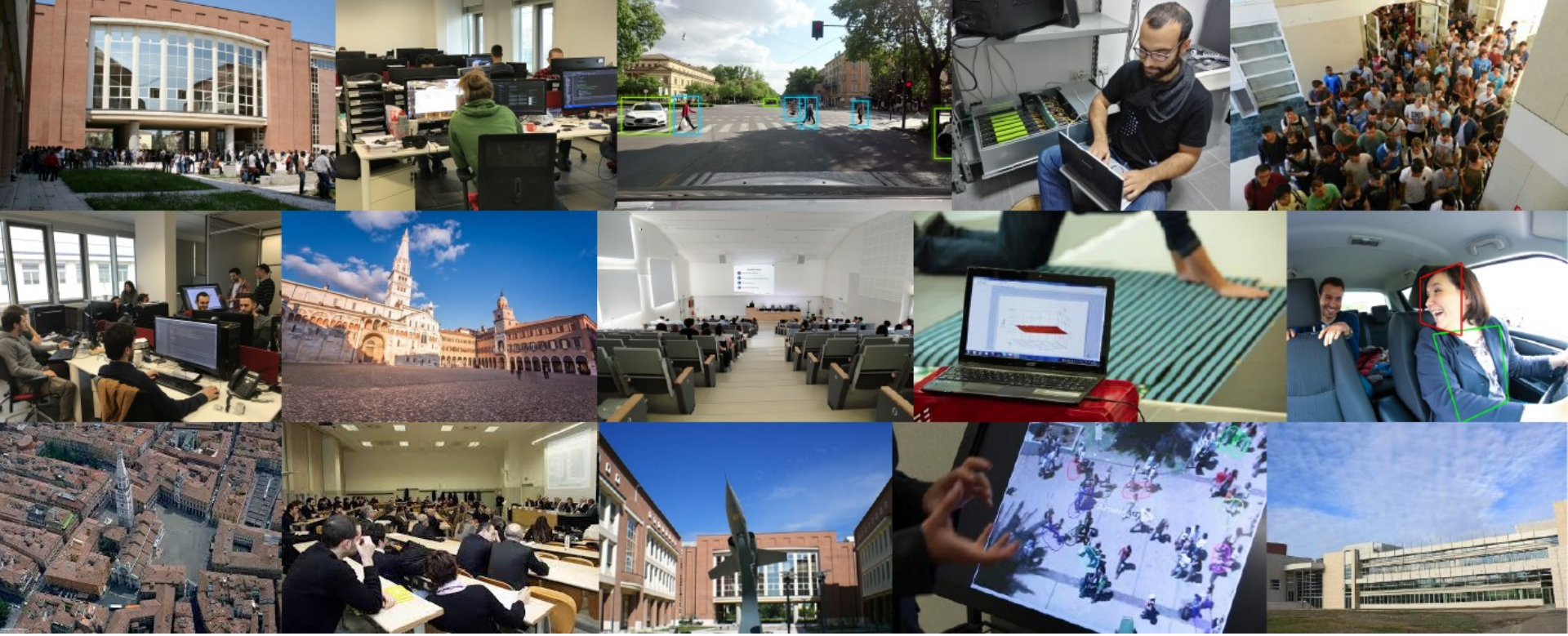


MACHINE LEARNING
Metodi Parametrici e Ottimizzazione -

AImage Lab

UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Metodi Parametrici e Ottimizzazione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

In questa lezione introdurremo dei concetti generali di ML

Considereremo solo il caso supervisionato, ma la maggior parte dei concetti che affronteremo possono essere estesi al caso non supervisionato

In particolare vedremo, anche dal punto di vista *intuitivo*, come un problema di ML possa essere risolto tramite una funzione dipendente da “parametri”

Trovare il valore dei parametri “giusti” è l’obiettivo della fase di training. Vedremo come

Notazione

x → valore scalare

$\mathbf{x} = [x_1, \ldots, x_j, \ldots, x_d]$ → feature vetctor

y → variabile target

$(\mathbf{x}^{(i)}, y^{(i)}),\quad i = 1, \ldots, n$ → training sample

$\mathbf{w} = [w_1, \ldots, w_j, \ldots, w_k]$ → parameter vector

Metodi Parametrici

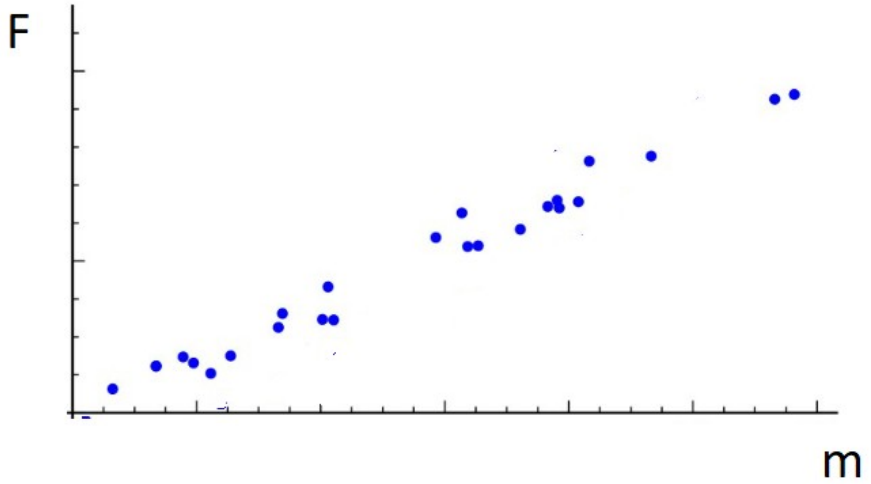
Modello statistico

- Riprendiamo l'esempio visto nella lezione di introduzione, in cui ci siamo messi nei panni di Newton per indurre dai dati una relazione che leghi la forza di attrazione gravitazionale misurata sulla superficie terrestre e la massa di un corpo

- Le nostre osservazioni empiriche costituiscono il dataset di training $T = \{(m_1,F_1), (m_2,F_2), \ldots, (m_n,F_n)\}$

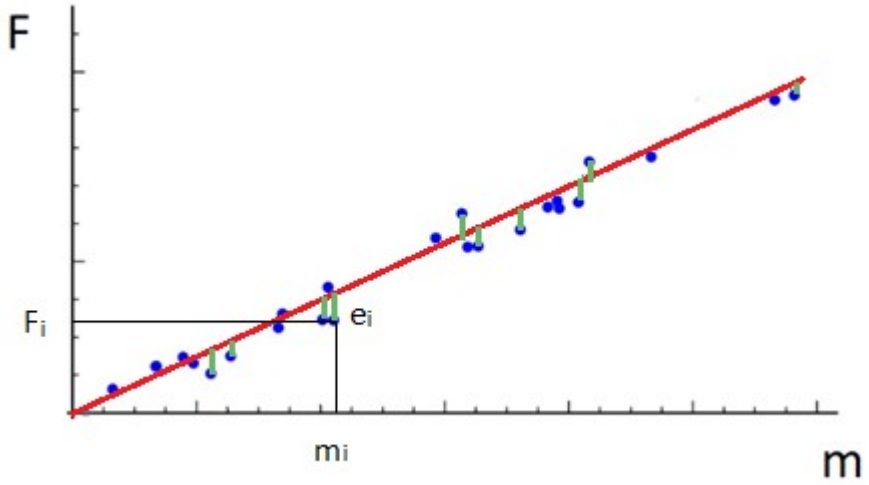
- Se rappresento T graficamente come nella figura a destra (EDA…) intuisco che F e m hanno una dipendenza lineare

- Potrei anche misurare quantitativamente questo fenomeno calcolando $corr(F,m)$



Modello statistico

- Data questa relazione lineare, un modello statistico che «spiega» *T* potrebbe essere il seguente:
- $F = m * k + e$,
- dove $m * k$ esprime la relazione lineare predominante tra le due variabili, mentre *e* è una variabile di «errore», che racchiude tutta la conoscenza non nota (è la parte non deterministica del modello)
- Detto in altri termini, sto assumendo che esista un valore di *k* tale che la retta $F = m * k$ (in rosso nella figura a fianco) «spiega» le osservazioni commettendo però degli errori
- Gli errori possono essere dovuti a vari fattori, tipo:
  - Errori di misurazione
  - Variabili *latenti*, ovvero non note, non comprese nel mio modello (e.g., la distanza dal centro della terra…)



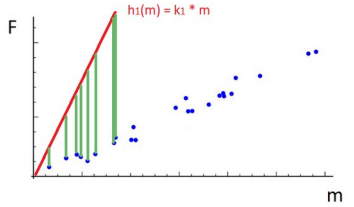
Regressione

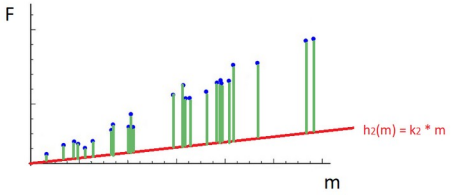
AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

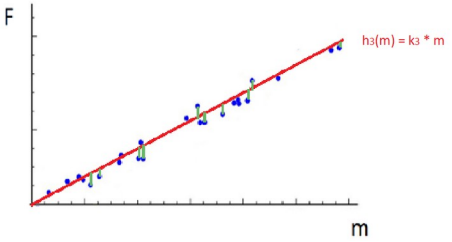
- Dal punto di vista del ML lo scopo del modello statistico è quello di creare un modello predittivo
- Per essere più precisi, nell'esempio che stiamo seguendo devo affrontare un task di *regressione*, voglio cioè predire il valore *numerico F* corrispondente ad una generica massa *m* (non nota in fase di training)
- Il termine «regressione» deriva dal fatto che, a partire da coppie di valori input-output ($T = \{(m_1,F_1), (m_2,F_2), \ldots, (m_n,F_n)\}$) di una funzione *ignota* $F = h(m)$, voglio «regredire» alla funzione che ha generato queste coppie
- $h(m)$ è detta *funzione ipotesi*
- Usando $h()$, posso riscrivere il modello statistico in questo modo:
- $F = h(m) + e$
- Ovvero, il modello statistico si basa su una funzione ipotesi che spiega i dati commettendo però degli errori
- Se cambio la funzione ipotesi, cambierà anche il modello statistico

Regressione Lineare

- Nel caso *particolare* in cui *h(m)* ha una forma lineare, si parla di *regressione lineare*
- Nel nostro esempio, la funzione ipotesi lineare potrebbe avere questa forma:
- *h(m) = m * k*,
- dove *k* è il *parametro* di *h()*
- Al variare di *k* ottengo funzioni ipotesi diverse ($h_1()$, $h_2()$, …)
- Per ora lasciamo stare come calcolare *k* (lo vedremo dopo)
- Intuitivamente, dovrò cercare il *k* che corrisponda ad una funzione ipotesi che faccia meno errori possibili rispetto alle osservazioni in *T*
- Per essere più precisi, la funzione ipotesi *F = h(m) è un modello deterministico che approssima il modello statistico F = h(m) + e*, e voglio trovare il parametro *k* per il quale quest'approssimazione sia la minore possibile
- Prima di vedere come calcolare *k*, generalizziamo i







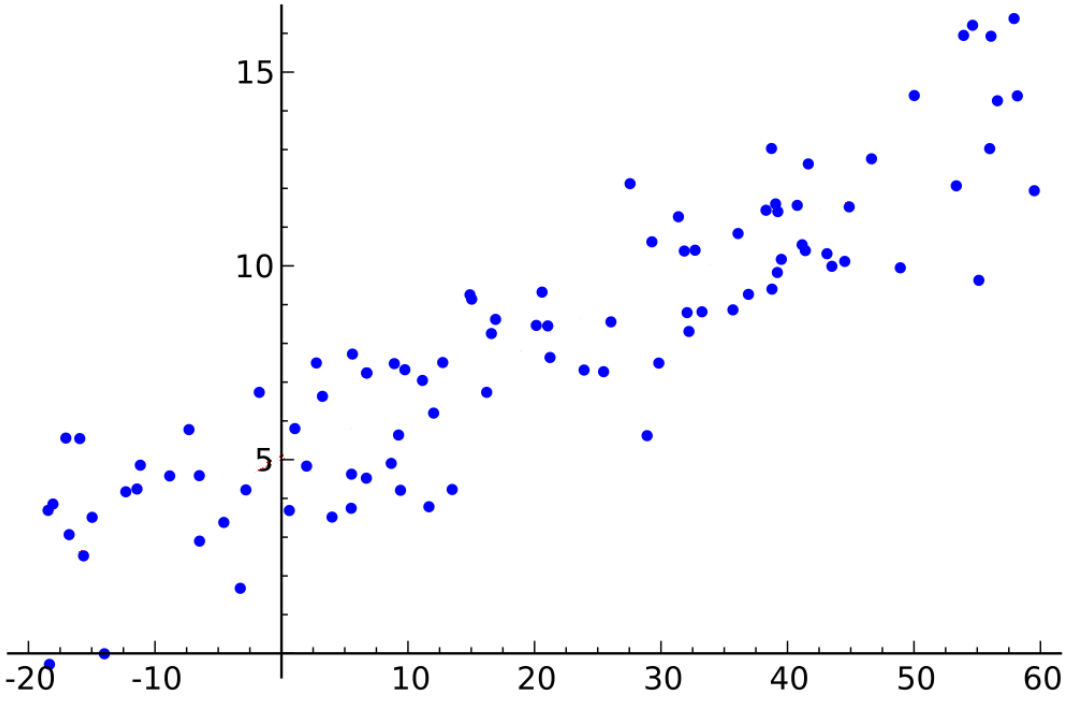
Regressione lineare

Se i dati di training sono distribuiti come nella figura a fianco, non posso più assumere che la funzione ipotesi sia una retta passante per l'origine

In questo caso posso usare una funzione ipotesi del tipo*:*

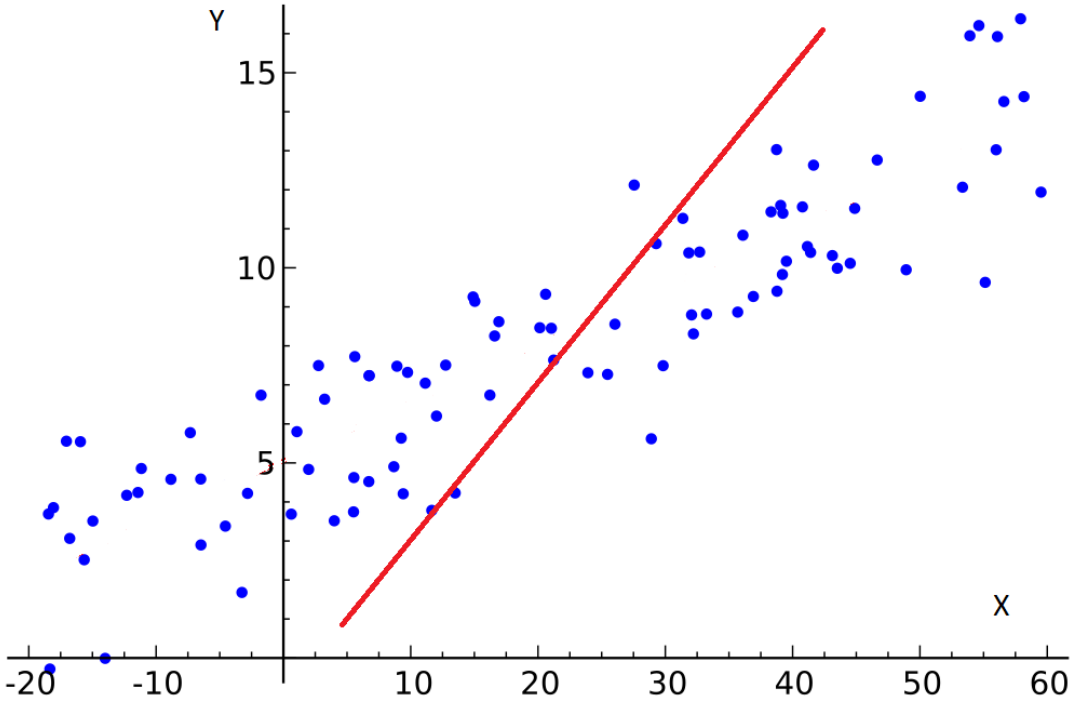
$y = h(x) = a\,x + b$

N.B.: *h()* è ancora lineare, ma adesso ha due parametri, *a* e *b*



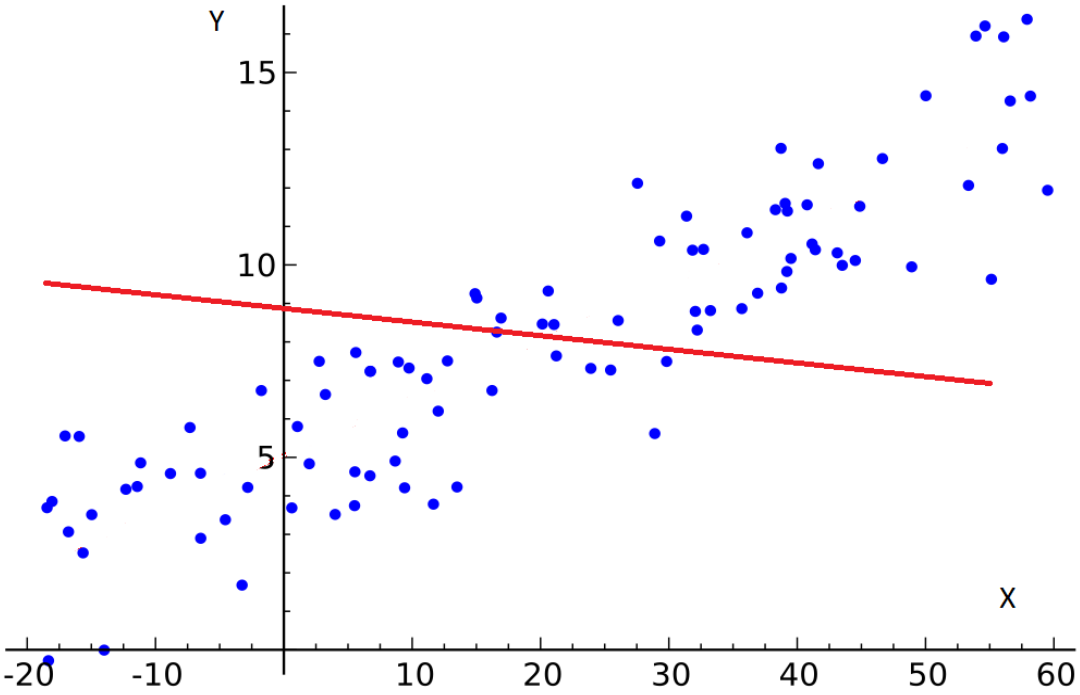
Esempio

$h_1(x) = a_1 x + b_1$



Esempio

$h_2(x) = a_2\, x + b_2$



Esempio

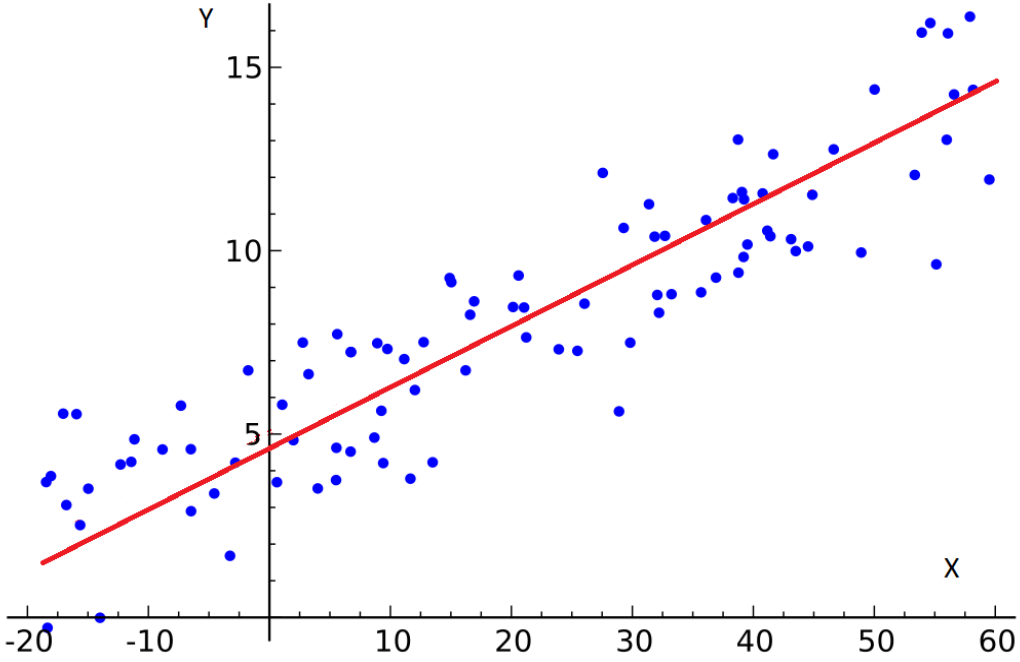
La *forma specifica* di *h()* è determinata dall'insieme dei suoi parametri

Posso indicare sinteticamente l'insieme dei parametri di *h()* tramite il *parameter vector* ***w*** *= [a,b]*

Scrivo *h(x; **w**) = a x + b* per indicare che *h()* è un *modello parametrico* e che ***w*** è il suo vettore di parametri

Un'altra terminologia equivalente, comunemente usata, è: $h_{\mathbf{w}}(x) = a\,x + b$

Trovare ***w*** è lo scopo della fase di

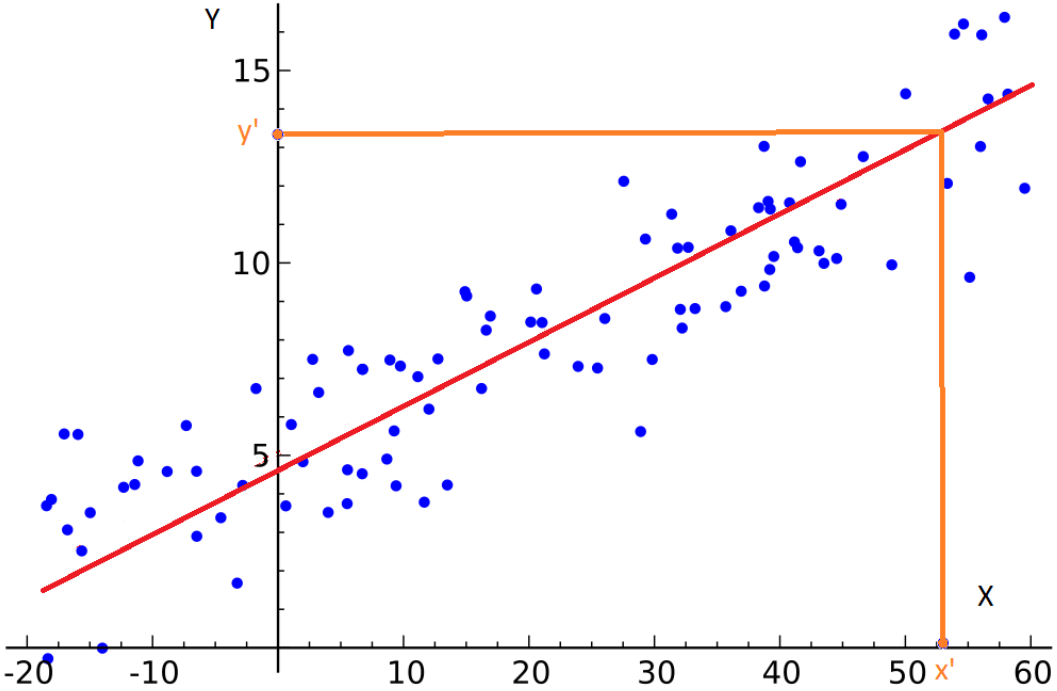


Esempio

Fase di inferenza: ora ***w*** è fissato e non lo modifico più!

Quando ho un nuovo input *x'*, uso:

*y' = h(x'; **w**) = a x' + b* per predire il valore più probabile della variabile dipendente rispetto a *x'*



Regressione a più variabili

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Nei casi reali, però, avrò parecchie variabili indipendenti (i.e., feature)
- Nell'esempio qui sotto, ho 6 variabili indipendenti, che indico, genericamente, con x_1, x_2, …, x_6, e una variabile dipendente (y)

Features | **Label**

Sample(s)

Temperature - Zone 1	Temperature - Zone 2	Voltage - Engine 1	Voltage - Engine 2	Accelerometer 1	Accelerometer 2	Energy consumption
25.4	26.8	12.2	14.4	0.01	0.05	25.6
21.3	19.7	11.8	14.7	0.033	0.045	22.8
22.2	33.6	11.9	14.9	0.012	0.067	27.8
24.3	32.1	12.0	14.0	0.098	0.105	21.2
22.8	27.9	11.0	13.5	0.001	0.003	18.8

Feature space

- Il generico sample ***x*** sarà quindi un vettore a *d* dimensioni (nell'esempio di prima, *d = 6*) e il training set *T* sarà:

$$T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(i)}, y^{(i)}), ..., (\boldsymbol{x}^{(n)}, y^{(n)})\}$$
$$y^{(i)} \in \mathbb{R}$$
$$\boldsymbol{x}^{(i)} = (x_1^{(i)}, ..., x_j^{(i)}, ..., x_d^{(i)}) \in \mathbb{R}^d$$

- Adesso ho uno spazio a *d* dimensioni che rappresenta le feature (detto *feature space*)
- Se volessi visualizzare *T* come ho fatto prima, avrei bisogno di rappresentare uno spazio a *d + 1* dimensioni … (impossibile se *d > 2*)

Regressione Lineare a più variabili

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Un modello *parametrico* ***e*** *lineare* per un task di regressione a *d* variabili è espresso, genericamente, tramite la seguente funzione ipotesi:

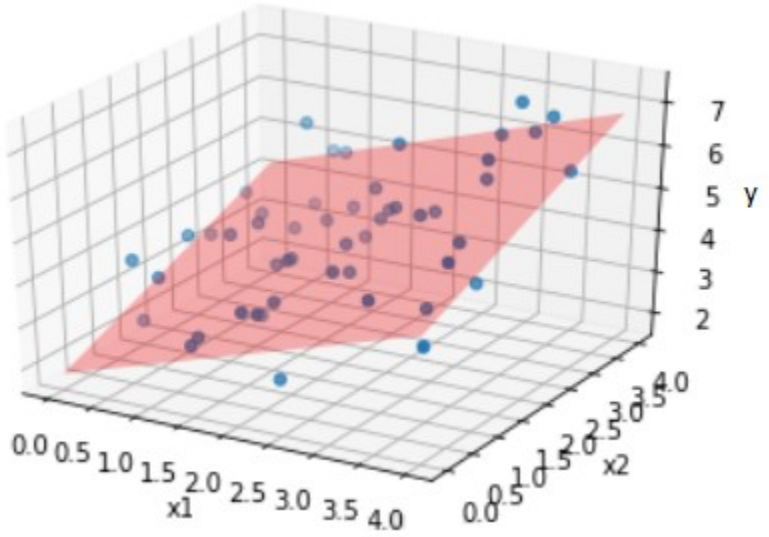
$$h(\boldsymbol{x};\boldsymbol{w}) = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + w_0$$

- dove $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]$ è il vettore dei parametri, detto anche vettore dei *pesi*
- Fissando il valore dei parametri in $\boldsymbol{w}$ ottengo una specifica funzione lineare
- In generale, qualsiasi sia il valore di $\boldsymbol{w}$, $h(\boldsymbol{x};\boldsymbol{w})$ corrisponde sempre ad un iperpiano nello spazio R^{d+1}

Esempio: caso *d = 2*

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

$h(\mathbf{x}) = h(x_1,x_2) = w_1 x_1 + w_2 x_2 + w_0$



Modelli parametrici

- La funzione ipotesi può essere anche non lineare. In quel caso avrà una forma differente da un iperpiano
- In generale, diremo che un modello di ML è *parametrico* quando si basa su una funzione ipotesi $h(\boldsymbol{x};\boldsymbol{w})$, dove:
  - $h()$ è una funzione «analitica», cioè (con un leggero abuso terminologico) è esprimibile tramite la composizione di funzioni analitiche di base note (polinomi, logaritmo, esponenziale, ecc.)
  - $\boldsymbol{w}$ è il ve determina la forma esatta
- Esempio: il m $h(\boldsymbol{x};\boldsymbol{w}) = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + w_0$ e *classe di funzioni*:

- fissando $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]$ ottengo una funzione specifica appartenente alla classe delle funzioni lineari
- I modelli parametrici sono probabilmente quelli più importanti nel ML e sono usati anche nei task di classificazione. Vediamo come

Classificazione binaria

Dataset di traini

$$T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(i)}, y^{(i)}), ..., (\boldsymbol{x}^{(n)}, y^{(n)})\}$$
$$y^{(i)} \in \{y_1, y_2\}$$
$$\boldsymbol{x}^{(i)} = (x_1^{(i)}, ..., x_j^{(i)}, ..., x_d^{(i)}) \in \mathbb{R}^d$$

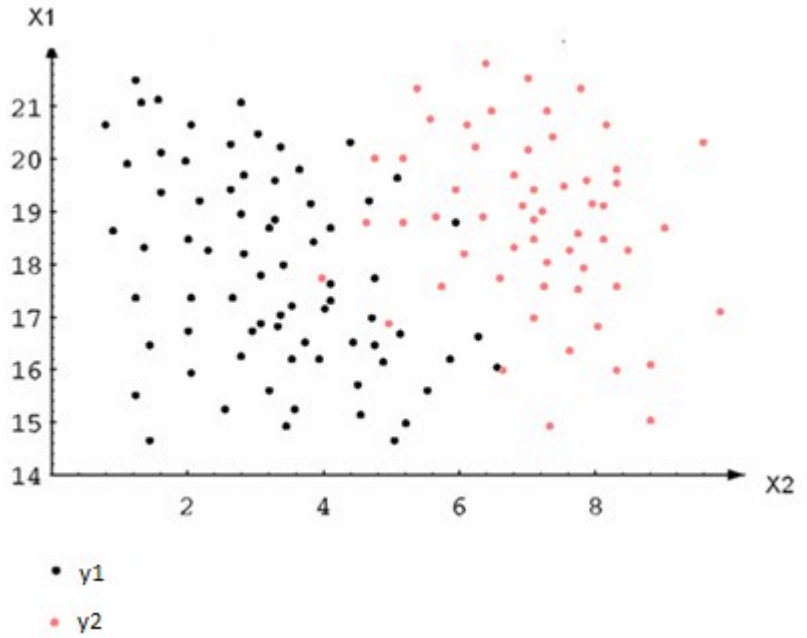
Task: vogliamo costruire un classificatore *C(**x**)* che, dato un nuovo esempio ***x***, mi dia la classe più probabile da associare ad ***x***: *y = C(**x**)*.

N.B.: in quest'esempio ***x*** è un feature vector numerico mentre *y* è una variabile categorica

Esempio ($d = 2$)

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

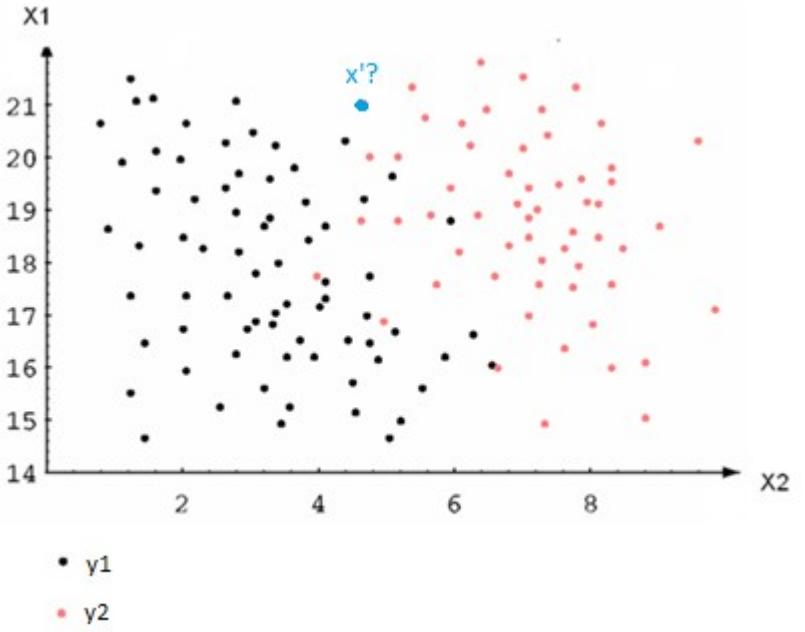
Rappresentazione grafica del dataset



Esempio

In fase di testing voglio predire:

$$y' = C(\boldsymbol{x}')$$

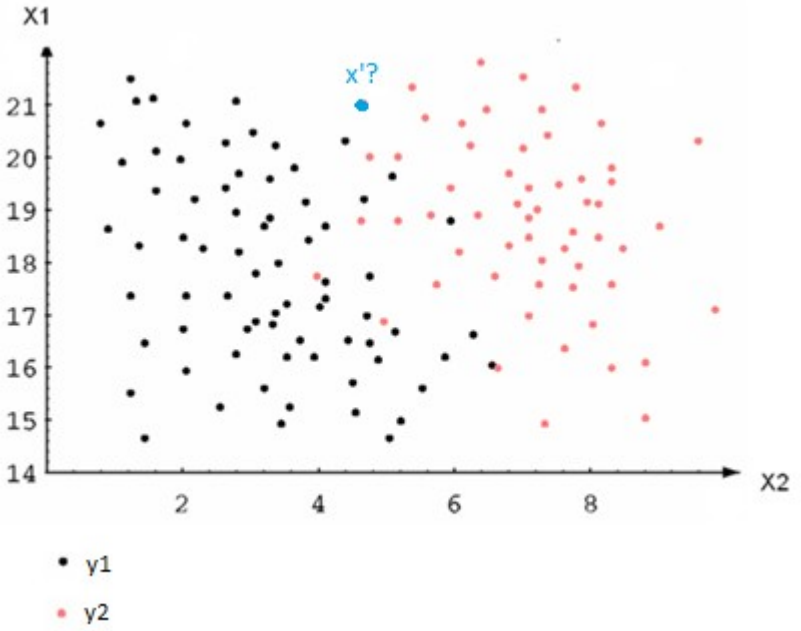


Esempio

Nell'esempio in figura, probabilmente

$y' = C(\boldsymbol{x}') = y_2$

Questo perchè assumo che la distanza nel feaure space *corrisponda ad una dissimilitudine* tra le classi rappresentate (e *x'* è più vicino a punti appartenenti a y_2)



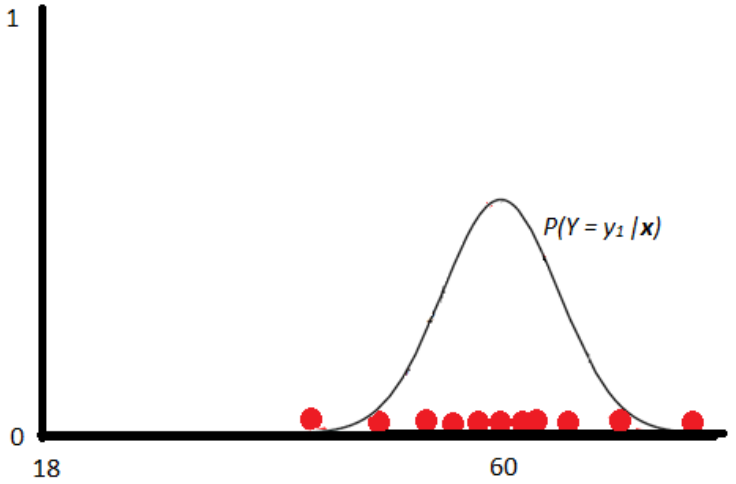
Stima della probabilità a posteriori

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Anche nel caso della classificazione l'obiettivo del ML sarà tipicamente trovare una funzione ipotesi $g()$ che approssimi un modello statistico
- Ad esempio, possiamo esprimere il modello statistico tramite la probabilità condizionale: $P(Y = y \mid X = \boldsymbol{x})$ (scritto sinteticamente con: $P(y|\boldsymbol{x})$)
- $g(\boldsymbol{x}) = P(Y = y_1|\boldsymbol{x})$ è la mia funzione ipotesi che deve essere indotta dai dati empirici in T

Esempio ($d = 1$)

- In figura accanto $g(x) = P(Y = y_1|x)$ rappresenta la probabilità che un generico elettore voti il partito dei pensionati (y_1) data la conoscenza della sua età x
- N.B.: x è una feature numerica
- I punti rossi rappresentano i feature value in T a cui è associata la label y_1 (nella figura mancano quelli che votano un altro partito, cioè quelli associati a y_2)



Stima della probabilità a posteriori

Dato che $g(\boldsymbol{x}) = P(Y = y_1|\boldsymbol{x})$, allora $g(\boldsymbol{x}) \in [0, 1]$

Per ora tralasciamo i dettagli su come *g()* possa essere fatta e limitiamoci ad osservare che è una funzione che prende in input un feature vector ***x*** e restituisce in output un valore di probabilità corrispondente alla stima che il modello fa circa la probabilità che ***x*** appartenga ad una delle due classi (e.g., la prima, y_1)

Devo ora convertire questo valore di probabilità in un valore categorico, effettuando una scelta binaria tra una delle due classi possibili

Per farlo uso una *regola di decisione* e il mio classificatore finale *C()* sarà dato da *g()* più questa regola

Ad esempio, *C()* potrebbe essere dato da:

- $C(\boldsymbol{x}) = y_1$ se $g(\boldsymbol{x}) > 0.5$
- y_2 altrimenti

Modello parametrico

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Come nel caso della regressione, per *g()* posso utilizzare un modello parametrico:

*g(**x**; **w**)*

Ad esempio, nel caso precedente ***w*** potrebbe essere dato dalla media e dalla varianza della Gaussiana: $\boldsymbol{w} = [\mu, \sigma^2]$

Anche per il task di classificazione dobbiamo trovare un modo per calcolare i valori di ***w*** (affronteremo questa cosa a breve)

Bordi di decisione

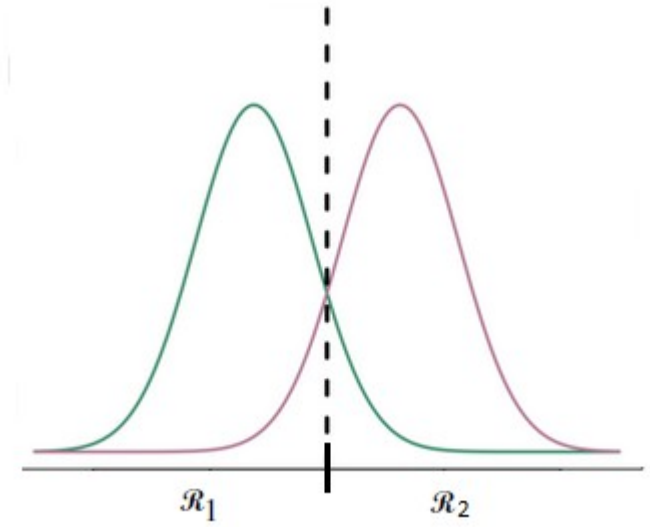
Se X è il feature space (e.g., $X = R^2$), allora le regioni di X in cui le due classi sono equiprobabili, ovvero:

$$\{\mathbf{x} : g(\mathbf{x}; \mathbf{w}) = P(y_1|\mathbf{x}) = 0.5 = P(y_2|\mathbf{x})\},$$

corrispondono ai *bordi di decisione* di $C()$

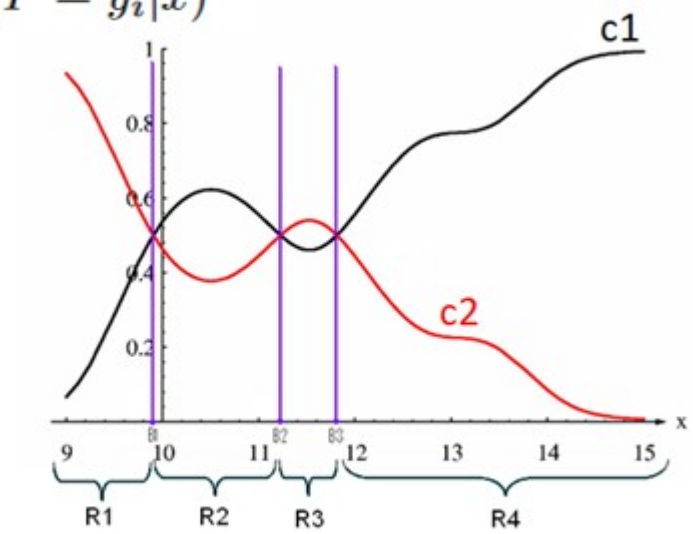
I *Decision boundaries* partizionano X in *decision regions*

Bordi di decisione: esempio 1D ($X = R$)



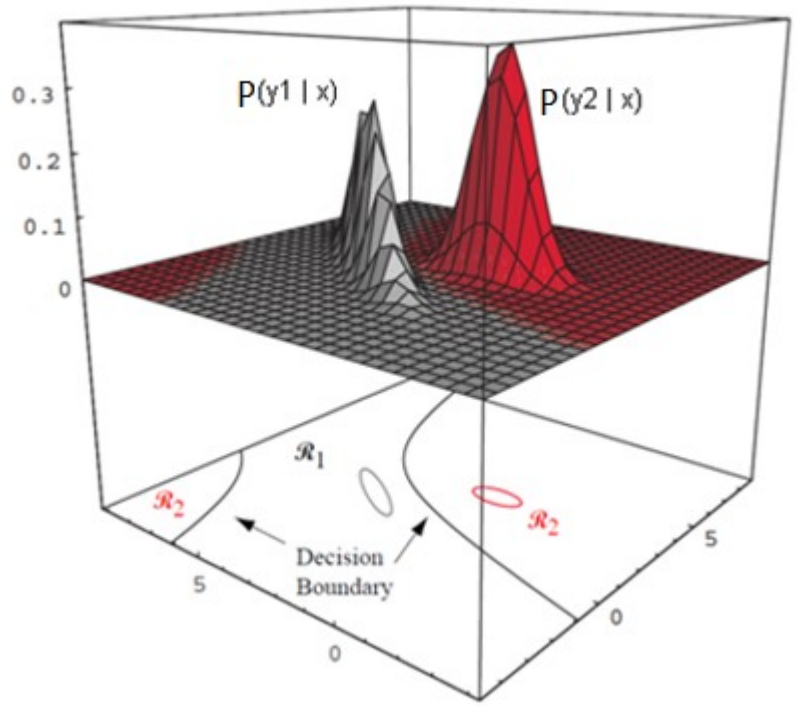
Bordi di decisione: altro esempio 1D (*X* = *R*)

$P(Y = y_i|x)$



$\mathcal{R}_2 = R_1 \cup R_3$, $\mathcal{R}_1 = R_2 \cup R_4$

Bordi di decisione: esempio 2D ($X = R^2$)



Ogni classificatore definisce dei bordi di decisione

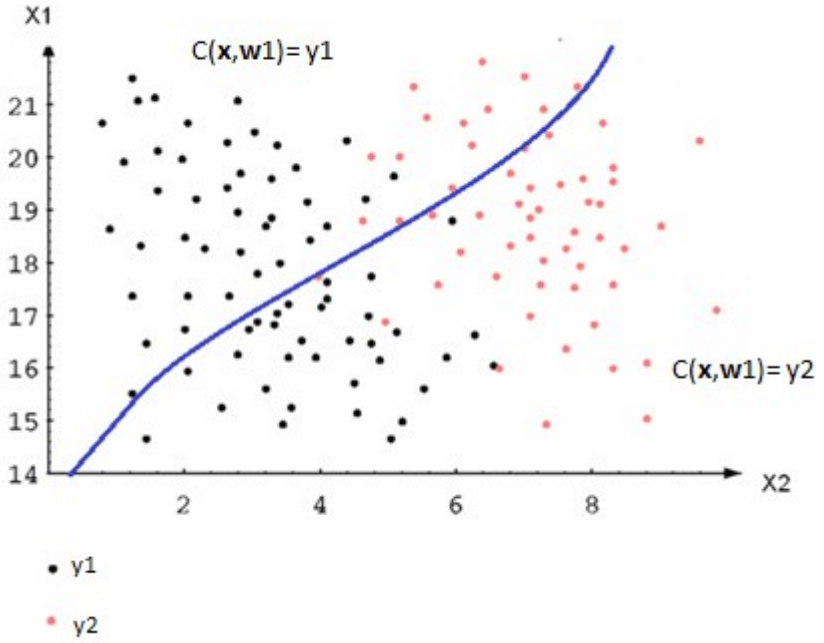
Anche se, generalmente, un classificatore *non* definisce *esplicitamente* i suoi bordi di decisione, modellando $P(y|\mathbf{x})$, *implicitamente* $g(\mathbf{x}; \mathbf{w})$ partiziona X in specifiche decision regions

Cambiando $\mathbf{w}$ ottengo classificatori diversi

Dato T, lo scopo è trovare un classificatore $C(\mathbf{x}; \mathbf{w})$ che partizioni X in decision regions che corrispondano effettivamente alle classi d'interesse

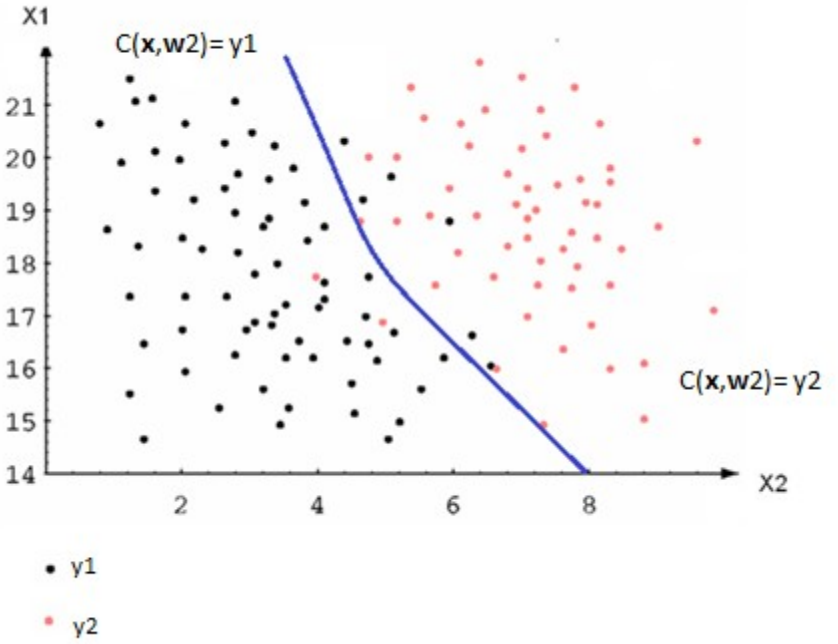
Esempio

$C(\boldsymbol{x}; \mathbf{w}_1)$



Esempio

$C(\boldsymbol{x}; \mathbf{w}_2)$



Ottimizzazione

Ottimizzazione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

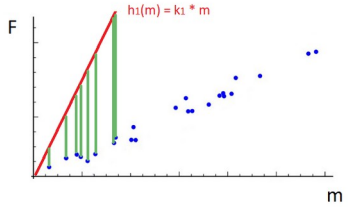
Abbiamo visto che, sia per task di regressione che per task di classificazione, posso spesso ricondurmi a funzioni ipotesi parametriche che dipendono da un vettore dei parametri $\boldsymbol{w}$

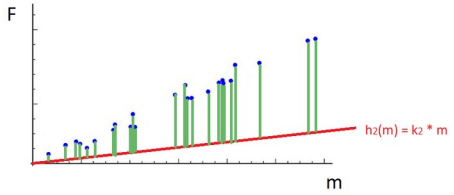
La fase di training, in questo caso, usando T, avrà il compito di stimare il vettore $\boldsymbol{w}$ "ottimale", ovvero il vettore di parametri che minimizza l'errore di predizione di quel modello rispetto a T

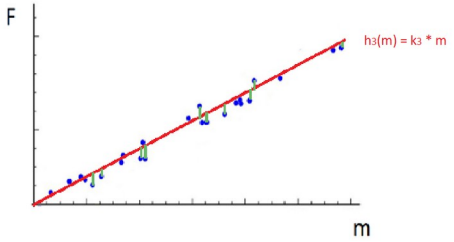
Ma andiamo con ordine…

Ottimizzazione

- Riprendiamo il caso «giocattolo» iniziale, quello della forza gravitazionale, in cui ci sono una serie di semplificazioni
- La funzione ipotesi che abbiamo scelto ha una sola variabile indipendente (*m*) e un solo parametro da stimare (*k*):
- *h(m) = m * k*
- *Ottimizzare* il valore di *k* significa trovare quello specifico valore per cui *h(m;k)* ha un errore di predizione minimo rispetto a *T*
- Come posso rendere operativa questa scelta?

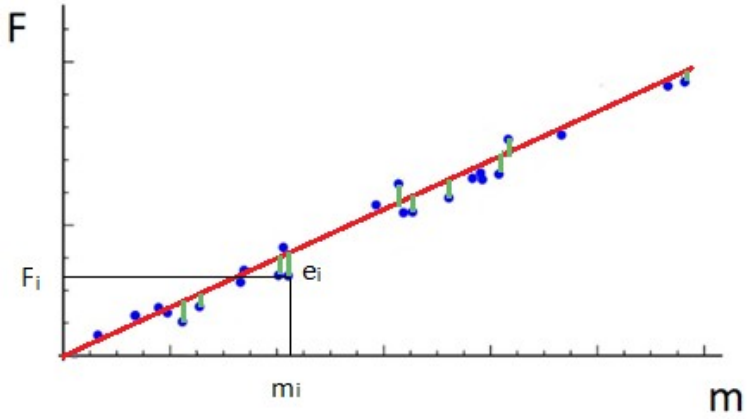






Ottimizzazione: calcolo dell'errore di predizione

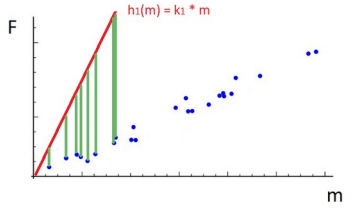
- Anzitutto devo quantificare l'errore totale rispetto a T
- Parto da un errore rispetto ad un singolo (generico) sample (m_i, F_i) in T, che sarà:
- $e_i = m_i * k - F_i$
- Attenzione: io ancora non conosco il valore di k, per cui, nell'espressione di sopra, lo indico con la *variabile* generica k, mentre m_i *ed* F_i sono *costanti* (perché sono valori noti)!

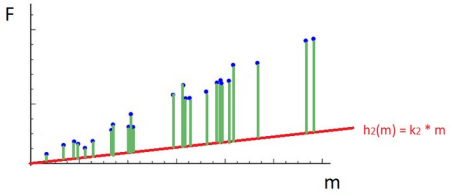


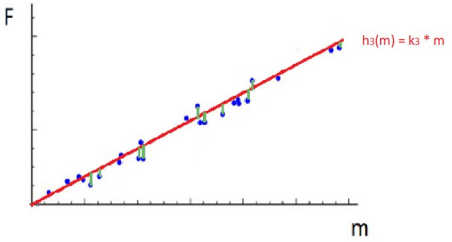
Ottimizzazione: calcolo dell'errore di predizione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- L'errore totale (L) di $h(m;k)$ rispetto a T sarà dato dalla somma di tutti gli e_i:
- $L = e_1 + \ldots + e_i + \ldots + e_n$
- L è detto *errore di training* o *rischio empirico*
- Io però non voglio che errori con segno positivo e negativo si cancellino, per cui è meglio avere una stima dell'errore che sia una somma di errori sempre positivi
- Ad esempio, potrei usare:
- $L = |e_1| + \ldots + |e_i| + \ldots + |e_n|$
- Esempio: *se* $T = \{(1, 9.81), (0.5, 4.9), \ldots\}$, allora
- $L = | k - 9.81 | + | 0.5\ k - 4.9 | + \ldots$
- Ottimizzare k significa trovare quel valore di k che minimizza L
- Come posso fare?







Ricerca esaustiva?

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Una prima idea potrebbe essere provare per tutti i possibili valori di k e calcolare l'errore L per ognuno di essi (*ricerca esaustiva*)

Impossibile: k è definito in R, che è uno spazio continuo

Anche assumendone una rappresentazione discreta (e limitata…), il numero dei tentativi sarebbe enorme

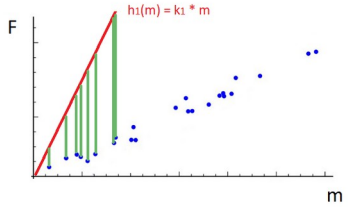
Potrei, ad esempio, limitare k in [-1000, +1000] e discretizzare quest'intervallo con passi di campionamento lunghi, e.g., 0.0001

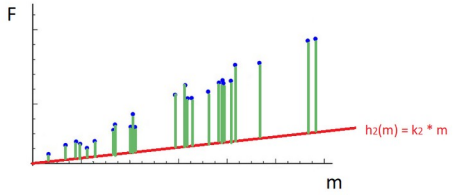
In totale dovrei fare $M = 2000 * 10000$ prove e, per ognuna di esse, dovrei calcolare n errori, con un costo computazionale di 20 miloni * n operazioni ($O(Mn)$)

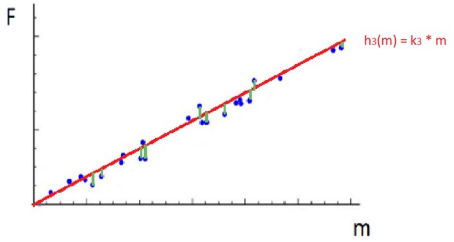
Quando, da un singolo parametro k, passeremo a *vettori* di parametri, vedremo che questo costo crescerebbe esponenzialmente!

Ottimizzazione: calcolo dell'errore di predizione

- Abbandonata l'idea della ricerca esaustiva, posso in realtà usare una strada molto più semplice, utilizzando gli strumenti di ricerca dei minimi di una funzione che avete già incontrato ad Analisi 1
- Per far ciò, osservo anzitutto che *L* è, in realtà, *una funzione di k*:
- $L(k) = |m_1 * k - F_1| + \ldots + |m_i * k - F_i| + \ldots + |m_n * k - F_n|$
- Esempio: $L(k) = |k - 9.81| + |0.5\,k - 4.9| + \ldots$
- Questa funzione, nel gergo del ML, si chiama *Loss function*
- Se *L(k)* fosse differenziabile, potrei cercare i punti dove la derivata prima si annulla: $L'(k) = 0$
- Seguendo quest'idea, decido di cambiare la definizione di errore e di sostituire il modulo con l'errore quadratico:
- $L(k) = (m_1 * k - F_1)^2 + \ldots + (m_i * k - F_i)^2 + \ldots + (m_n * k - F_n)^2$
- Adesso *L(k)* è differenziabile!

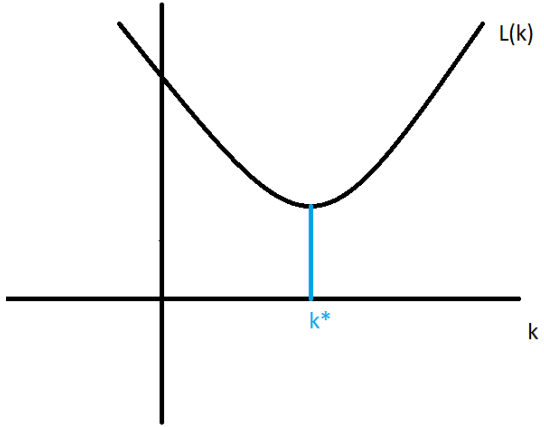






Ottimizzazione: calcolo dell'errore di predizione

- Definizione del problema:
  - Data la seguente funzione: $L(k) = (m_1 * k - F_1)^2 + \ldots + (m_i * k - F_i)^2 + \ldots + (m_n * k - F_n)^2$
  - Voglio trovare: $arg\ min_k\ L(k)$
- Il primo passo sarà calcolare la derivata prima di $L(k)$, che indico con $L'(k)$
- Poi pongo $L'(k) = 0$ e ottengo un'equazione con una sola incognita (k)
- Risolvendo rispetto a k, trovo dove $L'(k)$ si azzera. E.g., per un dato k^* ho che $L'(k^*) = 0$
- Per capire se k^* è un minimo, dovrei studiare la derivata seconda ($L''(k)$)
- In questo caso specifico, però, non è necessario, perché è possibile dimostrare che, se $L(k)$ è definita come sopra, allora $L''(k) > 0$ sempre (per ogni k in R)
- Ovvero, $L(k)$ è sempre una funzione convessa (indipendentemente dalla specifica funzione ipotesi $h(m)$,



Ottimizzazione: caso generale

- Problema risolto allora?
- Purtroppo no, perché, nel caso generale, avrò *un vettore di parametri* ***w*** e non un singolo parametro *k*
- Ad esempio, nel caso della regressione lineare con *d* feature (variabili), abbiamo visto che la funzione ipotesi è $h(\boldsymbol{x};\boldsymbol{w}) = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + w_0$

- dove $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]$ è un vettore nello spazio R^{d+1}
- Adesso l'errore quadratico rispetto ad un singolo (generico) sample $(\boldsymbol{x}^{(i)}, y^{(i)})$ in *T*, sarà:
- $e^{(i)} = ((w_1 x_1^{(i)} + \ldots + w_i x_i^{(i)} \ldots + w_d x_d^{(i)} + w_0) - y^{(i)})^2$
- E la loss functic

$$L(\boldsymbol{w}) = \sum_{i=1}^{n} e^{(i)} = \sum_{i=1}^{n} ((w_1 x_1^{(i)} + \ldots + w_j x_j^{(i)} + \ldots + w_d x_d^{(i)} + w_0) - y^{(i)})^2,$$

$$L : \mathbb{R}^{d+1} \rightarrow \mathbb{R}$$

Ottimizzazione: caso generale

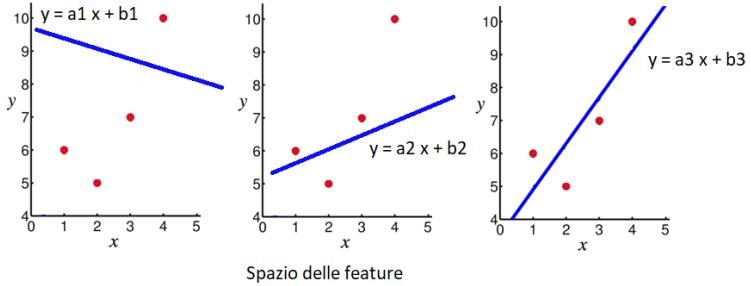
AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Quindi il problema è diventato quello di trovare il *vettore* ***w*** che minimizza *L(**w**)*:

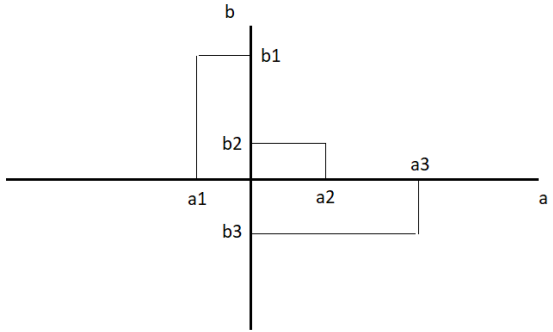
$$\boldsymbol{w}^* = \arg \min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} \sum_{i=1}^{n} e_i$$

- $W = R^{d+1}$ è lo *spazio dei parametri*
- Esiste una relazione biunivoca tra punti nello spazio dei parametri e funzioni ipotesi *h(**x**;**w**)*, ovvero: ad ogni punto ***w*** in $W = R^{d+1}$ corrisponde una specifica funzione *h(**x**;**w**)*
- Vediamolo in un caso semplice in cui *W* è bidimensionale e, quindi, può essere visualizzato

Spazio dei parametri in un caso bidimensionale



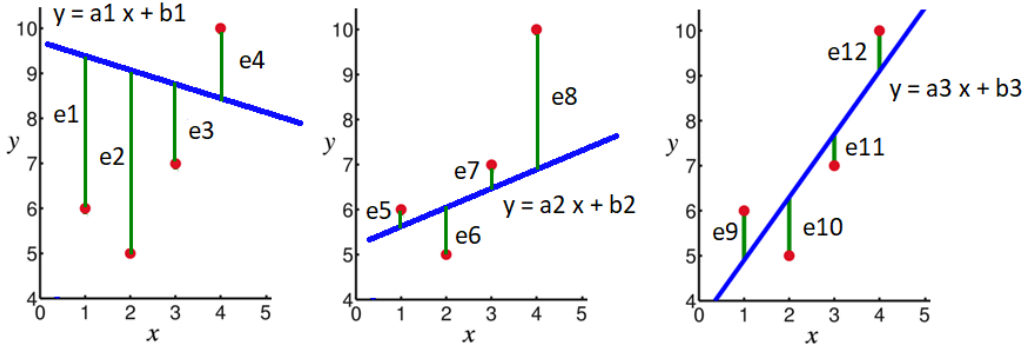
Spazio delle feature



Spazio dei parametri

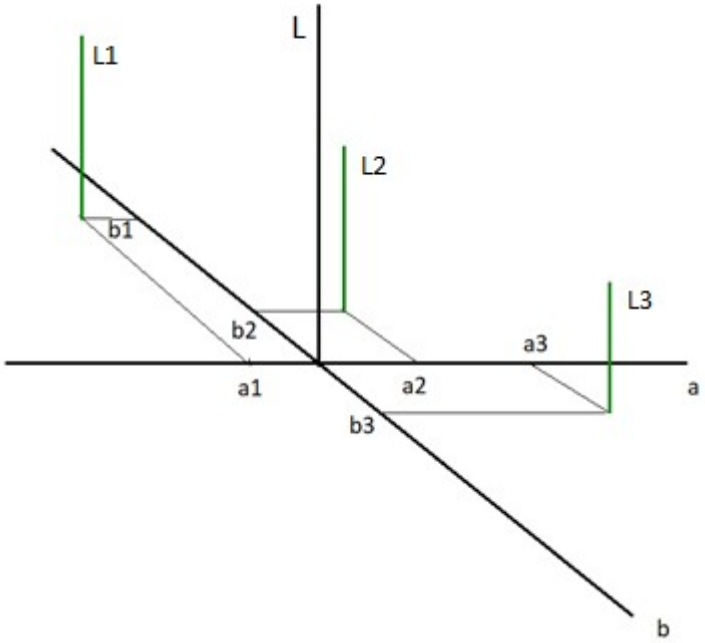
- Ad ogni funzione ipotesi del tipo $y = h(x; [a,b]) = ax + b$ posso associare un punto nello *spazio dei parametri* $W = R^2$ (e viceversa)
- Attenzione: lo spazio dei parametri ($W = R^2$) è diverso dallo spazio delle feature ($X = R$)
- Trovare la retta “migliore”, come sempre, equivale a trovare la coppia di valori $[a,b]$ in R^2 che minimizza l’errore di training

Spazio dei parametri in un caso bidimensionale



- Per $[a_1, b_1]$, ovvero, per $h(x) = a_1 x + b_1$, l'errore di training totale su T è dato da $L_1 = (e_1)^2 + (e_2)^2 + (e_3)^2 + (e_4)^2$
- Per $[a_2, b_2]$, ovvero per $h(x) = a_2 x + b_2$, l'errore di training totale su T è dato da $L_2 = (e_5)^2 + (e_6)^2 + (e_7)^2 + (e_8)^2$
- Per $[a_3, b_3]$, ovvero per $h(x) = a_3 x + b_3$, l'errore di training totale su T è dato da $L_3 = (e_9)^2 + (e_{10})^2 + (e_{11})^2 + (e_{12})^2$

Spazio dei parametri in un caso bidimensionale



Tra L_1, L_2 ed L_3 l'errore minore è L_3

In questo caso, quindi, voglio trovare la coppia di parametri [*a*,b**] tale che:

$$[a^*, b^*] = \arg \min_{[a,b] \in \mathbb{R}^2} \sum_{i=1}^{n} (y^{(i)} - (ax^{(i)} + b))^2$$

N.B.: la loss function *L([a,b])* è definita (in maniera continua) nello spazio dei parametri, mentre le funzioni ipotesi *h(x; [a,b])* sono definite nello spazio delle feature

Ricerca esaustiva

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Abbiamo già visto che, se $W = R$, allora la ricerca esaustiva è ($O(Mn)$)
(assumendo di dividere R in M passi di campionamento)

Se $W = R^2$, allora la ricerca esaustiva è ($O(M^2n)$)

In generale, se $W = R^{d+1}$, allora la ricerca esaustiva è ($O(M^{d+1}n)$)

Intrattabile…

Ritorniamo al caso generale

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

$$\boldsymbol{w}^* = \arg \min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} \sum_{i=1}^{n} e_i$$

Nel campo dell'ottimizzazione (usato in ML ma non solo in ML...) la loss function *L(**w**)* è detta *funzione obiettivo*

Un problema di ottimizzazione a più variabili consiste nel trovare il minimo (o, a volte, il massimo) della funzione obiettivo

Siccome *L(**w**)* non è una funzione monodimensionale, non posso cercarne il minimo usando "la derivata"

Per "ottimizzare" *L(**w**)* posso però estendere l'esempio monodimensionale visto prima usando delle tecniche analitiche che si basano sul *gradiente*

Soluzione del problema di ottimizzazione parametrico

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Se $L(\boldsymbol{w})$ è differenziabile, ovvero sono definite le derivate parziali rispetto a tutti gli assi di W, allora posso usare il gradiente per calcolare la soluzione del problema di minimizzazione, utilizzandolo:

- In maniera diretta
- O, più spesso, in maniera iterativa

Per semplificare la notazione, d'ora in poi assumeremo che $W = R^k$ (e.g., $k = d+1$)

Ricordiamoci che, calcolato in un punto di minimo (o di massimo) locale di $L(\boldsymbol{w})$, il gradiente sarà un vettore di k zeri: [0, …., 0]

Se, invece, il gradiente è calcolato in un punto generico di W (non un minimo o un massimo locale), allora il risultato sarà un vettore che punta nella direzione di massima crescita di $L(\boldsymbol{w})$

Soluzione del problema di ottimizzazione parametrico

Importante: quello che vedremo vale sia per task di classificazione che di regressione e, se non specificato diversamente, la funzione ipotesi può essere un modello parametrico qualsiasi, non necessariamente lineare

Prima di andare avanti, quindi, estendiamo il concetto di spazio dei parametri e di loss function al task di classificazione

Spazio dei parametri nel caso della classificazione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Nel caso della classificazione il discorso è analogo: supponiamo di avere un classificatore $C(\boldsymbol{x}; \boldsymbol{w})$ che dipende da una funzione parametrica $g(\boldsymbol{x}; \boldsymbol{w})$, dove:

- $\boldsymbol{x} = [x_1, \ldots, x_d]$ è un vettore di d feature che varia nello *spazio delle feature X*
- $\boldsymbol{w} = [w_1, \ldots, w_k]$ è un vettore di k parametri che varia nello *spazio dei parametri W*:

$$\boldsymbol{w} \in W = \mathbb{R}^k$$

Ottimizzazione dei parametri per la classificazione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Trovare il classificatore "migliore" $C(\boldsymbol{x}; \boldsymbol{w})$ rispetto al dataset di training T può essere formulato minimizzando il "rischio empirico" (o "training error"), ad esempio definito come segue:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in W} \frac{1}{n} \sum_{i=1}^{n} l(y^{(i)}, C(\boldsymbol{x}^{(i)}; \boldsymbol{w}))$$

$$l(y^{(i)}, \hat{y}^{(i)}) = \begin{cases} 1, & \text{if } y^{(i)} \neq \hat{y}^{(i)} \\ 0 & \text{otherwise.} \end{cases}$$

dove:

- $y^{(i)}$ è la ground truth associata ad $x^{(i)}$ in T,
- $\hat{y}^{(i)} = C(\boldsymbol{x}^{(i)}; \boldsymbol{w})$ è la classe predetta da $C()$ quando l'input è $\boldsymbol{x}^{(i)}$
- $l(y^{(i)}, \hat{y}^{(i)})$ è l'errore calcolato sul singolo sample

Loss function per la classificazione

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in W} \frac{1}{n} \sum_{i=1}^{n} l(y^{(i)}, C(\boldsymbol{x}^{(i)}; \boldsymbol{w}))$$

$$l(y^{(i)}, \hat{y}^{(i)}) = \begin{cases} 1, & \text{if } y^{(i)} \neq \hat{y}^{(i)} \\ 0 & \text{otherwise.} \end{cases}$$

$$L(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} l(y^{(i)}, C(\boldsymbol{x}^{(i)}; \boldsymbol{w}))$$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in W} L(\boldsymbol{w})$$

Attenzione: l'errore calcolato sul singolo sample non è una funzione differenziabile, per cui non lo sarà neanche *L(**w**)*

Se voglio usare metodi basati sul gradiente, dovrò trasformare la loss function qui sopra, in maniera analoga a ciò che ho fatto per l'errore di regressione (lo vedremo in un'altra lezione)!

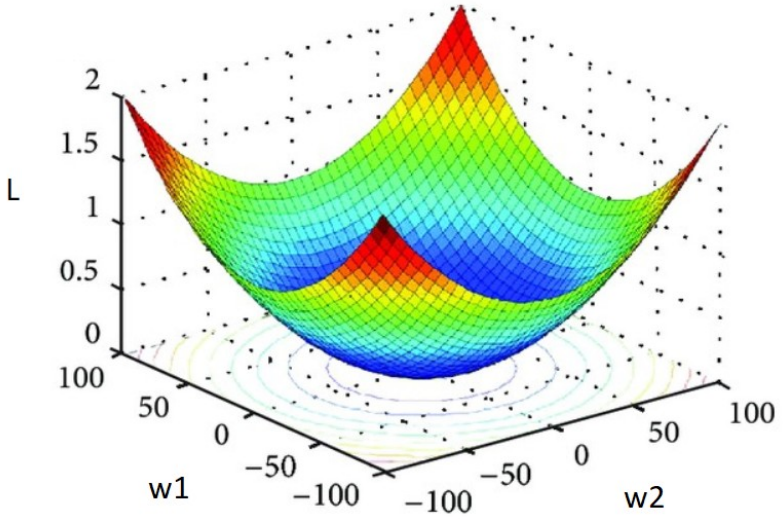
Soluzione analitica diretta («closed form solution»)

Se so che *L(**w**)* è *convessa*, ovvero esiste *un solo minimo globale*, allora posso calcolare il gradiente di *L(**w**)* e porlo a zero:

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = 0$$

Se *L(**w**)* è definita come la somma degli errori quadratici della linear regression, si può dimostrare che l'espressione di sopra è un sistema di *k* equazioni e *k* incognite (lo vedremo nei dettagli nella prossima lezione)

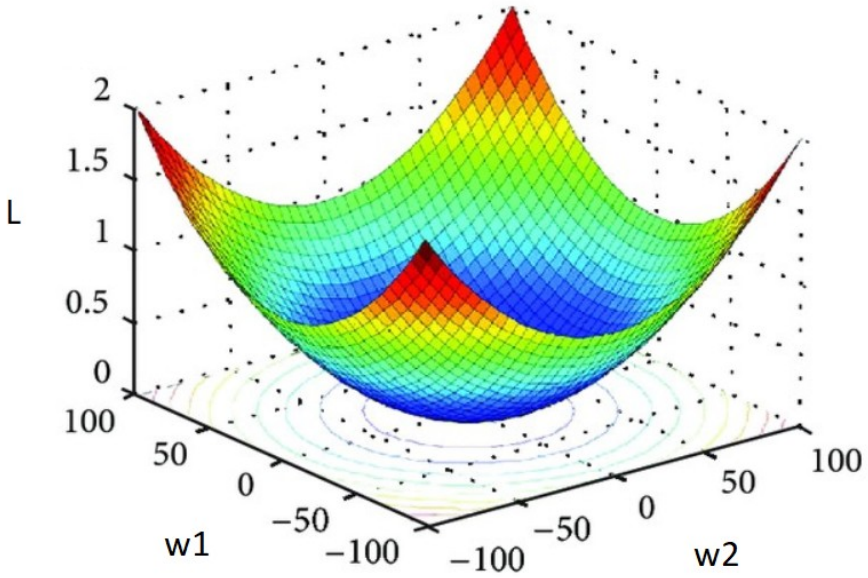
Inoltre, si tratta di un sistema lineare, per cui so come risolverlo rispetto alle *k* incognite $w_1, \ldots, w_k$



Esempio

Ma cosa facciamo se *L(**w**)* non è convessa?

Oppure se il sistema di equazioni non è lineare/non riesco a risolverlo analiticamente?

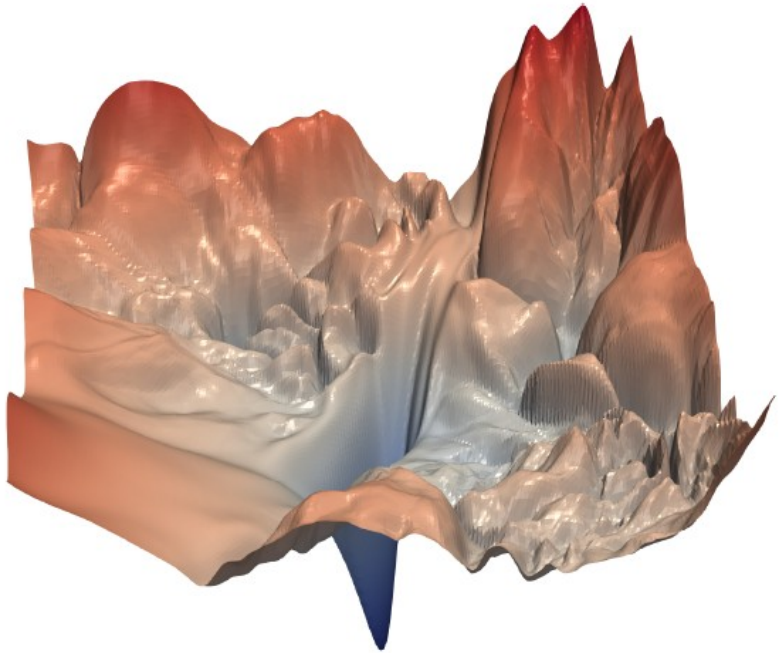


Esempio: loss function non convessa

La figura qui a fianco mostra la loss function reale di una rete neurale altamente non lineare, campionata intorno ad un punto di minimo “profondo”

Anche ammesso che riuscissi a risolvere il sistema di equazioni ottenuto ponendo a zero il gradiente, il numero di soluzioni ottenute sarebbe dello stesso ordine di grandezza di |W|

Quindi avremmo la stessa complessità computazionale della ricerca esaustiva…



Soluzione iterativa

Se *L()* non è convessa, in genere dobbiamo rinunciare all'idea di trovare il minimo globale

No panic: di solito è possibile trovare minimi *locali* che funzionano abbastanza bene

Per farlo, posso usare un algoritmo iterativo di tipo "greedy", il cui schema generale è:

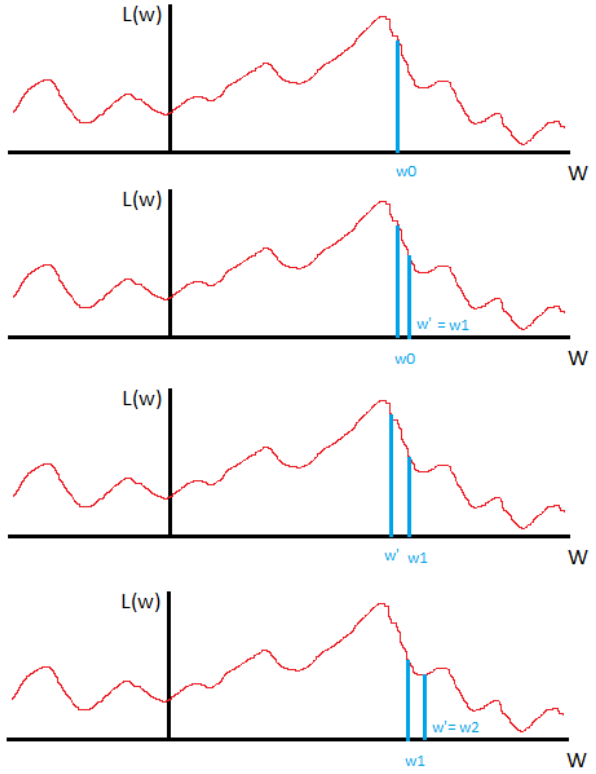
- Parto da una soluzione random, ovvero inizializzo ***w*** con dei numeri a caso
- Seguirà un ciclo in cui "perturbo" iterativamente ***w***, cioè lo modifico localmente seguendo un certo criterio che cerca di ottimizzare il *guadagno immediato* (da cui il nome greedy)

Vediamo un esempio

Esempio di algoritmo Greedy

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

1. Inizializzo $\boldsymbol{w}$ in maniera random, ottenendo $\boldsymbol{w}_0$; *t:= 0*
2. Iterazione t-esima: “perturbo” (e.g., in maniera random…) $\boldsymbol{w}_t$ localmente ottenendo $\boldsymbol{w}'$
3. Se $L(\boldsymbol{w}') < L(\boldsymbol{w}_t)$ allora $\boldsymbol{w}_{t+1} := \boldsymbol{w}'$, altrimenti torna al passo 2
4. *t:= t+1*
5. Se $L(\boldsymbol{w}_t)$ è abbastanza piccolo, allora return $\boldsymbol{w}_t$
6. Altrimenti vai al passo 2



Algoritmi Greedy

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Purtroppo, anche un algoritmo greedy come quello precedente può essere computazionalmente estremamente oneroso, soprattutto se k è grande (i.e., W ha molte dimensioni)

Questo perchè, ai passi 2 e 3, potrei dover fare un grosso numero di prove (potenzialmente, infinito) prima di riuscire a trovare un vettore $\boldsymbol{w}'$ tale che $L(\boldsymbol{w}') < L(\boldsymbol{w}_t)$

Ho bisogno di "perturbare" $\boldsymbol{w}_t$ in maniera intelligente, seguendo una direzione più razionale

Idee?

Uso il gradiente di $L()$ calcolato in $\boldsymbol{w}_t$ che mi "suggerirà" la direzione da seguire senza andare a caso…

Algoritmo Greedy *Gradient Descent*

Requisito: *L()* deve essere continua e differenziabile (ma va bene anche differenziabile “a pezzi”)

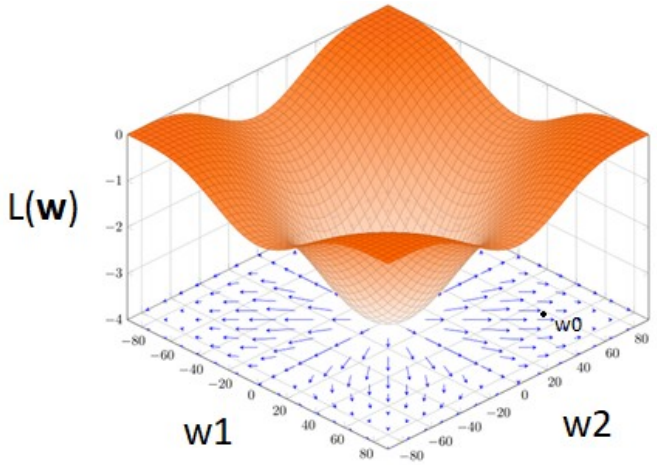
L’idea base è che il gradiente mi dà analiticamente la direzione (locale!) di massima crescita di una funzione

In queste condizioni posso usare un particolare (ma importante!) algoritmo greedy detto “Gradient Descent”

Gradient Descent

In linea generale, l'algoritmo di Gradient Descent è:

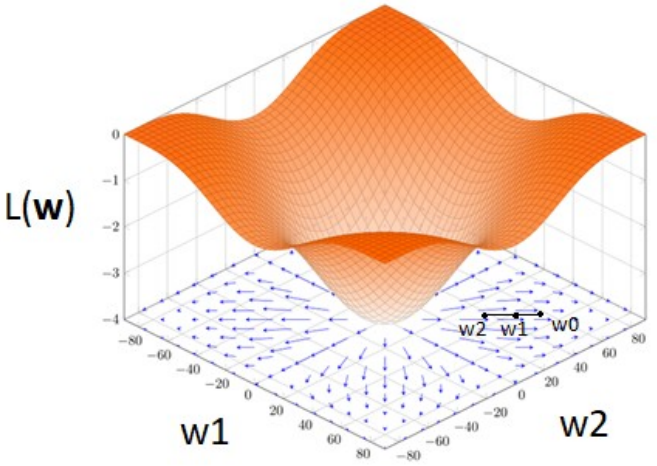
1. Inizializzo $\boldsymbol{w}$ in maniera random, ottenendo $\boldsymbol{w}_0$; $t := 0$



Gradient Descent

In linea generale, l'algoritmo di Gradient Descent è:

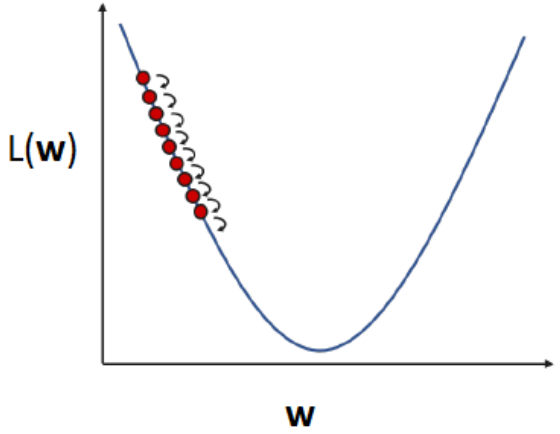
1. Inizializzo $\boldsymbol{w}$ in maniera random, ottenendo $\boldsymbol{w}_0$; *t:= 0*
2. Iterazione t-esima: $\boldsymbol{w}_{t+1} := \boldsymbol{w}_t - \alpha \nabla_{\boldsymbol{w}} L(\boldsymbol{w}_t)$
3. Se $L(\boldsymbol{w}_t)$ è "abbastanza piccolo", allora return $\boldsymbol{w}_t$
4. Altrimenti vai al passo 2



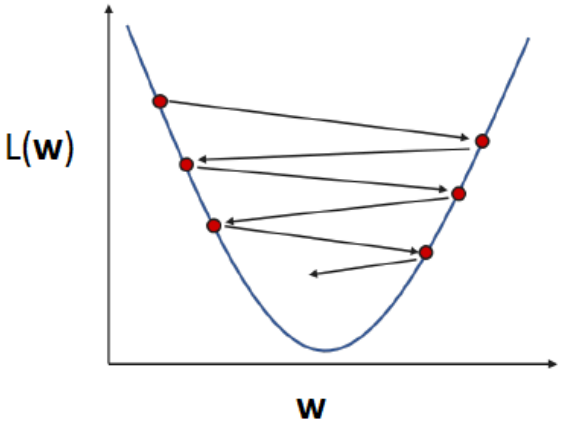
Al passo 2 *alpha* è il "learning rate", un *iperparametro* che decide la velocità di aggiornamento

Gli "iperparametri" sono quei parametri il cui valore non può essere trovato automaticamente dalla procedura di ottimizzazione ma deve essere fornito dal progettista facendo delle prove

Gradient Descent: learning rate



Learning rate troppo piccolo



Learning rate troppo grande

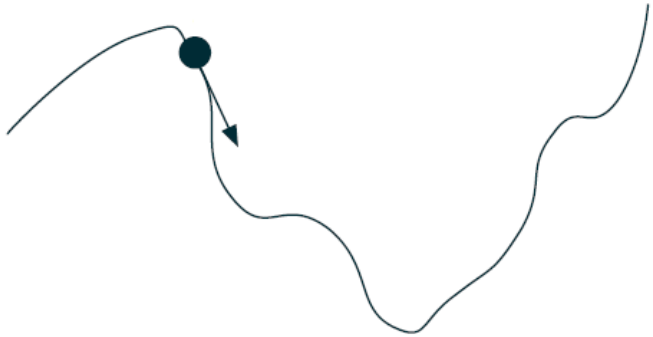
Gradient Descent

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

L() può essere altamente non convessa, con parecchi minimi locali non significativi

Per evitare continue oscillazioni, il Gradient Descent viene di solito usato insieme a varie “euristiche” di ottimizzazione, tipo il “momento”, che serve per mantenere una direzione usando una sorta di “forza d’inerzia”

Momentum



$$\boldsymbol{v}_{t+1} := \lambda \boldsymbol{v}_t - (1-\lambda)\alpha \nabla_{\boldsymbol{w}} L(\boldsymbol{w}_t), \quad \lambda \in [0,1]$$

$$\boldsymbol{w}_{t+1} := \boldsymbol{w}_t + \boldsymbol{v}_{t+1}$$

Il lambda che compare nell'espressione è un altro iperparametro

Gradient Descent

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Il Gradient Descent è in assoluto l'algoritmo di ottimizzazione più usato in ML

Usato congiuntamente al momentum o ad altre "euristiche", permette di raggiungere minimi locali sufficientemente "profondi" e di superare eventuali "collinette"

E… se invece *L()* non è differenziabile?

Lo vedremo tra poco, prima però ricapitoliamo quanto visto finora

Metodi parametrici: schema generale

Anzitutto definisco la mia funzione ipotesi in modo che sia dipendente da un vettore di parametri: $f(\mathbf{x};\mathbf{w})$

$f(\mathbf{x};\mathbf{w})$ è definita nello spazio delle feature e può avere una forma generica (non necessariamente lineare rispetto a $\mathbf{x}$): $f: R^d \rightarrow R$

L'espressione analitica di $f()$ definisce la *classe di modelli* che sto prendendo in considerazione. Ad ese., se $f()$ è lineare, sto restringendo le mie possibili soluzioni alla classe di funzioni lineari. In questa classe di modelli, una *specifica* funzione è definita specificandone i parametri $f(.; \mathbf{w})$

Devo poi calcolare una stima della "bontà" di ogni $f(.; \mathbf{w})$ su T, ad esempio definendo una *loss function* che dipende da $f()$ e, quindi, da $\mathbf{w}$: $L(\mathbf{w})$

$L(\mathbf{w})$ è definita nello spazio dei parametri: $L: R^k \rightarrow R$

A questo punto, voglio trovare un minimo di $L(\mathbf{w})$ rispetto a $\mathbf{w}$

Se $L(\mathbf{w})$ è differenziabile rispetto a $\mathbf{w}$, allora:

- Se $L(\mathbf{w})$ è convessa, posso usare un metodo diretto, ovvero:
  - Calcolo il gradiente di $L()$ rispetto a $\mathbf{w}$ e lo pongo a $\mathbf{0}$
  - Se il sistema di equazioni risultanti può essere risolto, trovo il minimo globale $\mathbf{w}^*$ "in un colpo solo"
- Se invece $L(\mathbf{w})$ non è convessa, posso usare un metodo iterativo basato sul Gradient Descent. In tal caso, dovrò accontentarmi di un minimo locale $\mathbf{w}^*$

Gradient-free solutions: cosa fare se $L(\theta)$ non è differenziabile

- Particle Swarm,
- Genetic algorithms,
- …

In genere si tratta di estensioni dell'algoritmo greedy visto prima che (ovviamente) *non* usano il gradiente e che tipicamente utilizzano un *insieme* di soluzioni (i.e., parameter vectors) modificate in maniera parallela

Intuitivamente, avere più soluzioni parallele che si evolvono in maniera indipendente permette di esplorare lo spazio delle soluzioni (i.e., spazio dei parametri) in maniera più ampia

Tipicamente sono computazionalmente molto onerosi

Overfitting

Errore di training uguale a 0?

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Abbiamo visto che, nel caso, e.g., della classificazione, l'obiettivo del training è trovare un minimo (globale o locale) **w*** della funzione di loss:

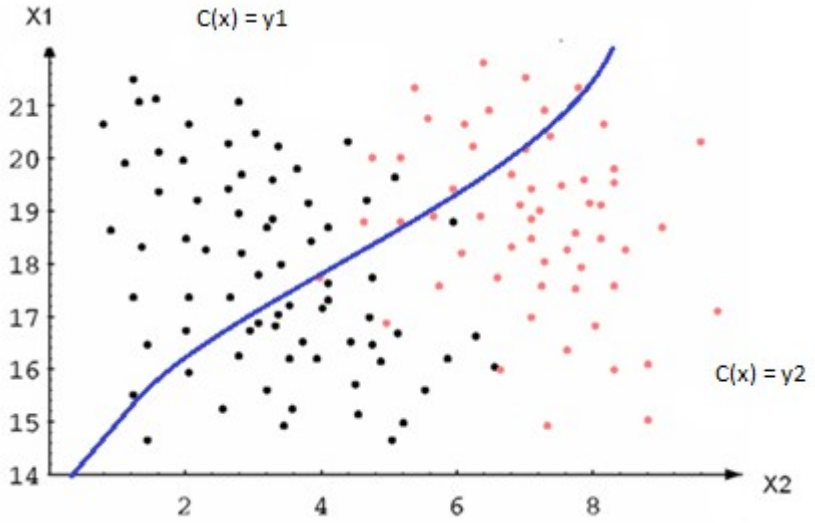
$$L(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} l(y^{(i)}, C(\boldsymbol{x}^{(i)}; \boldsymbol{w}))$$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in W} L(\boldsymbol{w})$$

Ma è desiderabile avere $L(\boldsymbol{w}^*) = 0$?

Esempio

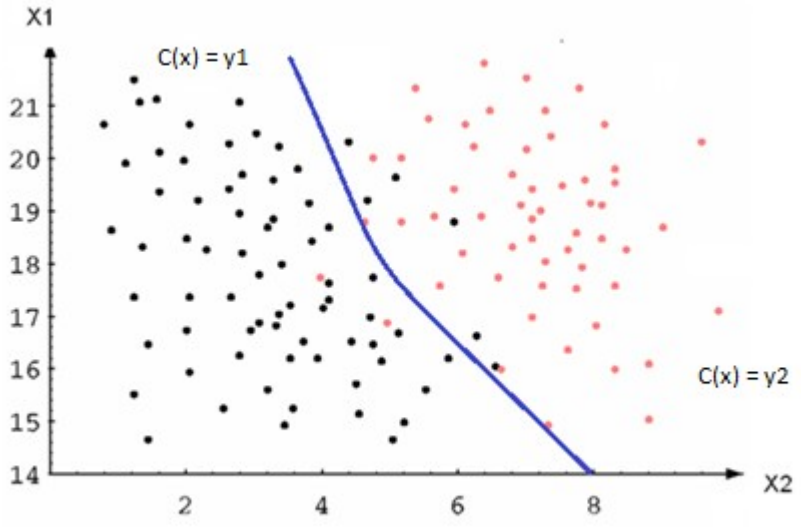
$C(.; \mathbf{w}_1)$



Esempio

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

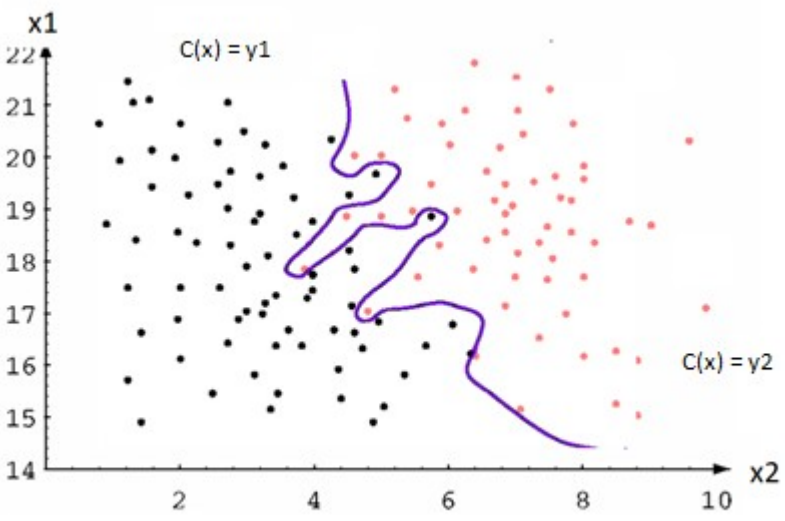
$C(.; \mathbf{w}_2)$



Esempio: errore di training = 0

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

$C(.; \mathbf{w}_3)$



Errore di training uguale a 0?

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Rasoio di Occam: “Entia non sunt multiplicanda praeter necessitatem"

Versione moderna: “Tra tutte le spiegazioni consistenti con le osservazioni date, l’ipotesi più semplice è quella più probabilmente vera”

In natura, le cose tendono ad essere semplici...

Perciò, classificatori/regresssori troppo complessi sono probabilmente una rappresentazione errata della realtà!

Il problema dell'Overfitting

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Un classificatore/regressore con un training error molto piccolo ma un grosso errore di predizione sul testing set è chiamato "overfitted"

Non riesce a *generalizzare* sui dati nuovi (quelli che arriveranno nella fase di inferenza), nonostante (e forse proprio perchè...) abbia "imparato a memoria" quelli di training

L'overfitting è un problema molto serio in Machine Learning

Il problema dell'Overfitting

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

L'Overfitting può dipendere da:

- La complessità eccessiva del modello parametrico scelto (e.g.: troppi parametri o troppe feature)
- Un numero insufficiente di training samples
- …
- Nelle prossime lezioni vedremo che, per quanto possa sembrare controintuitivo, spesso conviene avere un modello più semplice (e.g., lineare, oppure basato su meno feature) pur di evitare l'overfitting

Referimenti

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- **A tu per tu col Machine Learning. L'incredibile viaggio di un developer nel favoloso mondo della Data Science**, Alessandro Cucci, The Dot Company, 2017 [cap.4]

- **Pattern Classification, second edition,** R. O. Duda, P. E. Hart, D. G. Stork, Wiley-Interscience, 2000