

Regression

#Altask

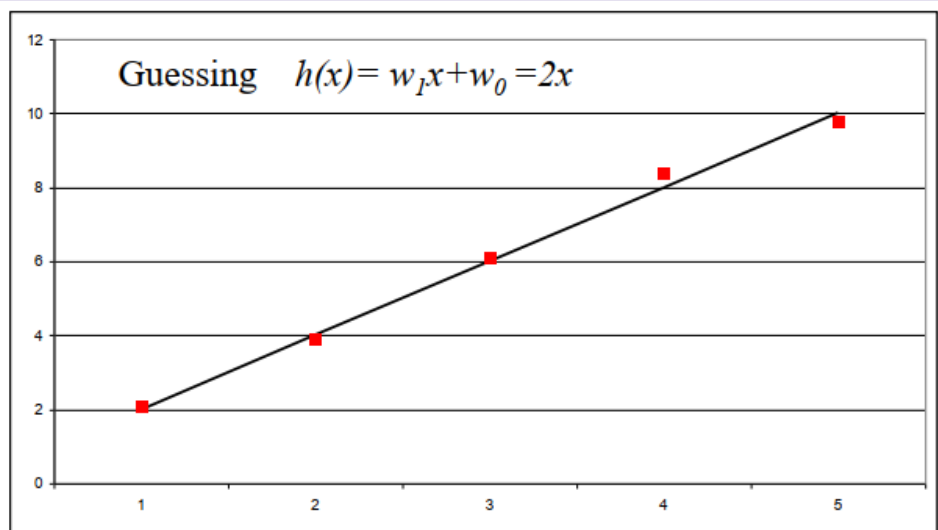
Regression ▾

Processo di stima di una funzione a valori reali sulla base di un insieme di campioni disturbati

- coppie note: $(x, f(x) + \text{random noise})$

Example

x	target
1	2.1
2	3.9
3	6.1
4	8.4
5	9.8
...	...



Vogliamo risolverlo (come trovare w_1 e w_0) in modo "sistematico"

Univariate Linear Regression

Univariate Linear Regression

Assumiamo di avere un $h_w(x)$ espresso come $out = h(x) = w_1 x + w_0$

$$\vec{x} \rightarrow \boxed{h} \rightarrow$$

dove w è un coefficiente a valori reali o un parametro libero (weights)

Caso invariato, semplice regressione lineare, partiamo da una variabile in input e una in output. Adattiamo i dati con una retta.

Task e Modelli

Proviamo quindi a trovare una h (*modello lineare*) che faccia il miglior fitting di dati, avendo un insieme di dati di valori x e y .
Assumiamo che le variabili x e y siano collegate da $y = w_1 x + w_0 + noise$, dove w è un parametro libero e $noise$ è l'errore nella misurazione dei targets.

Proviamo quindi a trovare i valori di w per costruire un modello che faccia predire/stimare y per i valori di x non noti.

Apprendimento via LMS

Premesse

Dobbiamo trovare i valori del parametro w per minimizzare l'errore nell'output del modello (fare un *buon fitting*).
Lo *spazio delle ipotesi* è infinito, ma abbiamo una buona soluzione dalla matematica classica:

- possiamo *apprendere* da strumenti basici
- include diversi concetti rilevanti del ML moderno

INPUT : Insieme di l esempi di training (x_p, y_p) con $p = 1, \dots, l$
OUTPUT : $h_w(x)$ nella forma $w_1x + w_0$ (quindi trovare w) che minimizzi la perdita di dati di training prevista

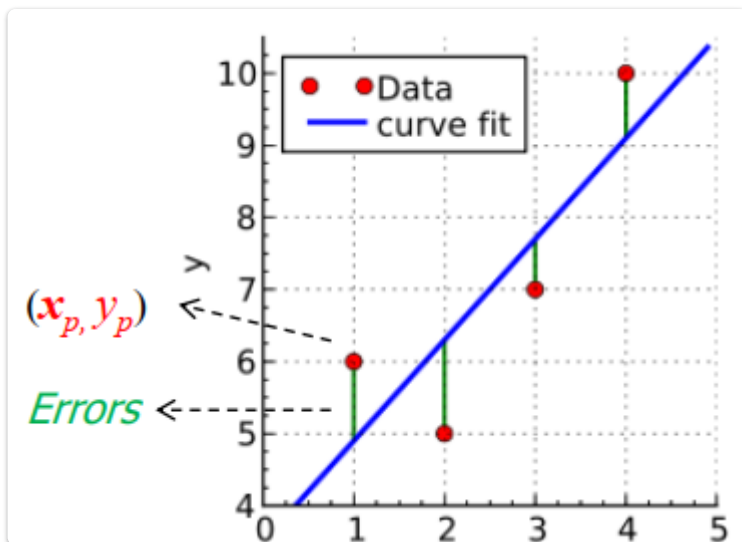
Per la perdita usiamo l'errore dei quadrati

Trovare w che minimizzi la somma residua dei quadrati

$$\begin{aligned} Loss(h_w) = E(w) &= \sum_{p=1}^l (y_p - h_w(x_p))^2 = \\ &= \sum_{p=1}^l (y_p - (w_1x - w_0))^2 \end{aligned}$$

Dove x_p è il p -esimo input, y_p è l'output per p , w sono i parametri liberi e l il numero di esempi.

Perché usarlo - Least Squares



Linee **blu** hanno diverse linee **verdi**. Minimizzare le linee **verdi** è un modo per trovare la migliore approssimazione/fitting dei dati. L'**errore quadratico** $E(w)$ quantifica le linee verdi

$$E(w) = \sum_{p=1}^l (y_p - h_w(x_p))^2$$

Il metodo **least squares** è un approccio standard per approssimare la soluzione di un sistema sopra-stimato, quindi un insieme di equazioni dove ci sono più equazioni conosciute che non.

Cosa risolve?

Minimo locale come punto stazionario \rightarrow il gradiente è nullo

$$\frac{\partial E(w)}{\partial w_i} = 0 \quad i = 1, \dots, (\dim_input + 1)$$

Per la regressione lineare semplice (2 parametri semplici)

$$\frac{\partial E(w)}{\partial w_0} = 0 \quad \frac{\partial E(w)}{\partial w_1} = 0$$

Le soluzioni finali non sono rilevanti per il corso

Calcolo del gradiente

Regola base

$$\frac{\partial}{\partial w} k = 0 \quad \left| \quad \frac{\partial}{\partial w} w = 1 \quad \left| \quad \frac{\partial}{\partial w} w^2 = 2w \quad \left| \quad \frac{\partial (f(w))^2}{\partial w} = 2f(w) \frac{\partial (f(w))}{\partial w} \right. \right. \\ \Rightarrow \frac{\partial E(w)}{\partial w_i} = \frac{\partial (y - h_w(x))^2}{\partial w_i} = 2(y - h_w(x)) \frac{\partial (y - h_w(x))}{\partial w_i} = 2(y - h_w(x)) \frac{\partial (y - (w_1 x + w_0))}{\partial w_i} \\ \boxed{\frac{\partial E(w)}{\partial w_0} = -2(y - h_w(x))} \quad \boxed{\frac{\partial E(w)}{\partial w_1} = -2(y - h_w(x)) \cdot x}$$

Gradiente decrescente

Local Search

La derivazione precedente, suggerisce di costruire un *algoritmo iterativo* basato su $\frac{\partial E(w)}{\partial w_i}$:

Gradiente = DIREZIONE DI SALITA

Possiamo muoverci verso il minimo con un gradiente decrescente ($\Delta w = -\text{gradiente di } E(w)$)

Local Search: si comincia da un vettore di pesi iniziale, poi si modifica iterativamente per decrementare e minimizzare l'errore delle funzioni

Nb

$$w_{new} = w + \eta \Delta w \quad \text{dove } \overset{\text{eta}}{\eta} \text{ è una costante } \left(\begin{matrix} \text{step} \\ \text{size} \end{matrix} \right) \\ \text{chiamata learning rate}$$

Motivazioni intuitive

$$w_{new} = w + \eta \Delta w \quad \left| \quad \Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2(y - h(x)) \quad \left| \quad \Delta w_1 = \frac{\partial E(w)}{\partial w_1} = 2(y - h_w(x)) \cdot x \right. \right.$$

Delta Rule

É una regola per correggere gli errori (detta **delta rule**) che combina w proporzionalmente all'errore ($target - output$):

- Se $target - output = err = 0 \rightarrow$ nessuna correzione
- Se $output > target \rightarrow (g - k) < 0 \rightarrow$ l'output è troppo alto
 - Δw_0 negativo \rightarrow riduce w_0
 - $if(input > 0) \Delta w_1$ negativo \rightarrow riduci w_1 (else incrementalo)

Miglioriamo l'apprendimento dagli errori precedenti.

L'approccio del gradiente decrescente è un approccio di **local search** semplice ed efficace per le soluzioni LMS. Ci permette di cercare in *spazi delle ipotesi infiniti* e può sempre essere applicato per **H continuo** e perdite differenziali. Per renderlo più efficiente ci sono miglioramenti possibili.

Linear models

Riassunto per l patterns (x_p, y_p)

$$\Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2 \sum_{p=1}^l (y_p - h_w(x_p)) \quad \left| \quad \Delta w_1 = -\frac{\partial E(w)}{\partial w_1} = 2 \sum_{p=1}^l (y_p - h_w(x)) \cdot x$$

Aggiornare w dopo (ripetendo)
ogni "epoca" di l dati di training



Batch Algorithm

Aggiornare w dopo ogni
pattern p



One-Line Algorithm

Multidimensional input

Caso standard: usando *da 2 a 100 variabili* in input

$$x = [x_1, x_2, \dots, x_n]$$

(il pattern in input è un vettore) ci sono molte possibilità in un largo range di campi applicativi

Recap sulla notazione dei dati

Pattern	x_1	x_2	x_i	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$	$x_{1,i}$	$x_{1,n}$
...
Pat p	$x_{p,1}$	$x_{p,2}$	$x_{p,i}$	$x_{p,n}$
...

X è una matrice $l \times n$.

Spesso dobbiamo omettere gli indici quando il contesto è chiaro, ad esempio:

- Ogni riga, un generico x , una riga nella tabella (*input*)
- x_i o x_j (scalari): componenti i o j (dato un pattern \implies quindi si ammette p)
- $x_{p,i}$ (scalare) come (x_p) : componente i del pattern p
- Per il target di solito usiamo y_p con $p = 1, \dots, l$

Notazione con input multidimensionale

Assumiamo di avere *vettori colonna* per x e y , un numero l di dati, una dimensione n dei vettori in input e un target y_p

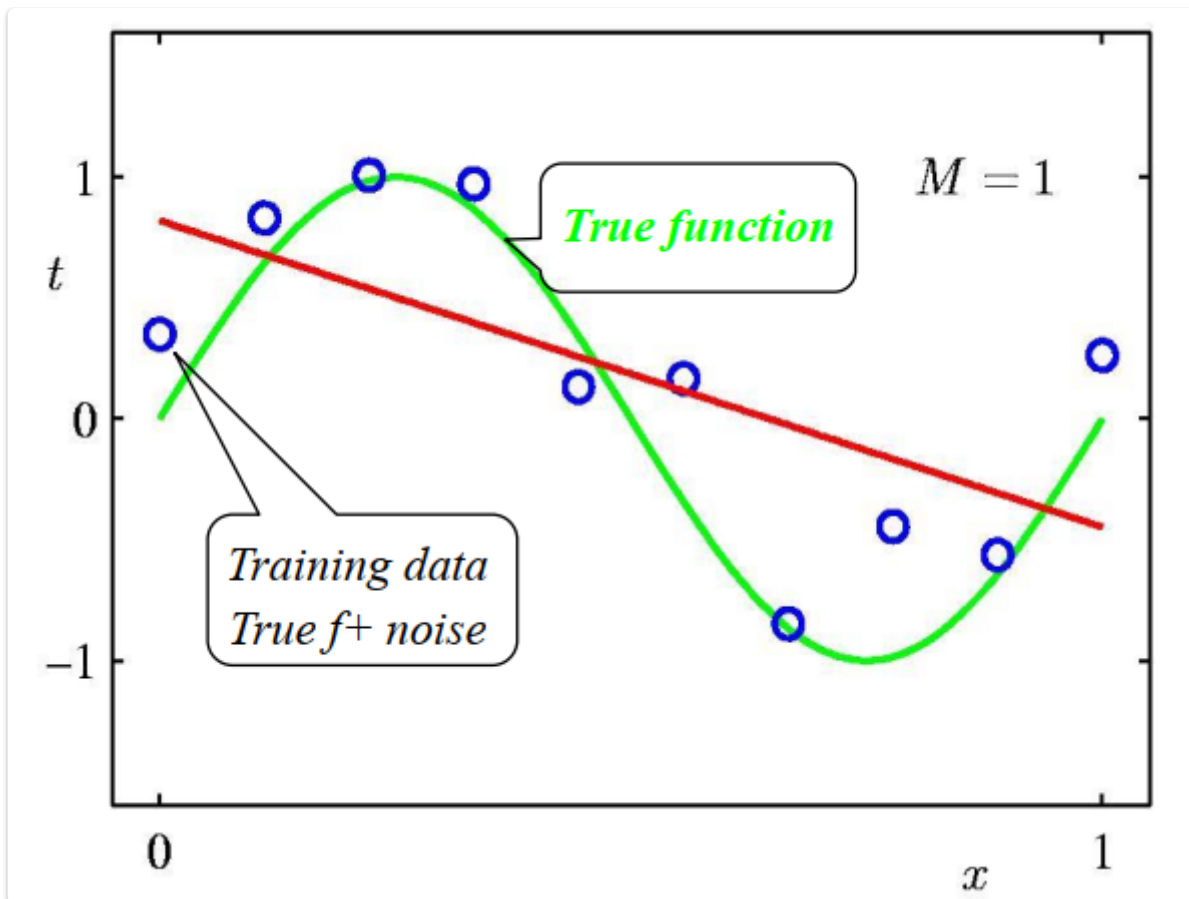
$$w^T x + w_0 = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = w_0 + \sum_{i=1}^n w_i x_i$$

Spesso conviene includere $x_0 = 1$ così possiamo scrivere $w^T x = x^T w$

Vantaggi del modello lineare

Se funziona bene è molto semplice, tutte le informazioni dei dati stanno in w , è facile da interpretare e sono ammessi dati disturbati. È una *baseline dell'apprendimento* ed è stato usato/incluso in molti modelli complessi.

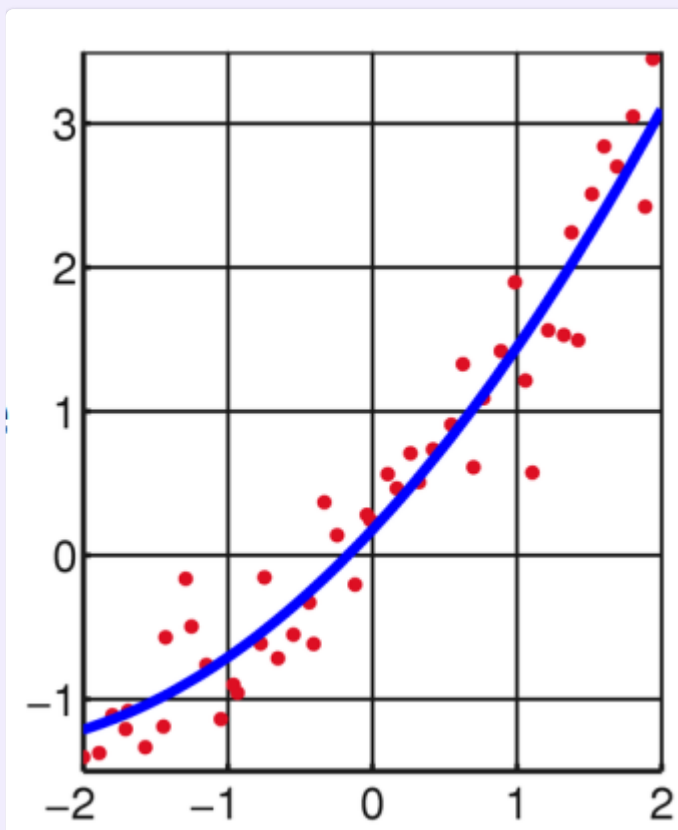
Limitazione



Muoversi verso una relazione non lineare: come un modello parametrico statistico: "lineare" non si riferisce alla linea dritta, ma al modo in cui i coefficienti di regressione w occorrono nell'equazione di regressione. Possiamo anche usare input trasformati con input e output a relazioni non lineari

$$h_w(x) = \sum_{j=0}^M w_j x^j \quad \begin{array}{l} \text{regressione} \\ \text{polinomiale} \end{array}$$

Example



Il risultato del fitting di una funzione quadratica, $M=2$, verso un insieme di dati. Nel linear squares least la funzione

necessita di essere *non lineare* negli argomenti (variabili in input), ma solo nei parametri w determinati per avere il miglior fit.

Generalizzazione - LBE

Quote

Trasformazione di base - *linear basis expansion*:

$$h_w(x) = \sum_{k=0}^N w_k \phi_k(x)$$

Si argomenta il vettore in input con variabili più che siano trasformazioni di x secondo la funzione ϕ ($\phi_k : \mathbb{R}^n \rightarrow \mathbb{R}$)

Ad esempio:

- Rappresentazione polinomiale di x : $\phi(x) = x_j^2$ o $\phi(x) = x_i x_j$ o ...
- Trasformazioni non lineari di input singoli: $\phi(x) = \log(x_j)$, $\phi(x) = \text{root}(x)$, ...
- Trasformazioni non lineari di input multipli: $\phi(x) = ||x||$

Il modello è *lineare nei parametri* (in ϕ , non in x): possiamo usare lo stesso algoritmo di apprendimento di prima

Example

$\phi_j(x) = x^j \Rightarrow h(x) = w_0 + w_1 x + w_2 x^2 + \dots + w_n x^M = \sum_{j=0}^M w_j x^j \Rightarrow$ Regressione polinomiale a 1 dimensione

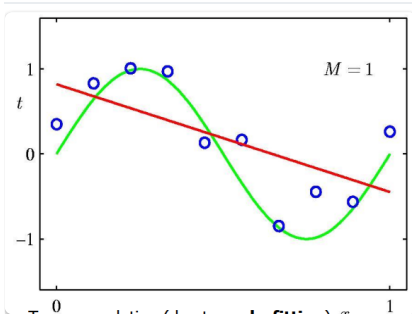
$\phi(x) = \phi([x_1, x_2, x_3]) \Rightarrow h(x) = w_1 x_1 + w_2 x_2 + w_3 \log(x_2) + w_4 \log(x_3) + w_5 (x_2 x_3) + w_0$

PRO CONTRO

È più più *espressivo*: può modellare relazioni più complicate

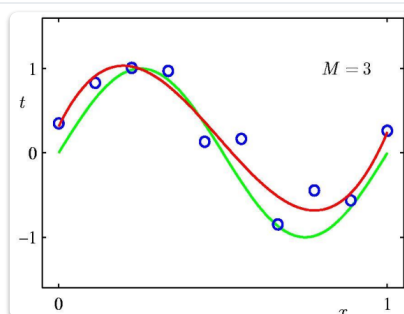
Con una grande base di funzioni, abbiamo bisogno di metodi per controllare la *complessità del modello*

Polinomio del 1° ordine



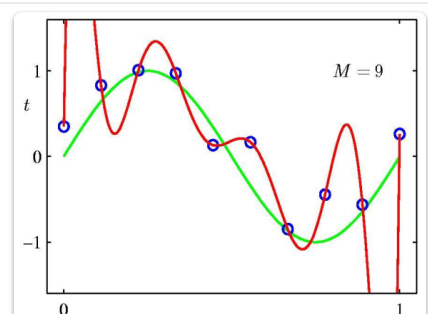
Soluzione povera: *underfitting*

Polinomio del 3° ordine



Più flessibile e più usabile

Polinomio del 9° ordine



Molto flessibile ma può essere *eccessiva* (overfitting)

Complessità

- Modello molto semplice \rightarrow non fa un buon fit dei dati \rightarrow *underfitting*
- Modello molto complesso \rightarrow troppa sensibilità alla perturbazione dei dati \rightarrow *overfitting*
Vogliamo trovare la regolarizzazione per bilanciare i due casi verso il *controllo della complessità del modello*. La complessità non è intesa come costo computazionale ma è una *misura della flessibilità* del modello per il fit dei dati.

Regolarizzazione

Può controllare l'*overfitting* penalizzando le funzioni "complesse" con i pesi w grandi o il numero di parametri liberi, mantenendo la flessibilità dello spazio delle ipotesi.

//The simplest explanation is more likely the correct one//
Ockham razor (1300)

//Prefer the simplest hypothesis that fits the datas//

Regolarizzazione con Ridge Regression

🔗 Ride Regression / Regolarizzazione di Tikhonov

Modelli smussati: possibilità di aggiungere *vincoli* alla somma dei $|w_j|$ favorendo modelli "sparsi".

$$Loss(h_w) = \sum_{p=1}^l (y_p - h_w(x_p))^2 + \lambda ||w||^2$$

Ad esempio, con meno termini, a causa dei pesi $w_j = 0$ (o quasi), λ (costante) è detta *coefficiente di regolarizzazione*.
L'*effetto* è la decadenza del peso $\Rightarrow w_{new} = w + \eta \cdot \Delta w - 2\lambda w$ (si aggiunge $2\lambda w$ al gradiente del Loss).

☰ Example

Con gradiente 0, decrementa il valore di ogni w con una frazione del vecchio w

Considerazioni

APPLICABILITÀ GENERALE

Ad esempio possiamo controllare la complessità del modello usando solo λ , senza sapere M (grado dell'LBE) o quando non conosciamo il modello.

NOTARE IL **BILANCIAMENTO** (TRADE-OFF) TRA I DUE TERMINI

- Vogliamo controllare la complessità del modello per controllare l'*overfitting* \rightarrow introduciamo un secondo termine nella minimizzazione
- Possiamo eccedere perché troppo peso sul secondo termine porta il focus della minimizzazione solo (o per la maggior parte) sulla *regolarizzazione* \rightarrow l'errore dei dati può crescere troppo \rightarrow *underfitting*
- Il trade-off è rimesso dal valore di λ

Limitazioni delle funzioni a base fissa

Avendo funzioni di base lungo ogni dimensione di uno spazio di input a D dimensioni (k per LBE), si richiede un *numero combinatorio di funzioni*

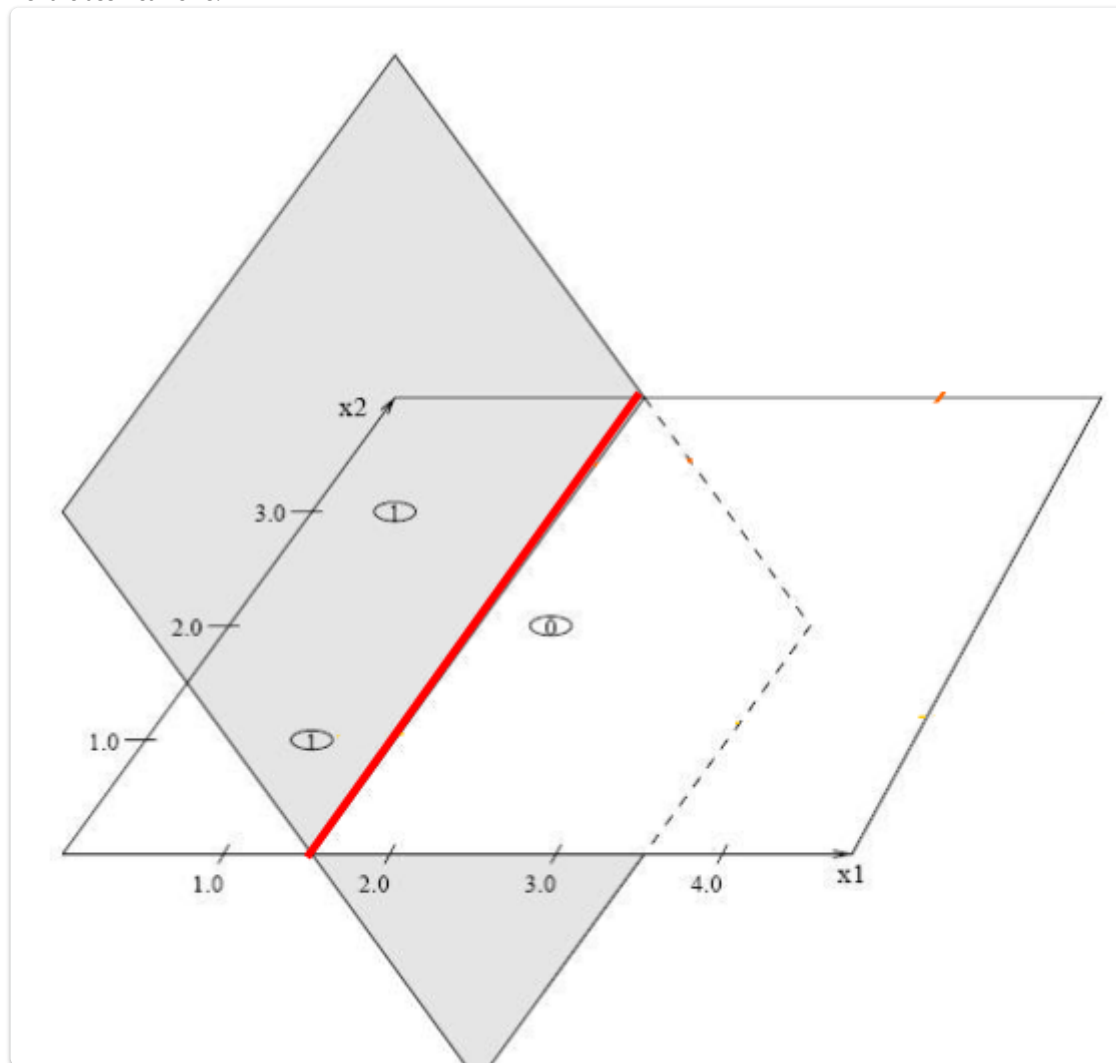
- I dati diventano sparsi in un grande volume \rightarrow La quantità di dati ha bisogno di supportare che il risultato cresce sempre esponenzialmente con la dimensione
- Φ è fissato prima di osservare i dati di training
Quando dovremmo vedere come possiamo uscire con funzioni con *basi basse*, scegliendole usando i dati di training: Φ dipende da w e il modello è non lineare nei parametri.
In altri modelli la computazione del nuovo spazio è fatto implicitamente attraverso il nucleo delle funzioni e controllando la complessità del modello.

Classificazione

Classificazione dei modelli lineari

Lo stesso modello usato per la regressione, può essere usato per la *classificazione* (**target concept**). In questo caso usiamo un *hyperplane* (wx) assumendo valori possibili e negativi. Sfruttiamo questi modelli per *decidere se un x appartiene alla zona* più o meno dell'*hyperplane*. Vogliamo quindi imporre x (con l'apprendimento) t.c. abbiamo una buona accuratezza

nella classificazione.



$w^T x$ definisce un hyperplane

$$w^T x = w_1 x_1 + w_2 x_2 + w_0 = 0 \quad \begin{array}{l} \text{decision} \\ \text{boundary} \end{array}$$

Può essere usato per la classificazione:

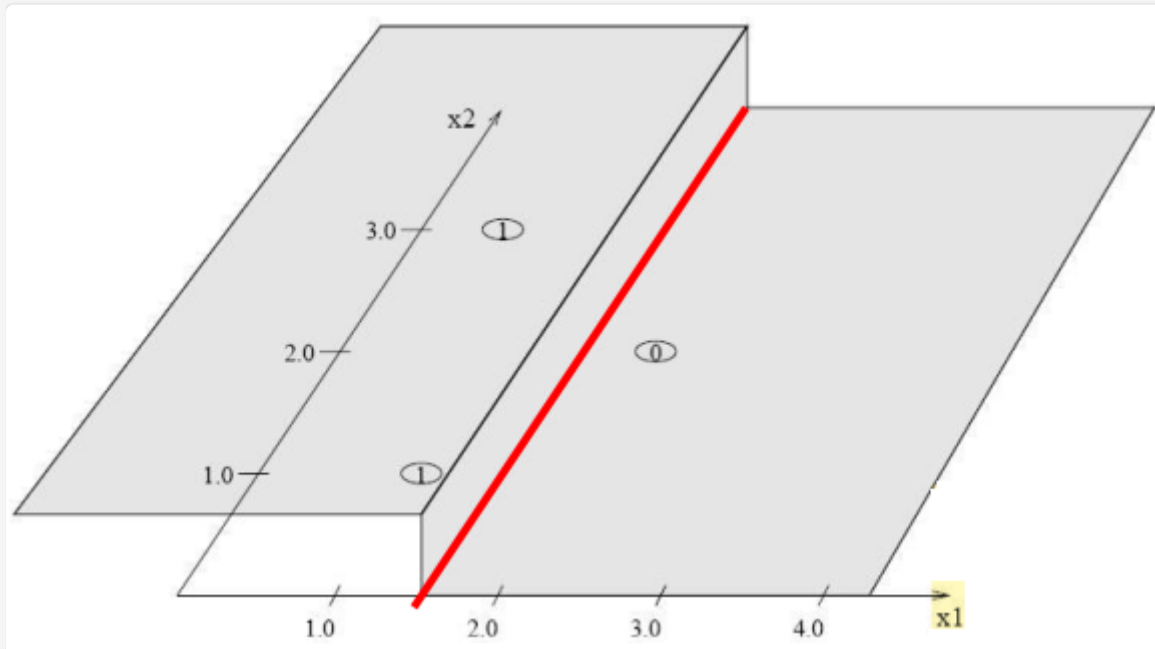
es:

$$\rightarrow \langle (1.0, 1.0), 1 \rangle$$

$$\rightarrow \langle (0.5, 3.0), 1 \rangle$$

$$\rightarrow \langle (2.0, 2.0), 0 \rangle$$

Dove x è il vettore di input e w i parametri liberi



$$h(x) = \begin{cases} 1 & \text{if } wx + w_0 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$h(x) = \text{sign}(wx + w_0)$$

$$h(x_p) = \text{sign}(x_p^T w) = \text{sign}(\sum_{i=0}^n x_{p,i} w_i)$$

Classificazione con boundary di decisione lineare

La classificazione può essere vista come l'allocazione dello spazio di input nell regione di decisione

$$h(x) = \begin{cases} 1 & \text{if } wx + w_0 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$h(x) = \text{sign}(wx + w_0)$$

Linear threshold unit (LTU)

⚠ Nota

dando il bias w_0 nell'LTU divedendo

$$h(x) = w^T x + w_0 \geq 0$$

è equivalente a dire

$$h(x) = w^T x \geq -w_0$$

con $-w_0$ threshold value

- Le due forme identificano la stessa zona positiva del classificatore
- La seconda enfatizza il ruolo del bias come un valore threshold per "attivare" l'output +1 del classificatore

Problema lineare per classificatore lineare

🔗 important

INPUT: un insieme l di esempi di training

OUTPUT: w che minimizza la somma residua dei quadrati (LS)

$$E(w) = \sum_{p=1}^l (y_p - x_p^T w)^2 = \|y - Xw\|^2$$

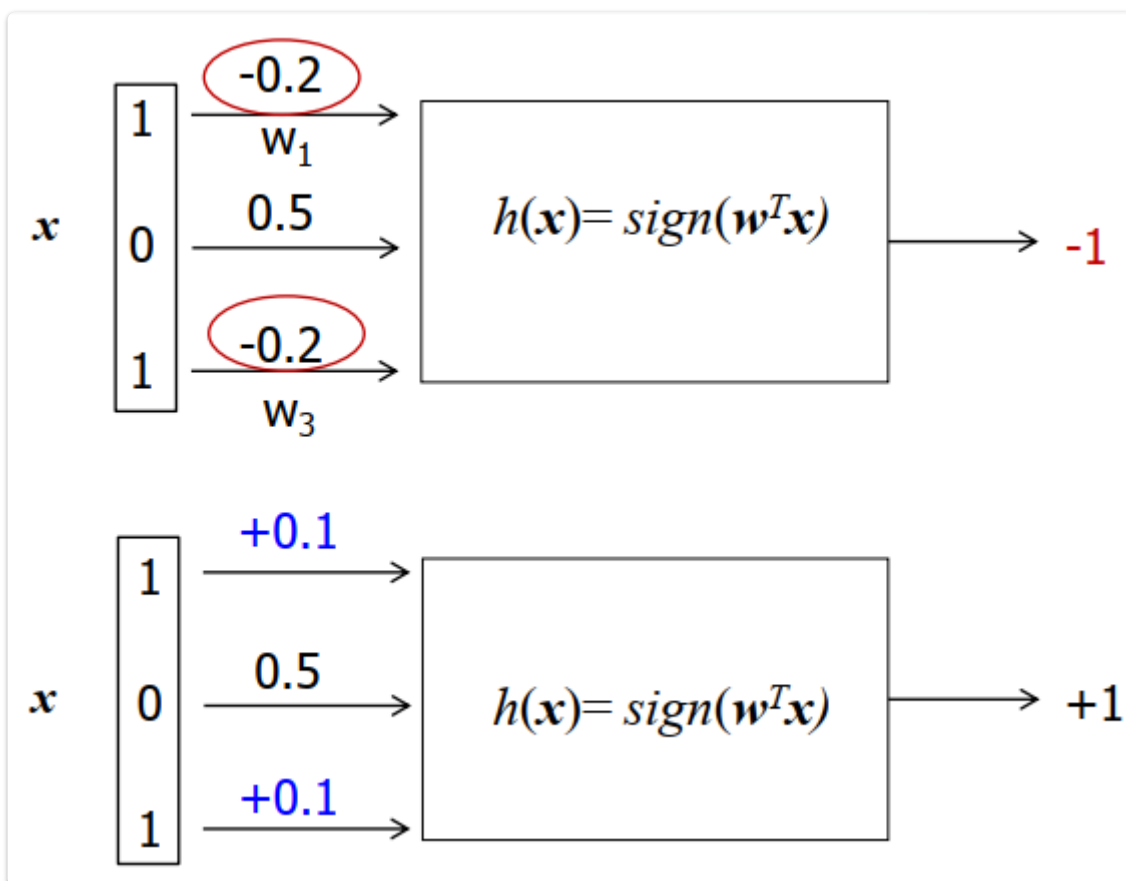
dove

- x_p è il p -esimo vettore di input
- y_p l'output per p
- w i parametri liberi
- l il numero di esempi
- n la dimensione degli esempi in input

Errore Minimo: se $y_p = 1$ allora $x_p^T w \rightarrow 1$. Se $y_p = 0 / -1$ allora $x_p^T w \rightarrow 0 / -1$. Si noti che $E(x)$ **NON** usa la forma $h(x)$.

Abbiamo un *algoritmo iterativo con gradiente decrescente*

$$\begin{aligned} \Delta w_i &= \frac{\partial E(w)}{\partial w_i} = \\ &= \sum_{p=1}^l (y_p x_p^T w - x_{p,i}) \end{aligned}$$



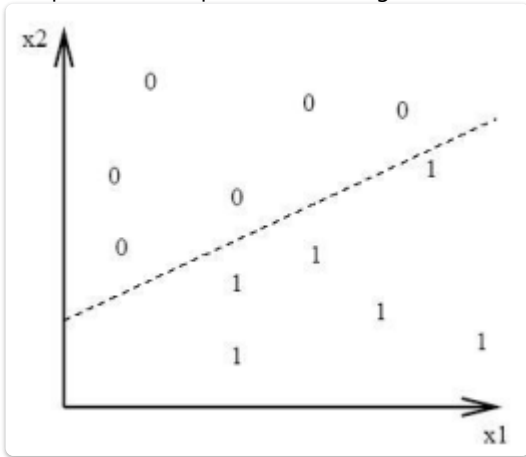
$$(w_0 = 0) \quad \Delta w_i = \sum_{p=1}^l (y_p - x_p^T w) - x_{p,i}$$

Se **mal classificato** (perché il target è +1)

- Il delta è positivo e grande per w_1 e w_3
- Si incrementano proporzionalmente al delta, quindi in questo esempio un valore positivo [**error connection rule**]

Limitazioni

In geometria, 2 insiemi di punti in un graico a 2 dimensioni è **linearmente separabile** quando i 2 insiemi possono essere completamente separati da una singola retta.

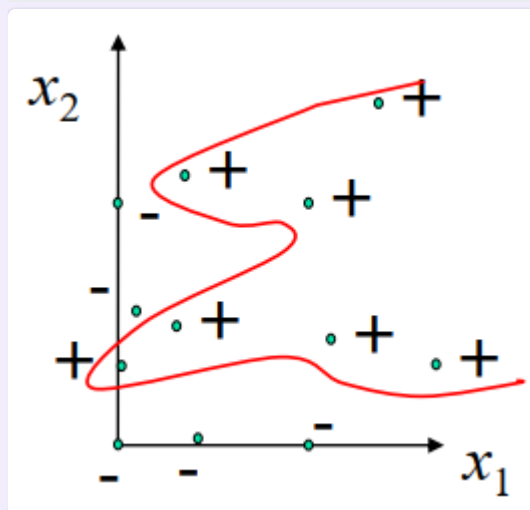


In generale, 2 gruppi sono linearmente separabili in **n -dimensioni** se possono essere separati da un **iperpiano a $(n - 1)$ dimensioni**.

Il **linear decision boundary** può fornire soluzioni esatte solo per insiemi di punti linearmente separabili.

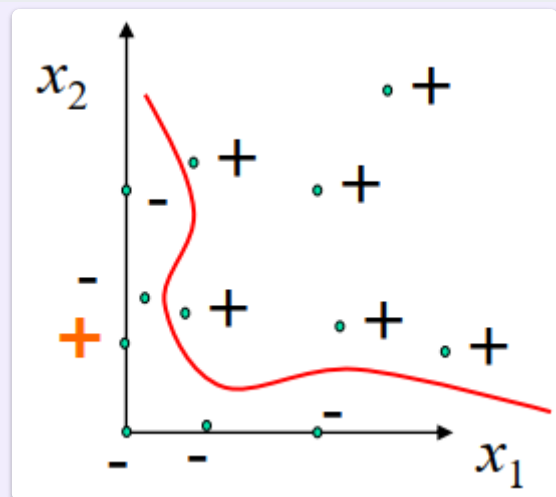
Example

decision boundary **non lineare** (da un'**espansione di base**)



Otteniamo un classificatore **non smooth** che può aver bisogno di regolarizzazione

decision boundary **non lineare** (da un'**espansione di base**) **regolarizzato**



In questo caso un errore di training (+) è ammesso

Altri modelli di apprendimento per la classificazione

- **Linear Discriminant Analysis** (anche multi-class)

- **Logistic Regression**

$P(y/x)$ comincia dal modellare la densità della classe come fosse una densità nota. Il threshold è soft (continua, differenziabile) con funzioni logistiche, anziché 0/1 hard threshold

- **Neural Networks (NN) e SVM**

Modelli flessibili che includono **approssimazioni non lineari** sia per la regressione che per la classificazione

- usa molte unità (tipo LTU) con diversi livelli
- apprendimento della rappresentazione delle features in ogni livello
- approccio del gradiente decrescente per l'apprendimento