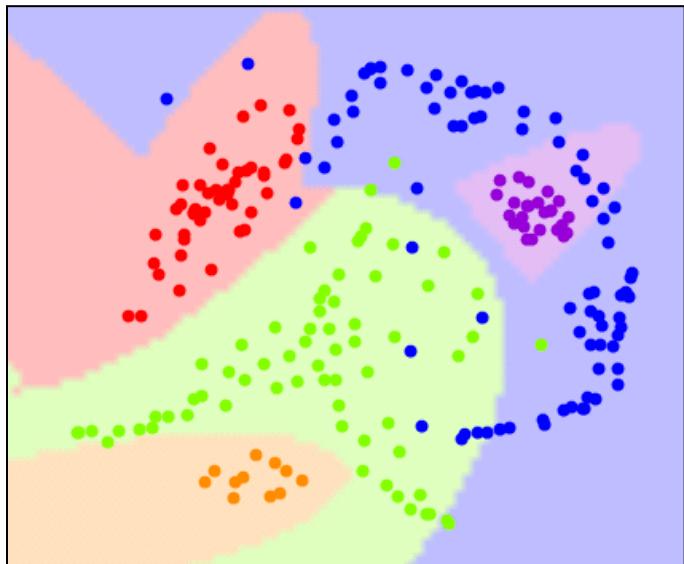
RBF, $\gamma = 10, C = 500$

Esempi LibSvm (3)

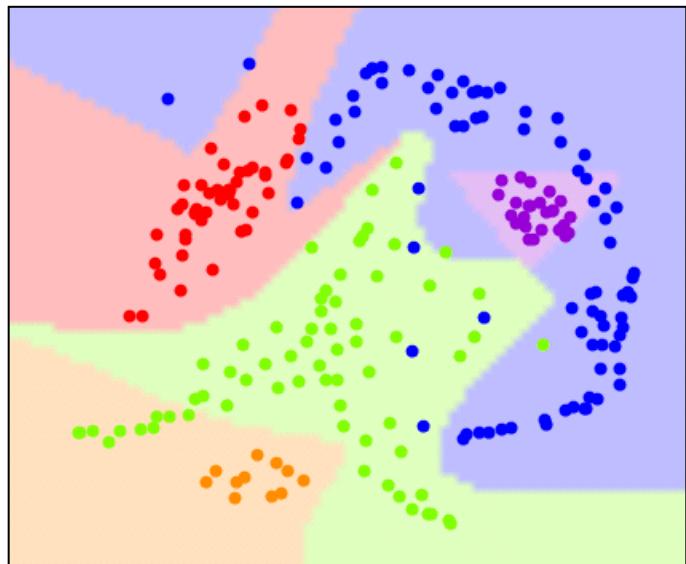
multiclasse



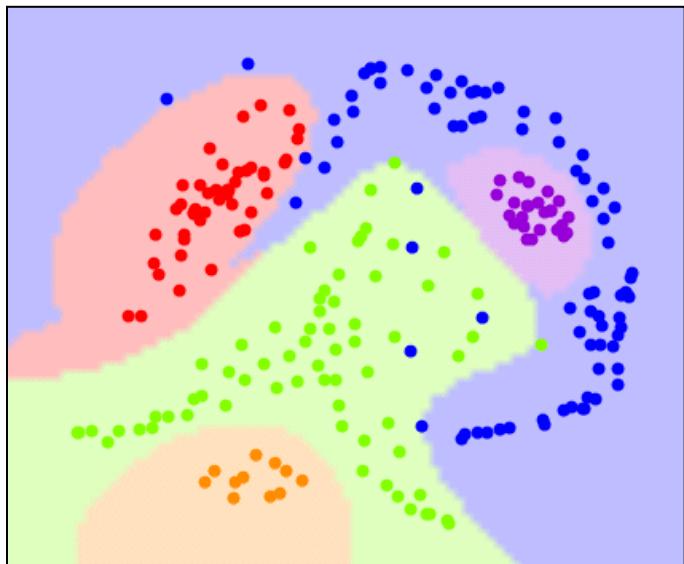
Lineare, $C = 100$



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?

↳ Usare SVM Lineare, perché diventa computazionalmente proibitivo calcolare il Kernel

- 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 .

↳ Possibilità di prendersi maggiori gradi di libertà

- per bassa dimensionalità (es. 20 feature) la scelta primaria è **SVM non lineare** con **kernel RBF**. Gli iperparametri da tarare sono C e σ (o γ 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. (scelte prese su fattori diversi/indipendenti)
- L'**indipendenza** (o **diversità**) è normalmente ottenuta: (avere indipendenza totale è difficile. Va bene anche parziale)
 - Utilizzando **feature diverse** (non correlate o poco correlate) i classificatori sono dello stesso tipo
 - Utilizzando **algoritmi diversi** per l'estrazione delle **feature** stesse feature su algo diversi
 - Utilizzando **diversi algoritmi di classificazione** stesse feature su classificatori di diverso tipo
 - Addestrando lo **stesso algoritmo** di classificazione su **porzioni diverse** del training set (**bagging**) Stesso Algo, Sottoinsieme di Training Set (la scelta del sottoinsieme può essere a re-introduzione)
 - Insistendo con l'addestramento sugli **errori** commessi dai predecessori (**boosting**) Stesso Algo, Addestro nuovo classificatore per ridurre gli errori del precedente Raffinamento incrementale su zone critiche (vicino a decision boundary): significa che se un pattern difficilmente viene classificato correttamente il prossimo classificatore dell'ensemble puntare a risolvere tale problema
- La combinazione può essere eseguita¹ a livello di decisione o² a livello di confidenza.

1 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 **combinate** 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:

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

Vince la classe che ha la maggioranza, quindi il nuovo pattern viene assegnato a quella classe vincente

In base al ranking di ogni classificatore, ogni classe riceve dei punti in base al rank e poi i punti, provenienti da tutti i ranking, vengono sommati per creare una classifica finale. Viene scelta la classe con più punti nella classifica finale, che serve per classificare il pattern.

- **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. cioè non considera solo i primi di ogni classificatore, ma considera tutta la classifica 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:

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

la sommatoria va da $(nc+1)/2$ a nc , cioè vengono considerati solo i casi in cui la metà_intera+1 dei classificatori classifica correttamente

Ricorda, la distribuzione binomiale permette di calcolare con quale probabilità estraiamo m palline rosse da un'urna contenente 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.

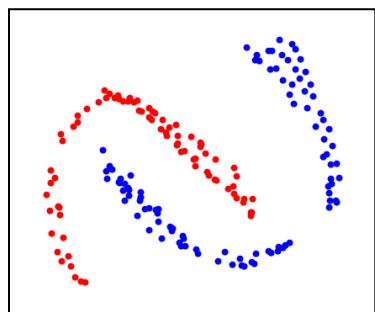
- **Esempio:** se $P = 0.8$ (ogni classificatore commette il 20% di errore di classificazione) stando alla formula precedente per un multi-classificatore con nc (variabile) componenti:

- ▶ $P_{multi}(3) = 0.896$ ($k = 2$)
 - ▶ $P_{multi}(5) = 0.942$ ($k = 3$)
 - ▶ $P_{multi}(7) = 0.966$ ($k = 4$)
 - ▶ $P_{multi}(9) = 0.980$ ($k = 5$)
 - ▶ $P_{multi}(15) = 0.995$ ($k = 8$)
 - ▶ $P_{multi}(21) = 0.999$ ($k = 11$)
- ES: metà_intera + 1 := 21/2=10.5 -> (prendendo il floor ho 10) -> 10+1 -> k = 11
- Eccezionale !!!

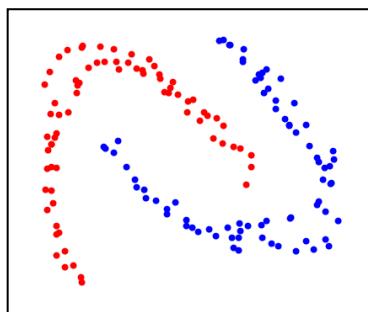
Dov'è il trucco? La formula precedente è basata sull'**indipendenza (totale) dei classificatori**, cosa difficilmente ottenibile in pratica!

Infatti, nella pratica, si punta ad avere almeno un'indipendenza parziale, il più possibile vicina all'indipendenza totale

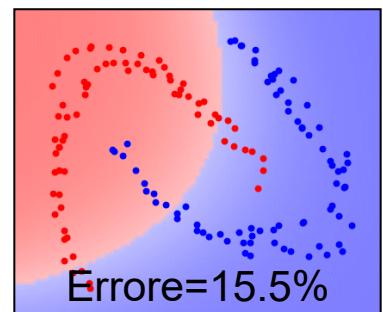
Majority vote rule (esempio)



Training set



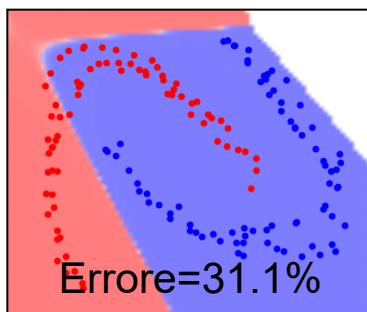
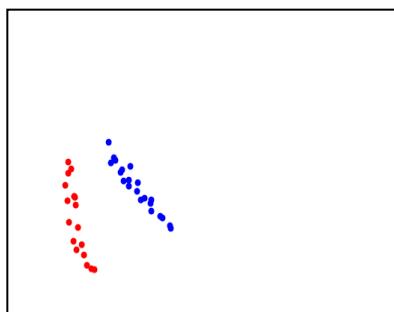
Test set



(Bayes – Distr. Normali)

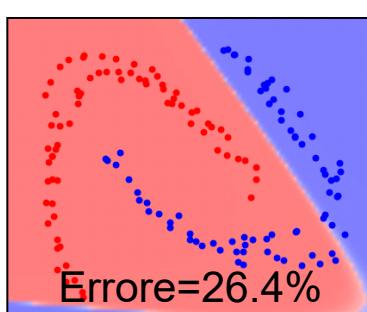
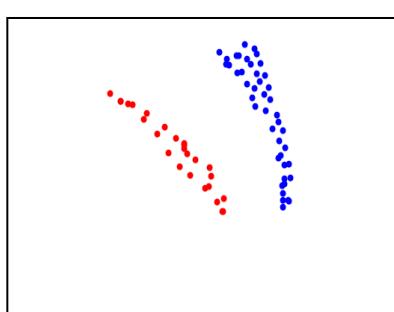
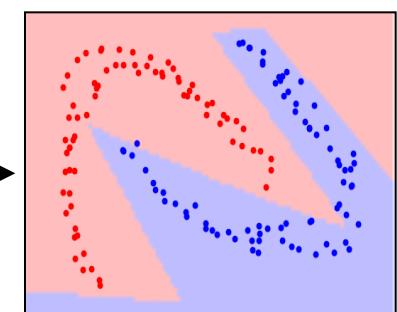
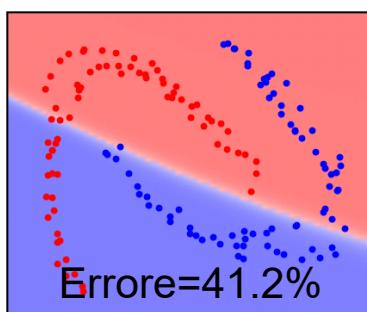
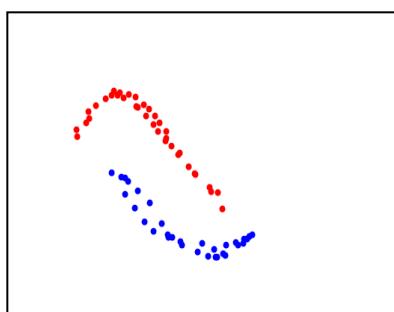
Tipo di Classificatore

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



Quindi, ho 3 classificatori Bayes allenati su diversi sottoinsieme del Training Set

Majority vote rule:



Singolarmente, hanno un errore molto alto. Combinandoli, si ottiene un errore molto basso. Questo perché i tre classificatori sono indipendenti: lavorano su 3 sottoinsieme diversi

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. (Tutte le classi si scontrano con tutte)
- 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.

anche se il numero di pattern considerati nel Training Set sono minori rispetto al OvR (vengono considerati solo i pattern delle due classi)

2 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 $\text{conf}_j = [conf_{j1}, conf_{j2} \dots conf_{js}]$ in cui l'i-esimo elemento indica il grado di appartenenza del pattern alla classe i-esima. (La somma delle probabilità dei vettori conf_j deve essere uguale ad 1)

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

Nel 90/95% dei casi, si usa la Somma

$$\text{sum} = \sum_{j=1..nc} \text{conf}_j \rightarrow t = \arg \max_{i=1..s} \{ sum_i \}$$

Vince la classe con il valore più alto
(si può normalizzare la probabilità sommata di ciascuna classe oppure lasciarla così com'è dopo la somma)

$$\text{prod} = \prod_{j=1..nc} \text{conf}_j \rightarrow t = \arg \max_{i=1..s} \{ prod_i \}$$

Vince la classe con il prodotto delle confidenze più alto (é più corretto della somma, se indipendente (teoria))

$$max_i = \max_{j=1..nc} \{ conf_{ji} \} \rightarrow t = \arg \max_{i=1..s} \{ max_i \}$$

Vince la classe che ha la probabilità più alta (massimo tra le probabilità di ciascun classificatore per quella classe)

$$min_i = \min_{j=1..nc} \{ conf_{ji} \} \rightarrow t = \arg \max_{i=1..s} \{ min_i \}$$

Vince la classe che ha la probabilità più bassa (minimo tra le probabilità di ciascun classificatore per quella classe)

- 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.

La somma è meno sensibile ad outlier, rispetto al prodotto

2 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 .

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

proportionale alle performance del modello j

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).

Per definire i pesi g_j , il training set deve essere abbastanza grande e rappresentativo (per evitare overfitting)

- 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**.

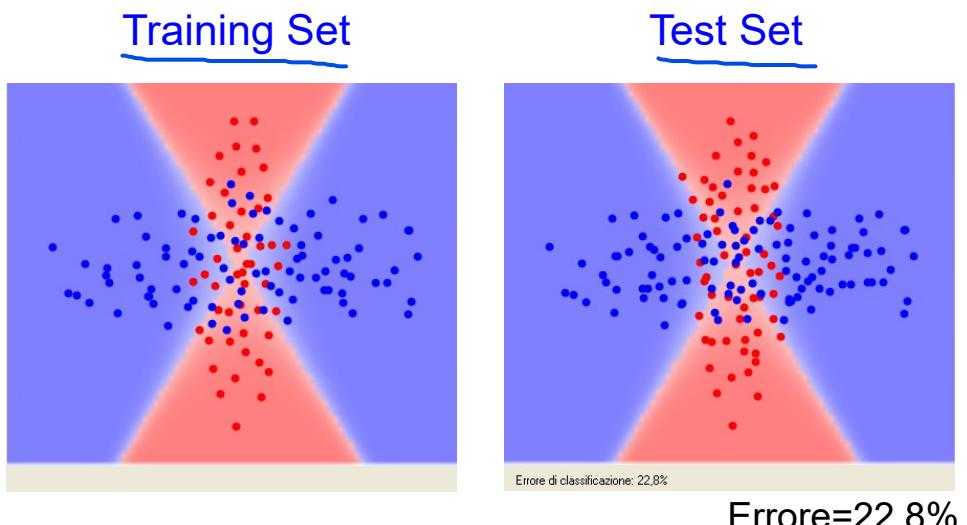
→ Come unire le distanze con le probabilità?

In alcuni modelli, che mostrano in output delle distanze, usano un dataset di calibrazione per calibrare la conversione delle distanze a probabilità (una distanza maggiore dal decisione boundary implica una probabilità/confidenza maggiore nella classe predetta)

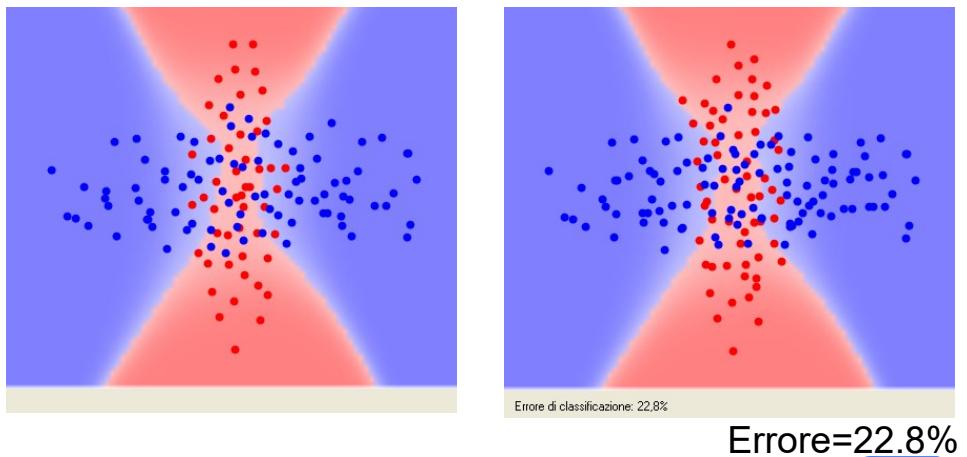
Importante: Il dataset di calibrazione deve essere diverso dai dati di addestramento e di test. Viene usato per "addestrare" il meccanismo di conversione/calibrazione dopo che i modelli di base sono stati addestrati.

Fusione a livello di confidenza: esempio

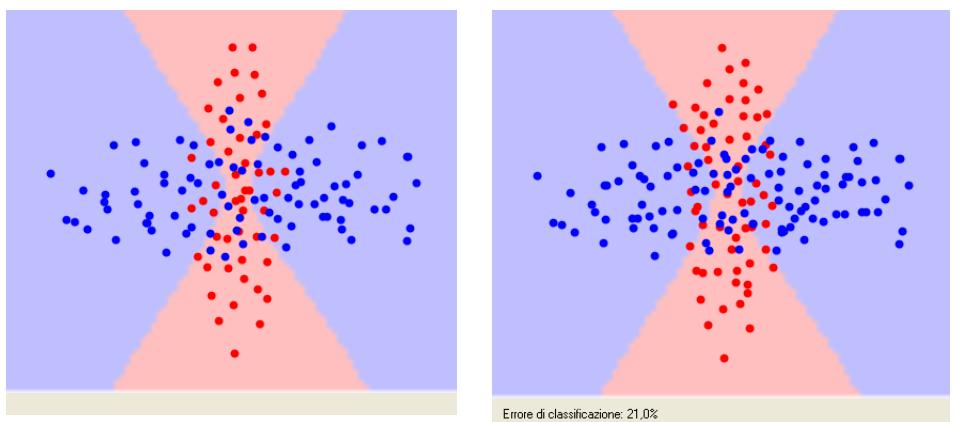
Bayes parametrico
(normali)



Bayes & Parzen
(normali h=10)



**Fusione a livello di
confidenza
(somma)**



I decision boundary dell'Ensemble, cambiano
al cambiare della tecnica di fusione delle
classificazioni dei modelli singoli

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):
(In Random Forest, i singoli classification tree vengono specializzati in sottoinsiemi del TS: indipendenza parziale)
 - 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**.

interno alla foglia (majority voting dei pattern del TS all'interno di quella foglia)

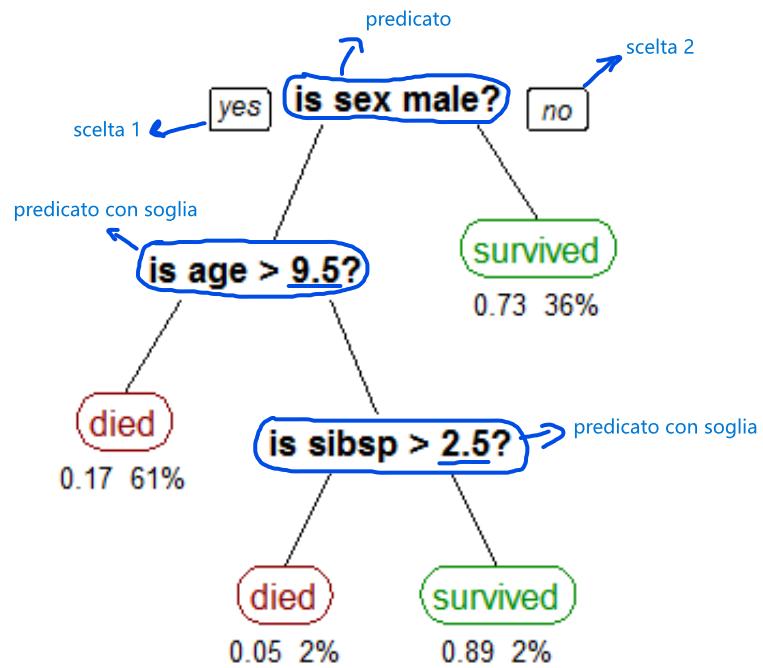
Decision Tree è interpretabile/spiegabile. Le variabili categoriche non vengono decodificate.

Come vengono scelti i predicatori? Si sceglie l'attributo che separa di più le classi (il più discriminante) per creare due nodi, nati dal predikato, il più puri possibili

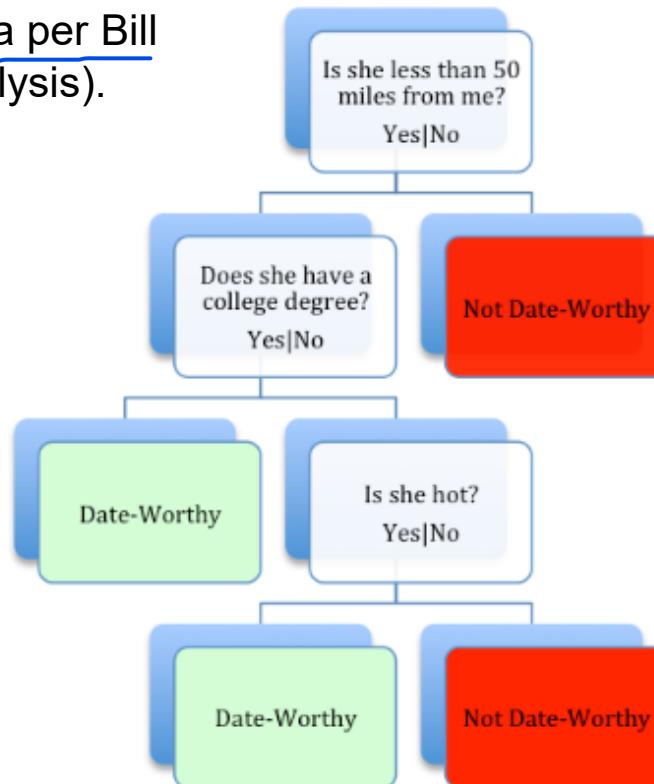
Ogni nodo interno (no foglia) al classification tree discrimina il dataset arrivato in quel punto dell'albero

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 $d' = \sqrt{d}$ (cioè, ogni Classification Tree, utilizza un sottoinsieme delle feature per creare i predici)
- In assenza di questo accorgimento (noto anche come **feature bagging**) molti tree sceglierrebbero con elevata probabilità le stesse variabili (quelle più discriminanti). (quindi tramite feature bagging, i singoli alberi diventano più indipendenti)
- Pertanto Random Forest opera simultaneamente due tipi di bagging: uno sui pattern del training set e uno sulle features.

→ Aumentando il numero di tree in Random Forest, aumenta le performance nel TS, ma non è detto che aumenti sul Validation
■ 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).

→ Senza un `max_depth`, i singoli alberi arrivano fino a massima purità (cioè in ogni foglia, ci sono solo quelle classi). Ciò porta ad overfitting (impara a memoria il Training Set)
■ 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. (OOB è utile solo in casi in cui il TS è molto piccolo e ciò non permette di avere un unico val test per tutti perché ridurrebbe il TS. Di solito, si utilizza Cross Validation e non si usa OOB)

■ 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.

(Essendo che ogni albero riceve i 2/3 del Training Set, il validation set di ogni albero è il completare)

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 multclasse.
- 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
 - α^t il peso del classificatore $h^t(\mathbf{x})$ nell'ambito del multi-classificatore. (Hanno maggior peso i classificatori con alte performance)

AdaBoost: pseudocodice

$$w_i^1 = \frac{1}{n}, \quad i = 1..n$$

All'inizio

(tutti i pattern hanno lo stesso peso iniziale)

For $t = 1 \dots T$

Andando avanti con le iterazioni t , i pesi cambiano (quelli con maggior peso sono i pattern difficili da predire/classificare)

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

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

Calcola errore (pesato sull'importanza dei pattern)

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

funzione indicatrice

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

Alla fine, sommo tutti gli errori compiuti da quel classificatore e quello con epsilon minore (ϵ_j) viene scelto come classificatore di quella iterazione t

(cond = true) significa che c'è errore nella classificazione del pattern quindi avviene il prodotto con il peso

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

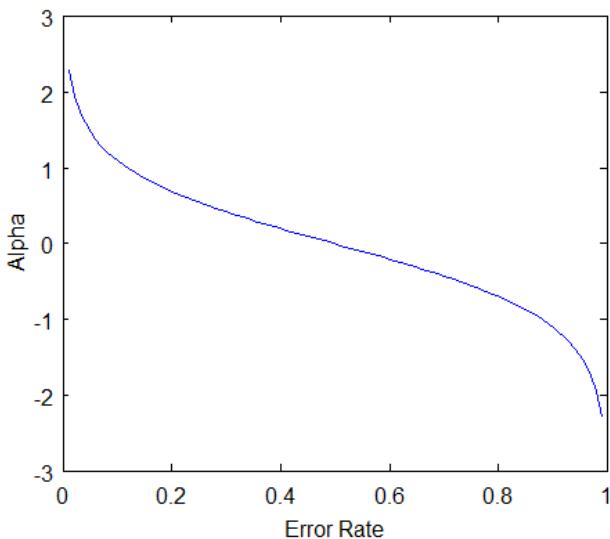
Al termine dell'addestramento, il **classificatore finale (strong)** è la media pesata dei weak classifiers:

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

AdaBoost: scelta dei pesi

Peso del classificatore
scelto all'iterazione t

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



Classificatore Random

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!).

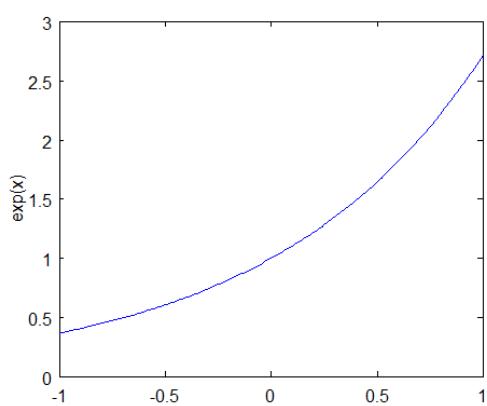
inverte la
decisione
usando α^t

applicabile solo in classificazione binaria
(per multi-classe, dipende dal valore
intermedio, che in questo caso è 0.5)

$$w_i^{t+1} = w_i^t \cdot e^z \text{ dove}$$

$$z = -\alpha^t y_i h^t(\mathbf{x}_i)$$

Prodotto etichetta
vera con predetta



Quindi, i pattern
critici influenzano la
scelta dei
classificatori
successivi

osservando che:

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

("di solito classifica bene")

- w_i^{t+1} aumenta se $z > 0$: α^t positivo e classificazione errata, oppure α^t negativo e classificazione corretta ("di solito classifica male")
- w_i^{t+1} diminuisce se $z < 0$: nei 2 casi complementari.

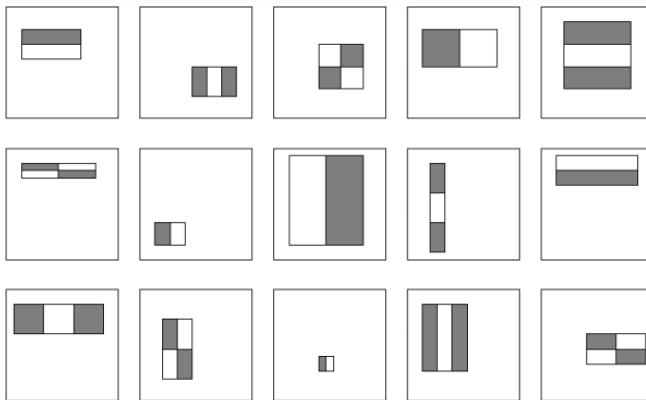
Viola & Jones Face detector (2001)

Queste features sono progettate per catturare contrasti caratteristici che si trovano spesso nei volti umani

- 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.

Haar Features: Sono filtri rettangolari semplici che calcolano la differenza tra la somma dei valori dei pixel in due o più regioni rettangolari adiacenti di un'immagine.



soglia per h_j

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

- 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. Un volto viene erroneamente classificato come non-volto e scartato.
 - 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 è può essere facilmente aggiunta al toolkit dei modelli da considerare per la risoluzione di un problema. Vedi [link](#).