Corso di Bioinformatica

Algoritmi di classificazione supervisionati

Giorgio Valentini

DI – Università degli Studi di Milano

Metodi di apprendimento supervisionato per problemi di biologia computazionale

Problemi biologia computazionale

- Predizione del fenotipo
- Predizione di classi funzionali di geni
- Diagnosi dei tumori
- Predizione della risposta a terapie
- Identificazione di target per i farmaci
- Identificazione di geni correlati a malattie
- ...

Metodi supervisionati

- Classificatori Nearest-Neighbours
- Classificatori Naïve-Bayes
- Discriminanti lineari
- Alberi di decisione
- Percettroni multistrato
- Support Vector Machine
- Metodi basati su kernel
- Metodi di ensemble
- ...

Esempio: supporto alla diagnosi come problema di apprendimento automatico (1)

I dati generati da bio-tecnologie high-throughput (ad es: DNA microarray) sono rappresentabili come insiemi di coppie (**x**,t):

- $\mathbf{x} \in R^d$, $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ rappresenta i livelli di espressione genica di *d* geni
- t rappresenta un particolare stato funzionale

Es: $t \in C = \{ s, m \}$, s → paziente sano, m→malato

Esempio: supporto alla diagnosi come problema di apprendimento automatico (2)

Obiettivo dell'apprendimento automatico:

Apprendere la funzione non nota f :
$f : R^d \rightarrow \mathrm{C}$ che mappa i livelli di espressione genica $\mathbf{x} \in R^d$ nella corrispondente classe funzionale $\mathrm{t} \in \mathrm{C}$ (es: paziente sano o malato)

tramite un algoritmo di apprendimento (learning machine) L che utilizza solo un training set $D = \{(\mathbf{x}_i, t_i)\}_{i=1}^{n}$ di campioni distribuiti in accordo alla distribuzione di probabilità congiunta P(**x**,t).

Algoritmi di apprendimento supervisionato e learning machine

• L' algoritmo di apprendimento *L* genera un' approssimazione $g : R^d \rightarrow$ C della funzione non nota *f* utilizzando il training set *D*: $L(D) \rightarrow g$.

• Si desidera che tale funzione sia la più "vicina" possibile ad *f*

• A tal fine si usa una funzione di perdita Loss(*f*(**x**),g(**x**)) che misuri quanto *g* differisca da *f*.

• Nei problemi di classificazione si usa la funzione di perdita 0/1:

$$Loss\big(g(\mathbf{x}), f(\mathbf{x})\big) = \begin{cases} 1 & se \quad g(\mathbf{x}) \neq f(\mathbf{x}) \\ 0 & se \quad g(\mathbf{x}) = f(\mathbf{x}) \end{cases}$$

Ma f non è nota (se lo fosse avremmo risolto il problema) ...

Addestramento delle learning machine

- Nella realtà si dispone spesso solo di un insieme relativamente limitato di dati (ad es: un insieme *D* di dati di espressione genica) e la learning machine viene addestrata ad approssimare *f* utilizzando tali dati come una serie di esempi da apprendere:

$(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, ... $(\mathbf{x}_n,t_n)$.

- La learning machine verrà addestrata ad apprendere una funzione *g* tale che $g(\mathbf{x}_1)=t_1$, $g(\mathbf{x}_2)=t_2$, ..., $g(\mathbf{x}_n)=t_n$, in modo da minimizzare il rischio empirico R_{emp} rispetto al training set $D = \{(\mathbf{x}_i, t_i)\}_{i=1}^{n}$:

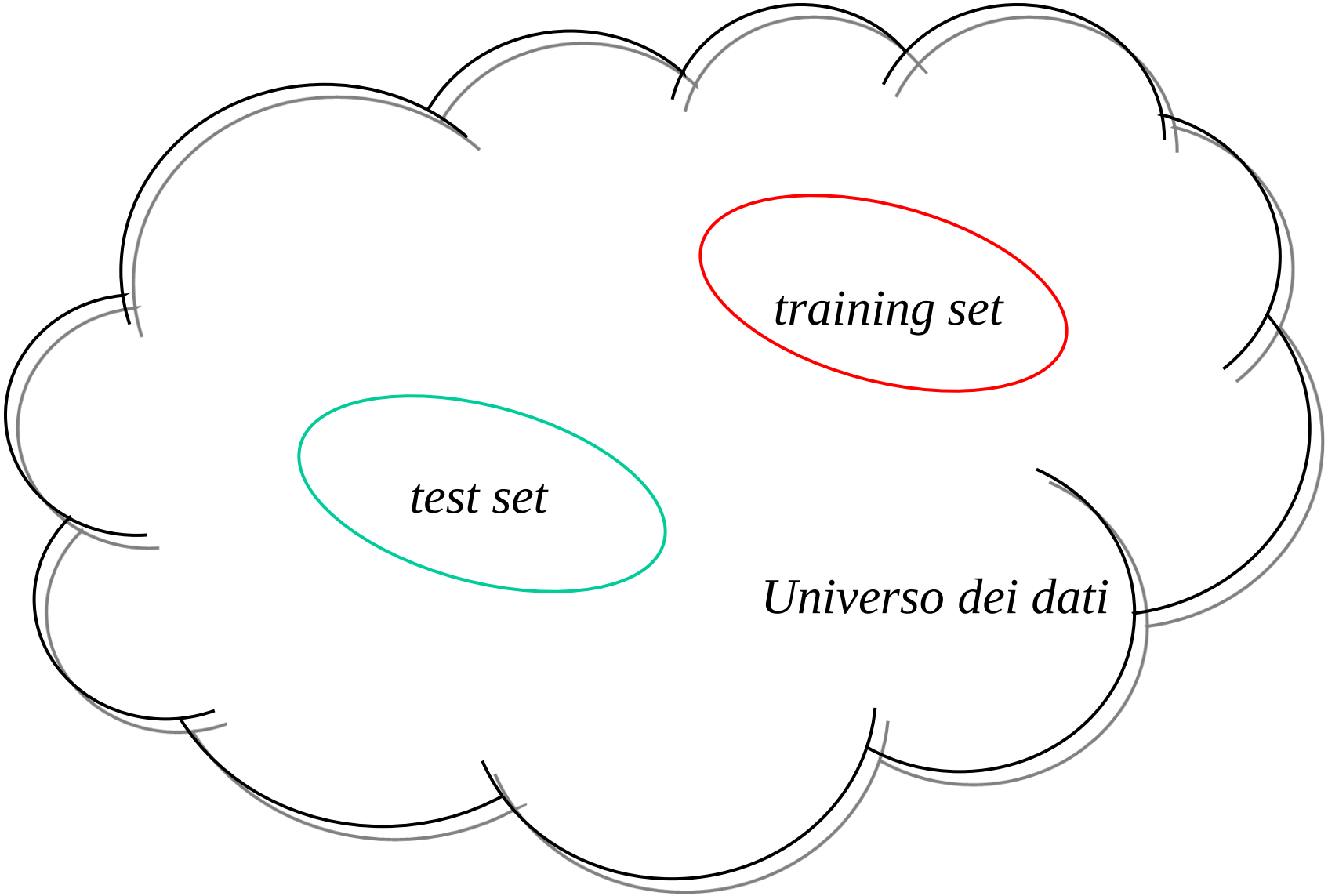
$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} Loss\left(g(\mathbf{x}_i), t_i\right)$$

Generalizzazione

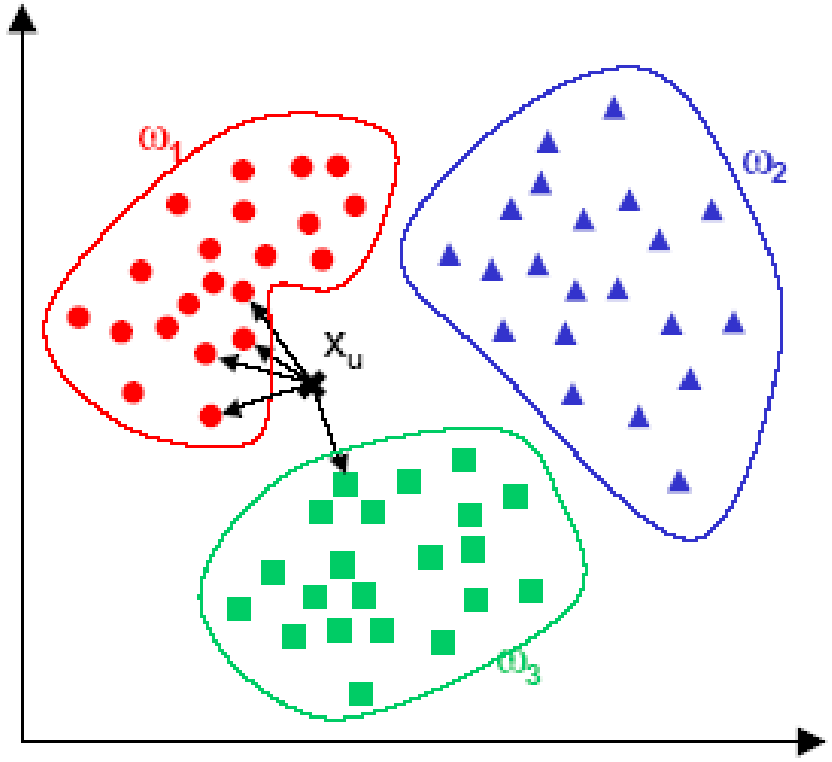
- L viene addestrata su un *training set* $D \subset U$.
- La learning machine L è utile se può fare delle previsioni sull' Universo non noto U dei dati:

  vogliamo cioè che *generalizzi* correttamente su dati che non conosce.
- A questo fine L deve prevedere correttamente non tanto i dati D su cui è stata addestrata, ma i dati $(\mathbf{x},t) \in U$, $(\mathbf{x},t) \notin D$ che non conosce.
- Siccome di solito non si conosce a priori U o equivalentemente la distribuzione di probabilità congiunta $P_U(\mathbf{x},t)$, le capacità di generalizzazione di L vengono valutate utilizzando metodi sperimentali per la stima dell'errore di generalizzazione.

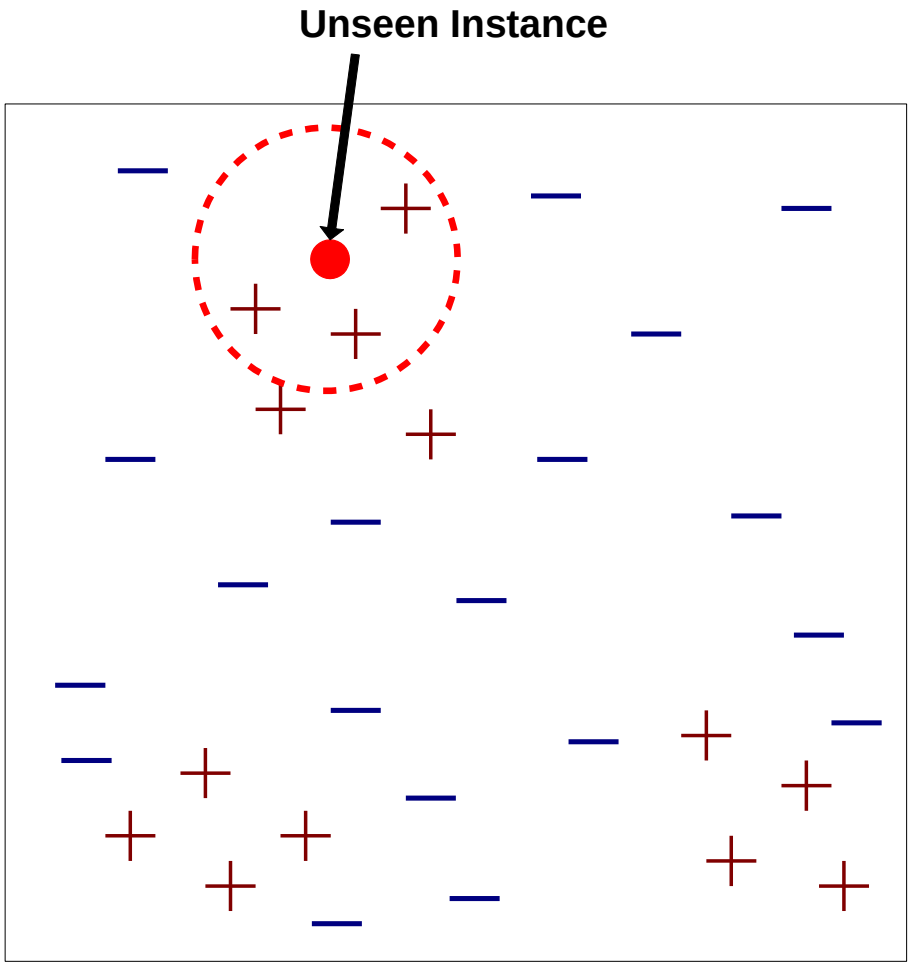
Universo dei dati e campioni



K-nearest-neighbour



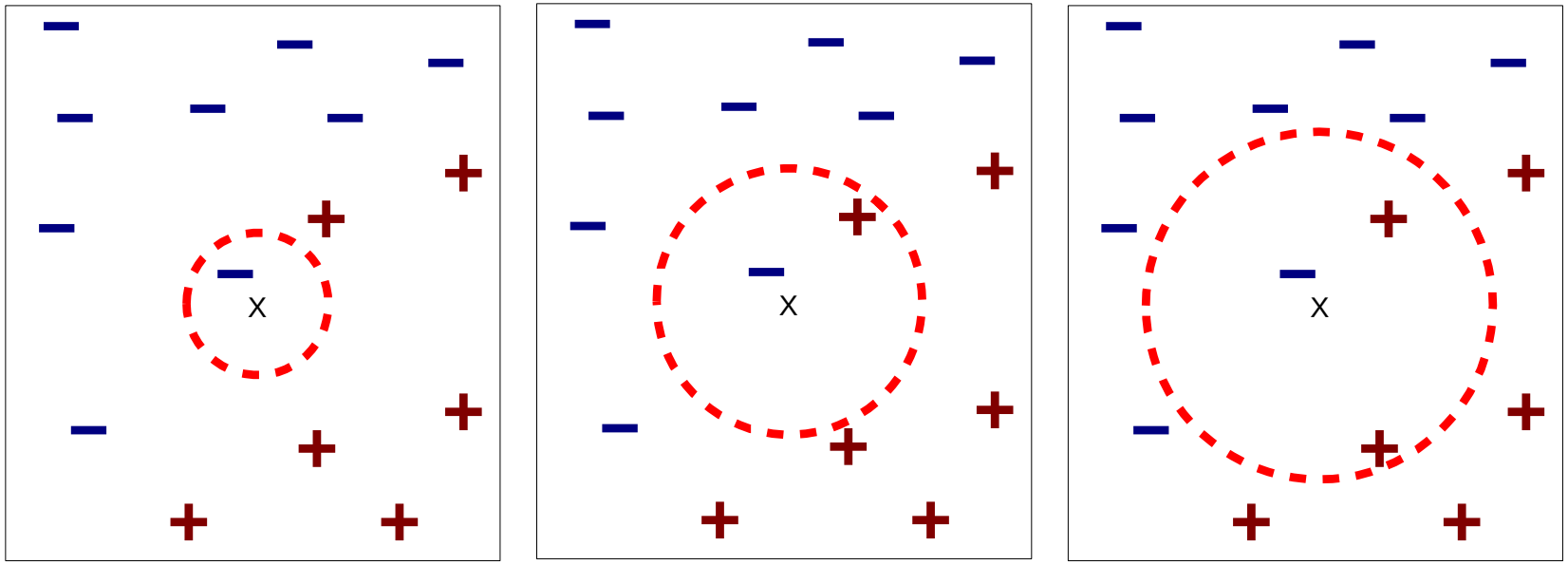
Classificatore Nearest-Neighbor



- L' *apprendimento* richiede:
  - Un insieme di pattern (**x**,*t*) di dati di espressione genica (training set)
  - Una metrica per calcolare le distanze fra pattern
  - Il valore di k, cioè del numero dei primi k vicini.

- La *classificazione* di un pattern **x** la cui classe *t* di appartenenza non è nota richiede :
  - La ricerca dei primi *k* vicini a **x**
  - Assegnamento ad **x** della classe *t* maggiormente frequente nei primi *k* vicini (voto a maggioranza)

Esempio di k-Nearest Neighbor

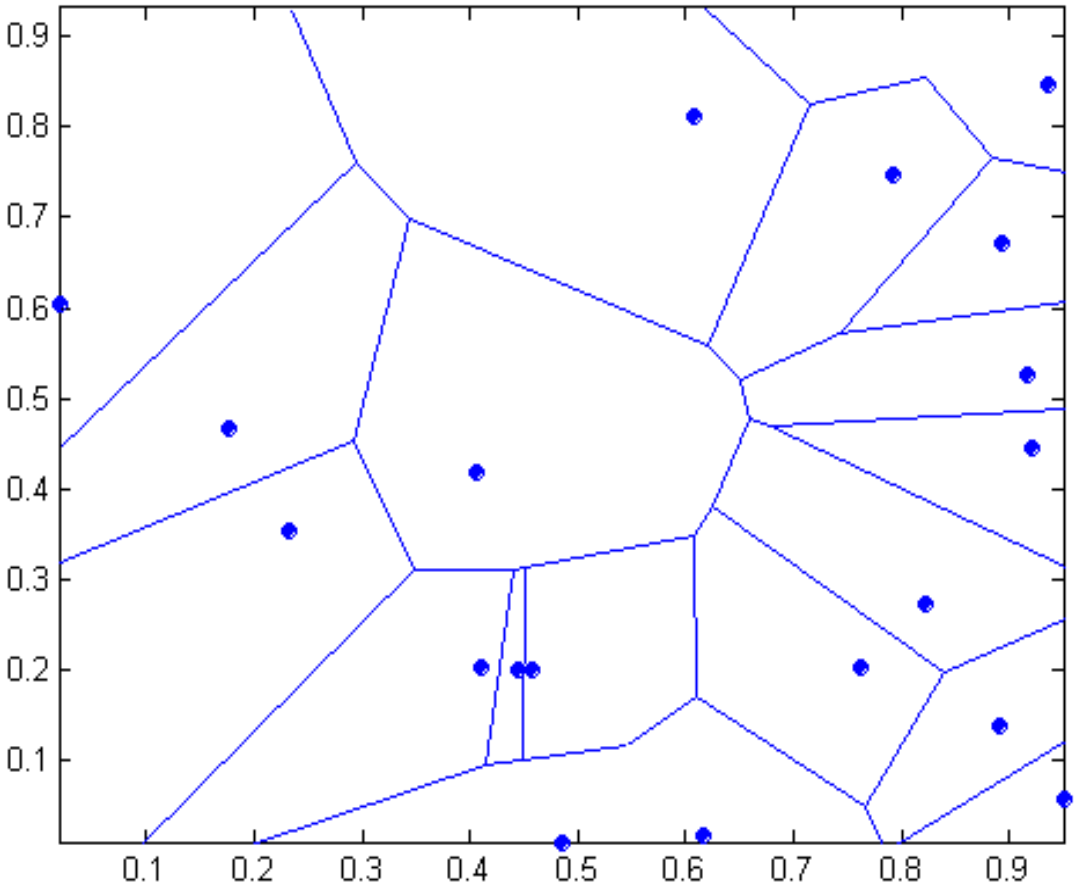


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

I k-nearest neighbors di un campione x sono i campioni che hanno le k minori distanza da x.

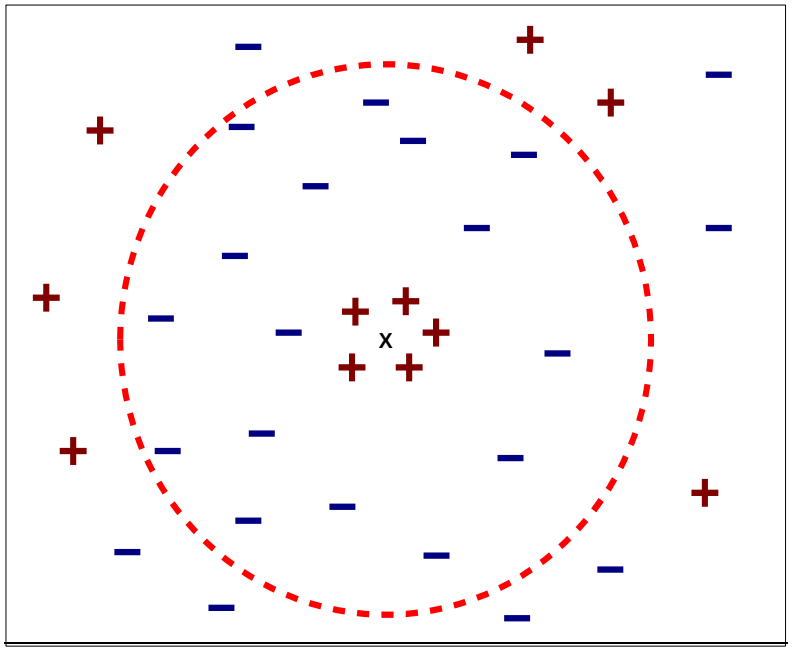
1- nearest-neighbor

E’ in relazione biunivoca con i diagrammi di Voronoi



Il parametro k in k-NN

- Se k è troppo piccolo la classificazione può essere sensibile a dati rumorosi
- Se k è troppo grande:
  - può essere computazionalmente costosa
  - l' intorno può includere campioni appartenenti ad altre classi

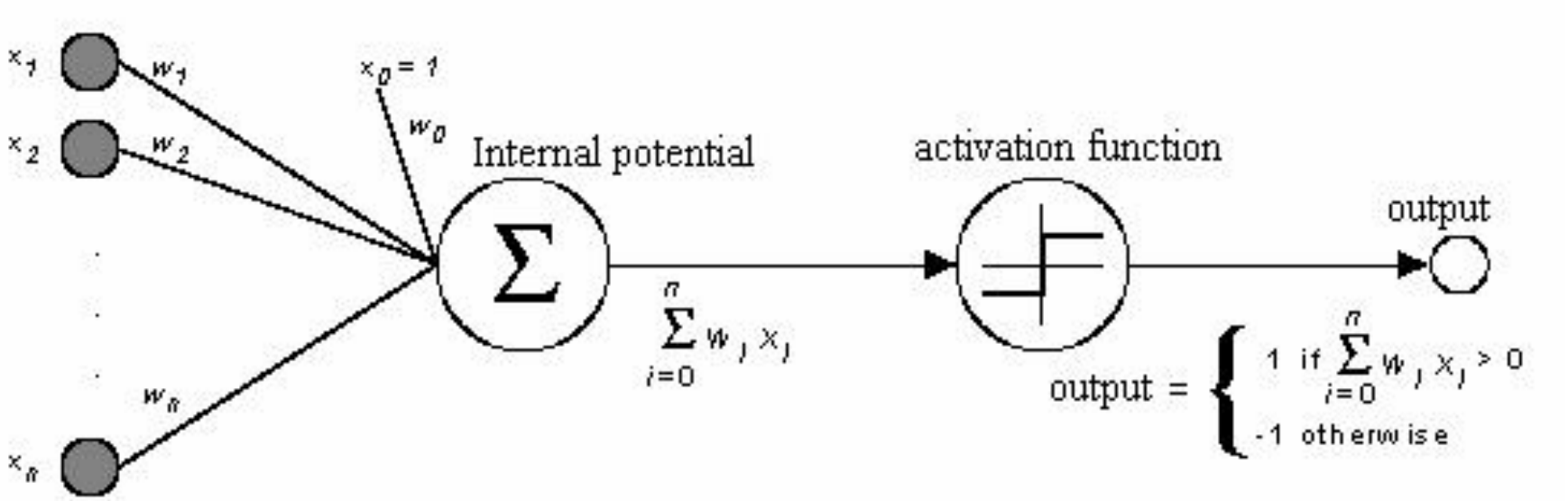


Limiti dei k-NN

– I classificatori k-NN sono “pigri”
  - non costruiscono esplicitamente un modello (come ad es: fanno gli alberi di decisione o le reti neurali)
  - la classificazione può essere realtivamente costosa

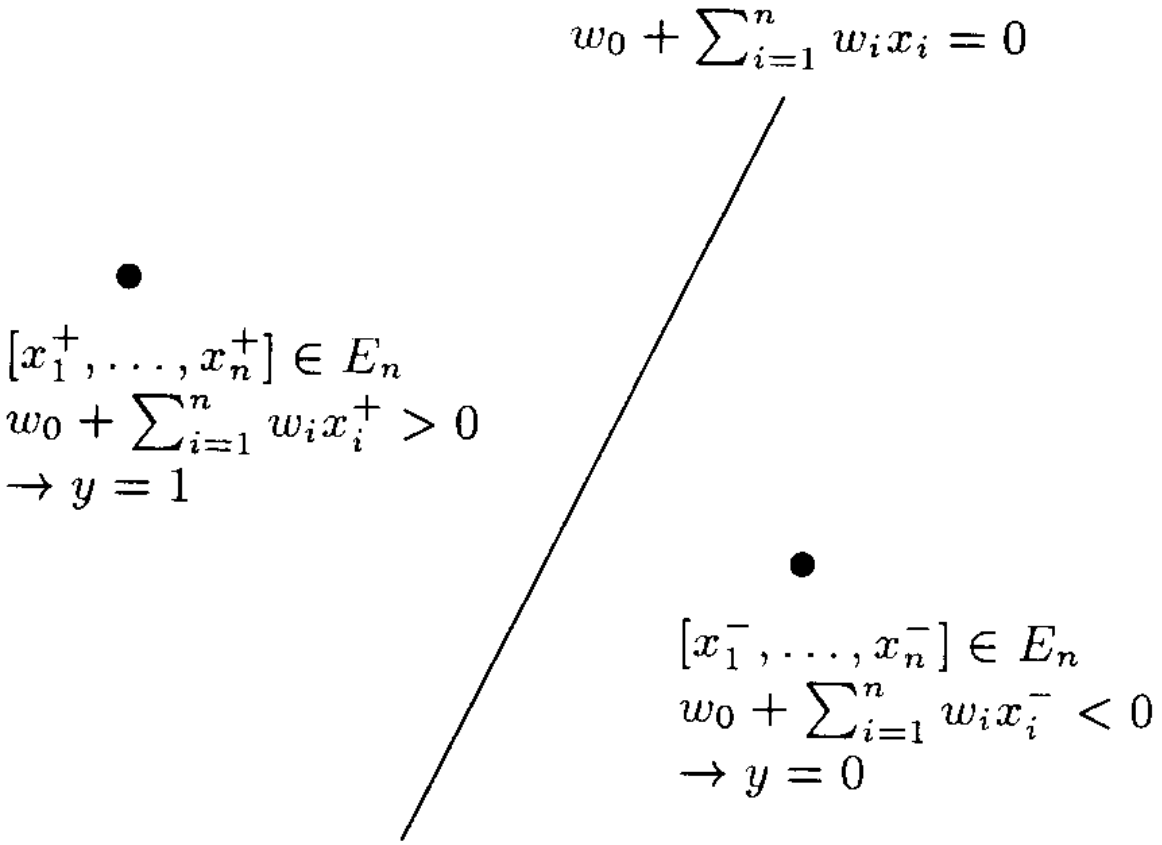
– I dati di espressione genica sono di elevata dimensionalità e k-NN è soggetto al problema del *curse of dimensionality*.

Percettrone (modello lineare di neurone)



- x_1, ..., x_n - input
- w_1, ..., w_n – pesi sinaptici
- w_0 – fattore costante (bias)
- $\sigma(\xi)$ – funzione di attivazione: $\sigma(\xi) = \mathrm{sgn}(\xi)$
- output: $y = \sigma(\xi)$

Interpretazione geometrica della funzione computata da un singolo percettrone

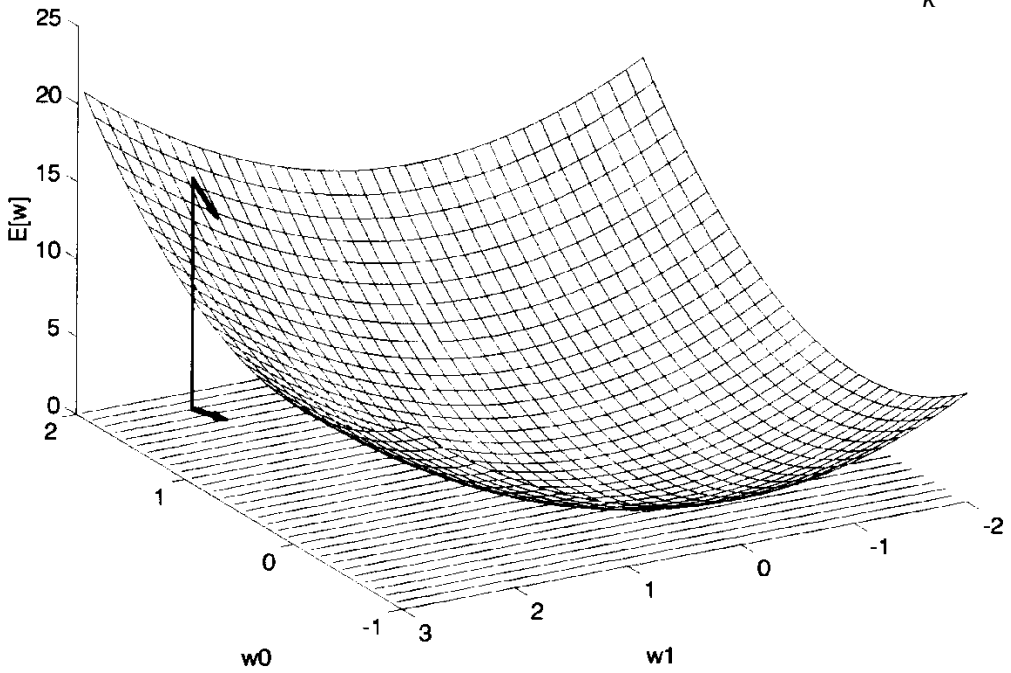


- Un percettrone effettua un classificazione in 2 classi utilizzando un iperpiano (una retta in 2 dimensioni) come superficie separatrice: $\boldsymbol{w}\,\boldsymbol{x} = 0$
- Può classificare correttamente solo insiemi linearmente separabili

Apprendimento del percettrone: minimizzazione dell' errore

- Come calcolare il vettore dei pesi ***w*** dell' iperpiano separatore ? Si minimizza l' errore della funzione $y_k = \mathrm{sgn}(w \cdot x_k)$ computata dal percettrone calcolata rispetto al training set $T = \{(\mathbf{x}_k, t_k)\}_{k=1}^{n}$:

$$E(w) \equiv \frac{1}{2} \sum_{t_k \in T} (t_k - y_k)^2$$



Il vettore dei pesi viene modificato in modo da minimizzare $E(w)$.

Minimizzazione del vettore dei pesi tramite discesa a gradiente

- Idea di fondo: spostarsi sulla superficie di errore verso il minimo.
- La direzione è determinata dalla derivata di *E* rispetto a ciascuna componente di ***w***.

  Gradiente di *E*: $\nabla E(w) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$

- Apprendimento per discesa a gradiente:

  $w_i \leftarrow w_i + \Delta w_i$, dove $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

- Differenziando *E* rispetto ai pesi w_i:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{t_k \in T} (t_k - y_k)^2 = \sum_{t_k \in T} (t_k - y_k)(-x_i)$$

Apprendimento del neurone: algoritmo iterativo di discesa a gradiente

1. Inizializzazione di ciascun peso w_i a valori casuali (piccoli)
2. Finchè non si raggiunge la condizione di stop:
   - $\Delta w_i = 0,\ 1 \leq i \leq d$
   - per ciascun $(\boldsymbol{x}_k, t_k) \in T$:
     - Calcola $y_k = \boldsymbol{w}\boldsymbol{x}_k$
     - Per ciascun peso w_i :

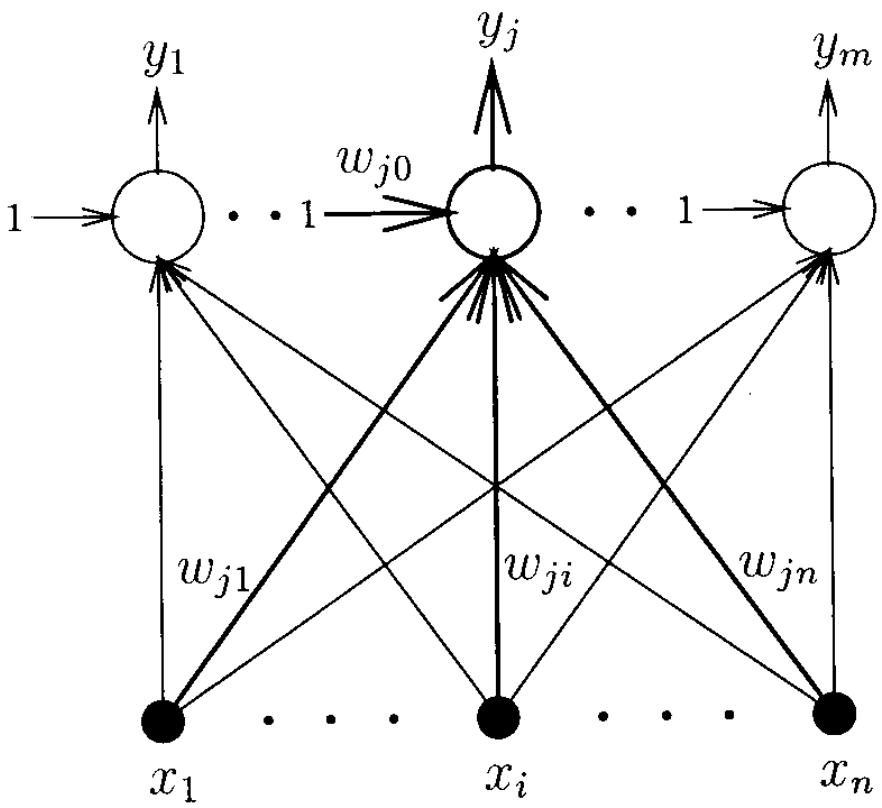
       $\Delta w_i \leftarrow \eta(t_k - y_k)\, x_i$

       $w_i \leftarrow w_i + \Delta w_i$

Possibili condizioni di stop:

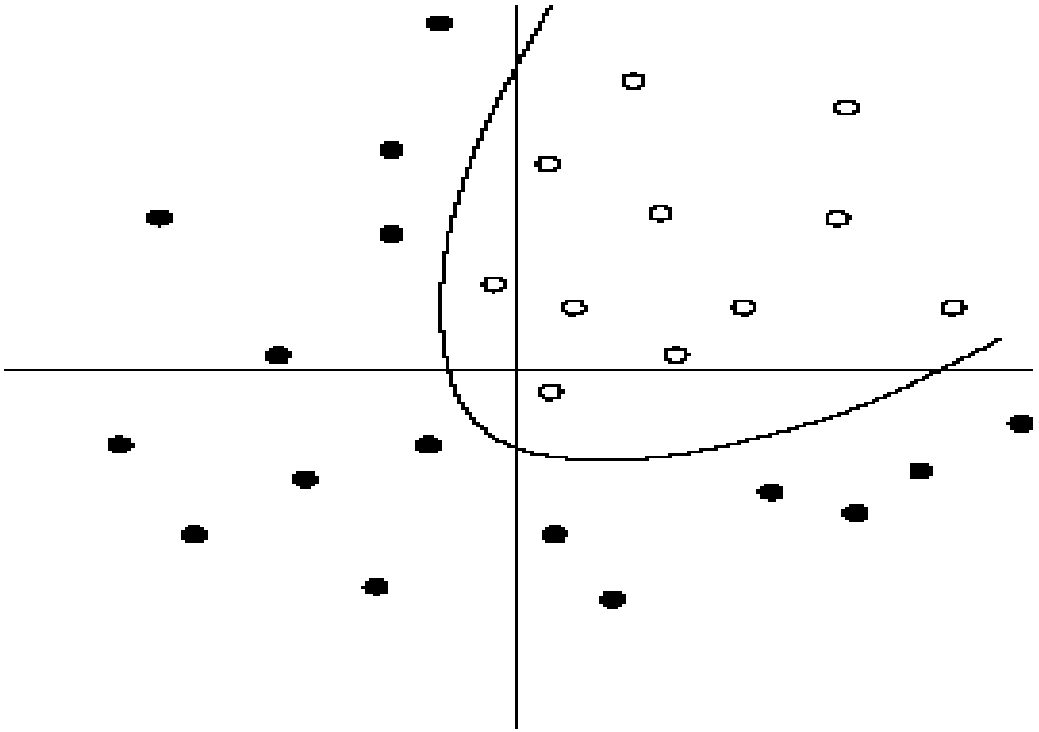
- $E(\boldsymbol{w})$ < soglia prefissata
- $\Delta w_i \rightarrow 0,\ 1 \leq i \leq d$
- Numero di iterazioni > soglia prefissata

Percettrone a singolo strato per classificazione a più classi

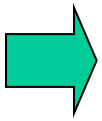


- Un neurone y_j per ogni classe j, $1 \leq j \leq m$ completamente connesso all' ingresso x.
- La classe di uscita viene computata tramite tecnica *WTA* (Winner Takes All)

Come classificare dati non linearmente separabili ?

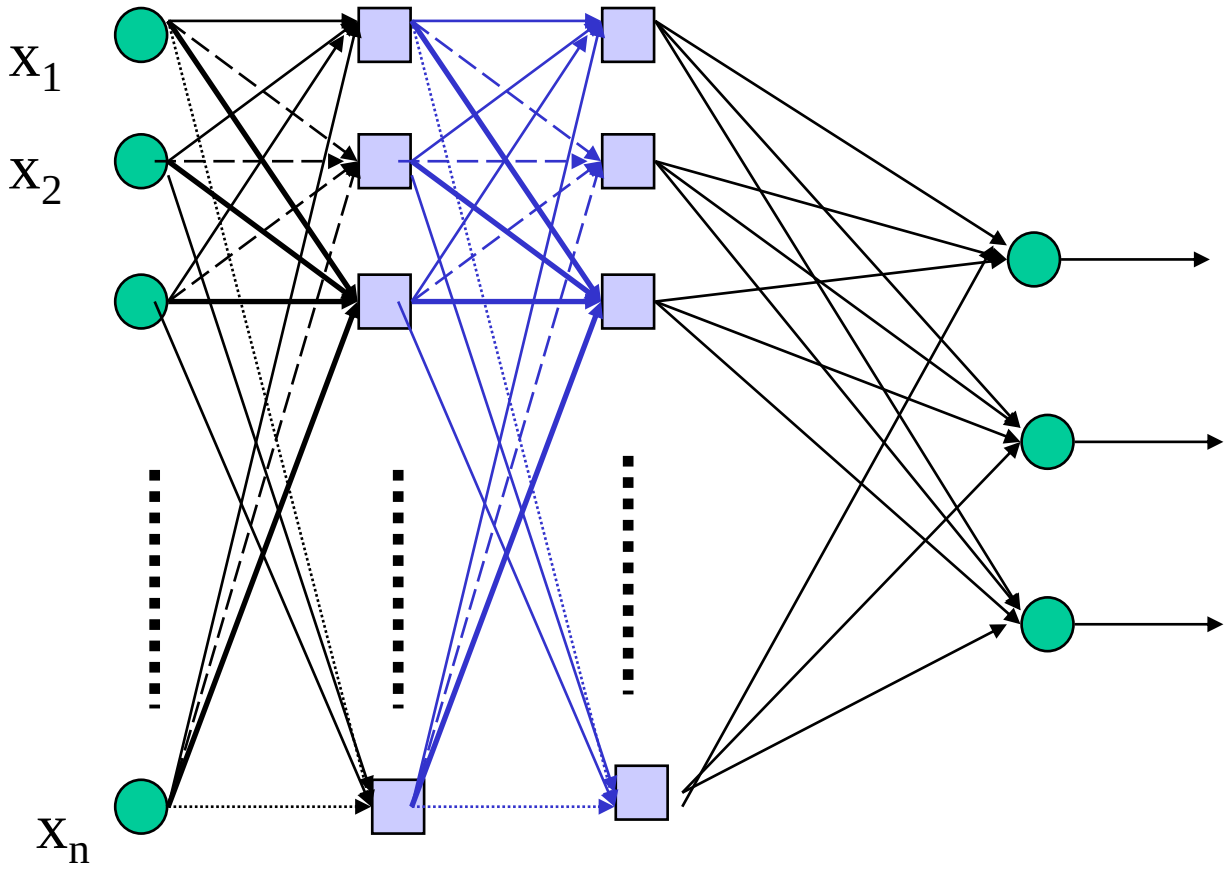


- E' impossibile separare le 2 classi con una retta (percettrone semplice)



- Si devono utilizzare percettroni a più strati …

Struttura di un percettrone multistrato (MLP)



Strato di ingresso

Strati nascosti

Strato di uscita

Addestramento dei MLP

Un algoritmo di discesa a gradiente non è direttamente applicabile:

- Ogni strato ha i suoi pesi che devono essere aggiornati
- Solo l' errore rispetto all' uscita è noto

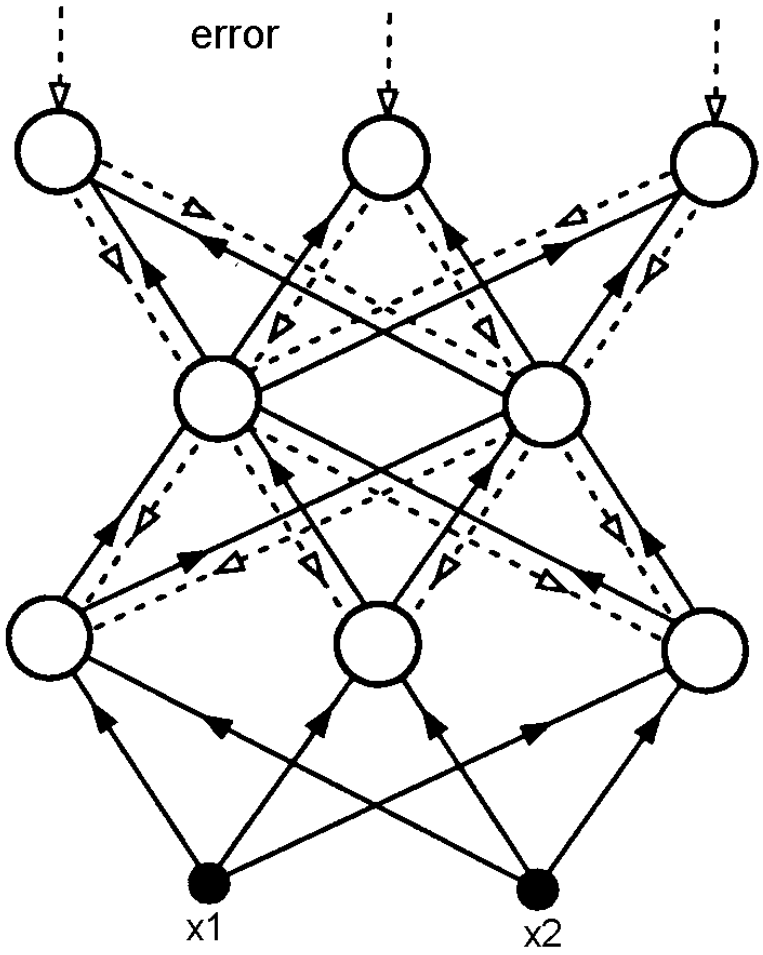
Algortimo di backpropagation (retropropagazione)
(*Rumelhart* et al. 1986)

Visualizzazione dell' algoritmo di backpropagation in una rete neurale a 3 strati

Output

Strati intermedi

Input



computazione

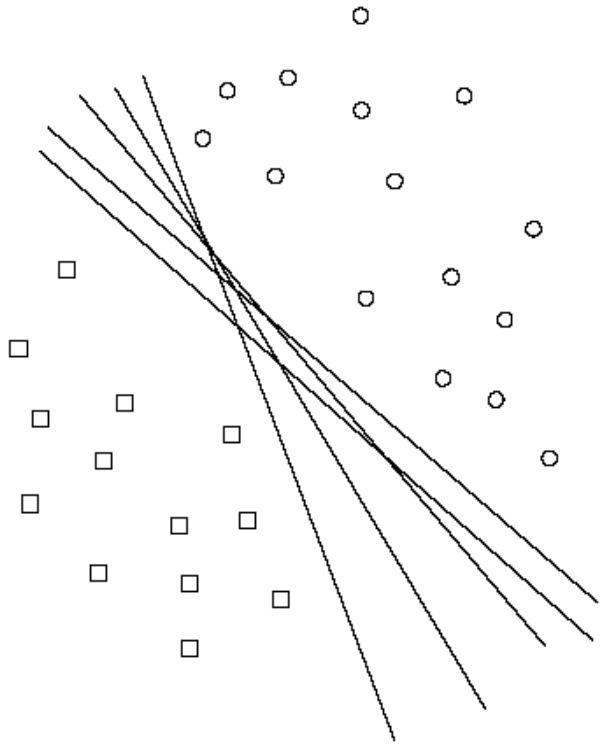
retropropagazione dell' errore

MLP: scelta del modello

- L’ apprendimento dipende dalle condizioni iniziali (valori casuali iniziali di pesi)
- Le capacità di generalizzazione dipendono:
  - dalla topologia
  - dal numero di neuroni degli strati intermedi
  - dalla *regolarizzazione* della rete
  - dalle condizioni di stop selezionate
  - dal coefficiente di apprendimento
  - dalla variante algoritmica utilizzata

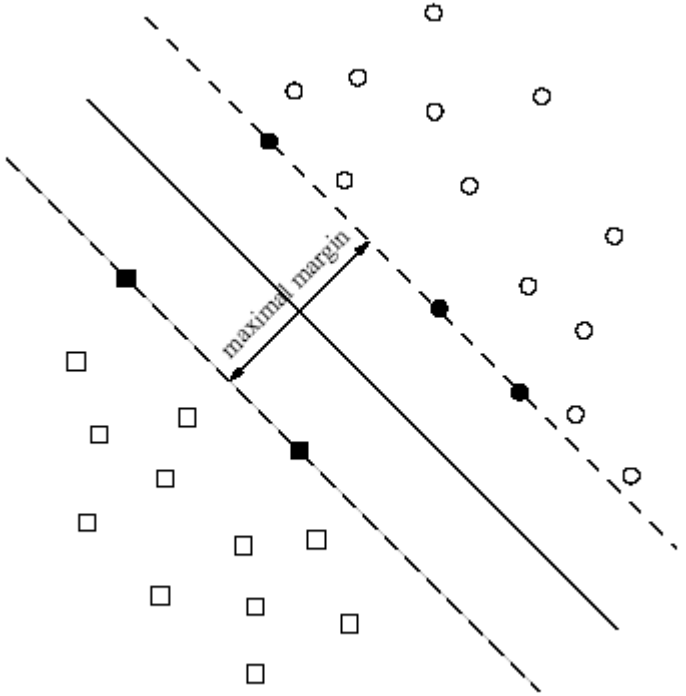
- Provare almeno:
  - diversi numeri di unità nascoste (neuroni degli strati intermedi)
  - diverse condizioni di stop
  - MLP regolarizzati (weight decay)

Support Vector Machine (SVM)



Come classifica una rete neurale (non regolarizzata)

Soluzioni molteplici (minimi locali multipli)

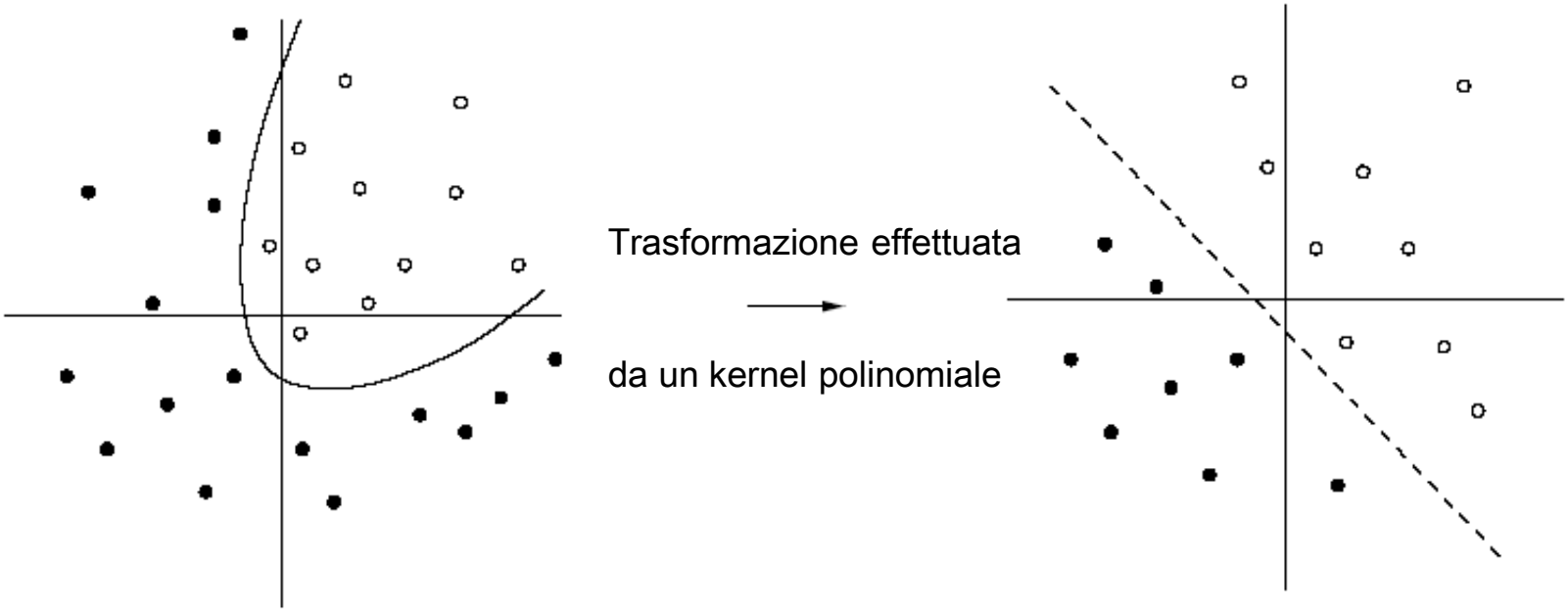


Come classifica una SVM lineare

Soluzione unica (minimo globale)

SVM per problemi non linearmente separabili

- Tramite funzioni kernel (es: funzioni gaussiane e polinomiali) i campioni da classificare sono proiettati in uno spazio iperdimensionale.
- In tale spazio è più probabile che un classificatore lineare riesca a classificare correttamente i campioni (*Teorema di Cover*)



Spazio di ingresso originale: i dati non sono linearmente separabili

La SVM calcola l' iperpiano di separazione ottimale nello spazio trasformato