



La Funzione di Costo e il Gradient Descent

Machine Learning Fundamentals

Fondamenti di Machine Learning per la Regressione Lineare

DATA ANALYTICS - ITS

Cos'è la Funzione di Costo?

La funzione di costo **J(w,b)** misura la differenza tra i valori previsti dal modello e i valori reali nei nostri dati. È lo strumento fondamentale che ci permette di valutare quanto il nostro modello sia accurato nelle sue previsioni.

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

m

Numero totale di esempi nel dataset

$\hat{y}(i)$

Valore previsto dal modello

y(i)

Valore reale osservato

1/2m

Fattore di normalizzazione

- ❑ **Obiettivo:** Trovare i valori di w e b che minimizzano J(w,b), ottenendo così il modello più accurato possibile.

Squared Error Cost Function

È la funzione più utilizzata per i problemi di regressione lineare perché penalizza maggiormente gli errori grandi, guidando il modello verso previsioni sempre più precise.

Esempio Pratico: Prezzo degli Immobili

Rivediamo come funziona la regressione lineare con un esempio concreto: prevedere il prezzo di un immobile in base alla sua superficie. Questa applicazione è tra le più comuni nel settore immobiliare.

Il Modello

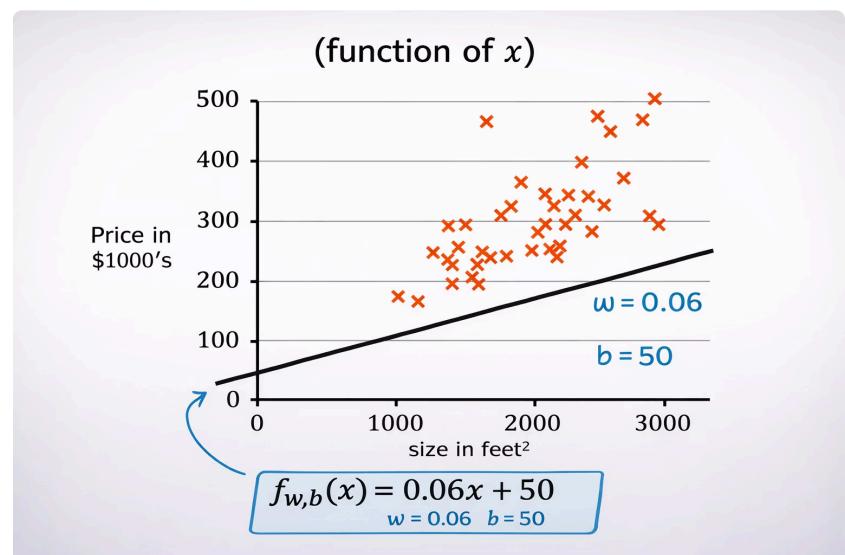
$$f(x) = 0.06x + 50$$

w = 0.06

Pendenza della retta (coefficiente angolare)

b = 50

Intercetta (valore iniziale)



Esempio di Calcolo

Appartamento Piccolo

$x = 1000 \text{ ft}^2$

$0.06(1000) + 50 = 110$

Prezzo stimato: **\$110.000**

Appartamento Grande

$x = 2000 \text{ ft}^2$

$0.06(2000) + 50 = 170$

Prezzo stimato: **\$170.000**

Come si può notare, ogni 1000 piedi quadrati aggiuntivi aggiungono circa \$60.000 al prezzo dell'immobile, mentre il valore base parte da \$50.000.

Visualizzare la Funzione di Costo in 2D

Caso semplificato con $b = 0$

Per comprendere meglio come funziona la funzione di costo, abbiamo visto in precedenza un caso semplificato dove abbiamo impostato l'intercetta b uguale a zero. Questo ci ha permesso di visualizzare la funzione in due dimensioni, rendendo i concetti più intuitivi.

01

Un solo parametro

Impostando $b=0$, la funzione di costo dipende solo da w . Possiamo scriverla come **$J(w)$** . Questa semplificazione ci permette di lavorare con un grafico bidimensionale facilmente interpretabile.

02

Forma a Parabola

Il grafico di $J(w)$ assume la forma caratteristica di una parabola, ovvero una curva a forma di U. Questa forma è fondamentale perché indica che esiste un unico punto di minimo.

03

Il Minimo

Il punto più basso della curva rappresenta il **costo minimo**. Questo corrisponde al valore ottimale di w , quello che produce le previsioni più accurate possibili per il nostro modello.

Grafico della funzione di costo $J(w)$ - La parabola mostra chiaramente come il costo varia al variare del parametro w , con un unico punto di minimo globale.

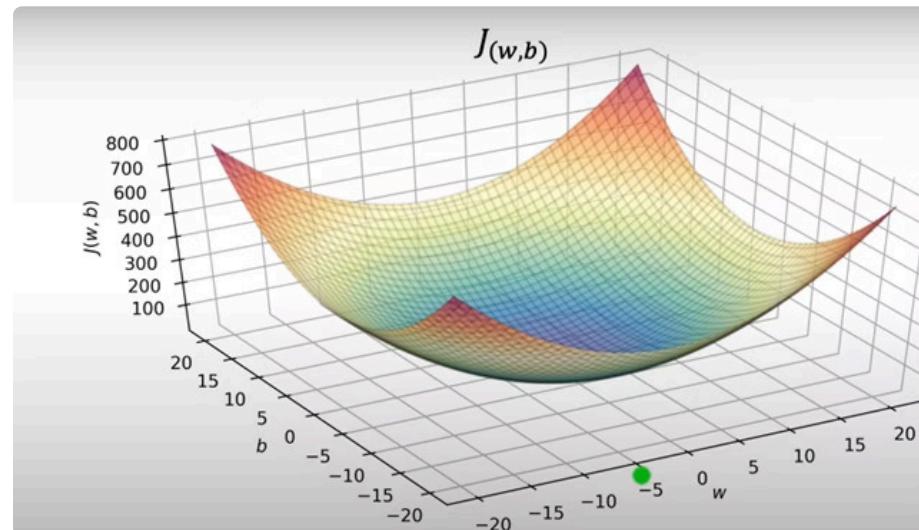
Visualizzare la Funzione di Costo in 3D

Il caso completo con w e b

Rappresentazione 3D

Quando consideriamo entrambi i parametri w e b simultaneamente, la funzione $J(w,b)$ si trasforma da una semplice curva bidimensionale a una superficie tridimensionale complessa. Questa visualizzazione ci mostra come il costo varia in funzione di entrambi i parametri contemporaneamente.

- **Asse x:** rappresenta il parametro w (pendenza)
- **Asse y:** rappresenta il parametro b (intercetta)
- **Asse z:** rappresenta il Costo $J(w,b)$



$J(w,b)$ Surface Plot

Bowl Shape (Ciotola)

La forma caratteristica a "ciotola" o "scodella" è cruciale: indica che esiste un unico punto di minimo globale. Questa proprietà geometrica garantisce che qualunque sia il nostro punto di partenza, l'algoritmo di ottimizzazione convergerà sempre verso lo stesso minimo ottimale.

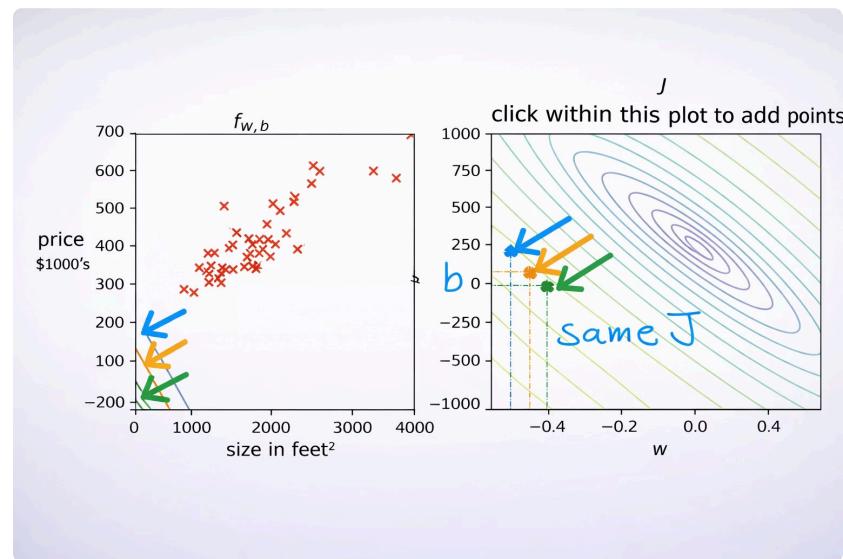
Obiettivo dell'Ottimizzazione

Il nostro obiettivo è trovare il punto più basso della superficie tridimensionale. Questo punto corrisponde alla combinazione ottimale di w e b che minimizza l'errore di predizione del modello su tutti i dati di training.

Il Contour Plot

Vista dall'alto della funzione di costo

Rappresentazione 2D



Il contour plot (o grafico a curve di livello) è una rappresentazione bidimensionale della superficie tridimensionale della funzione di costo. Immagina di guardare la "ciotola" dall'alto: ogni linea ellittica rappresenta punti con lo stesso valore di costo J .



Linee di Livello

Ogni ellisse indica tutti i punti (w,b) che producono lo stesso valore di J . Le ellissi più esterne corrispondono a costi elevati, quelle interne a costi minori.



Il Minimo

Il centro delle ellissi concentriche rappresenta il minimo globale della funzione di costo, il nostro obiettivo finale.



Pendenza

Linee vicine tra loro indicano una pendenza più ripida. Dove le curve sono distanziate, la superficie è più piatta.



"Come una mappa topografica vista dall'alto: ogni linea rappresenta la stessa altitudine sul terreno."

Esempi di Rette di Regressione

Caso 1: Una retta lontana dall'ottimo

Il Modello Errato

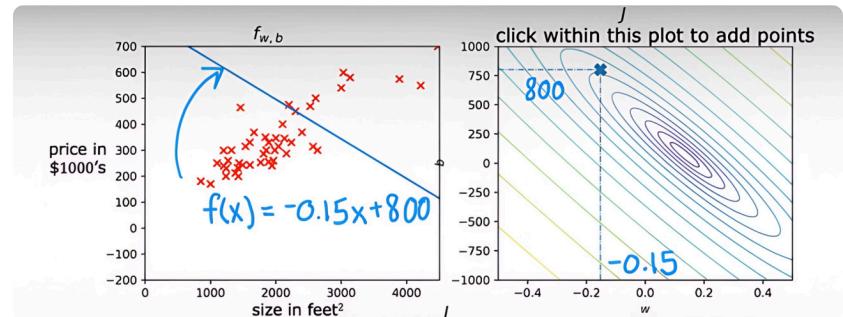
$$f(x) = -0.15x + 800$$

w = -0.15

Pendenza negativa

b = 800

Intercetta molto alta



Analisi degli Errori

Pendenza Negativa

La retta scende invece di salire, andando in direzione completamente opposta rispetto all'andamento naturale dei dati. Questo è un errore fondamentale che indica parametri molto lontani dall'ottimo.

Intercetta Troppo Alta

Il valore di partenza di 800 è esageratamente alto rispetto ai valori reali dei punti dati, causando errori sistematici enormi.

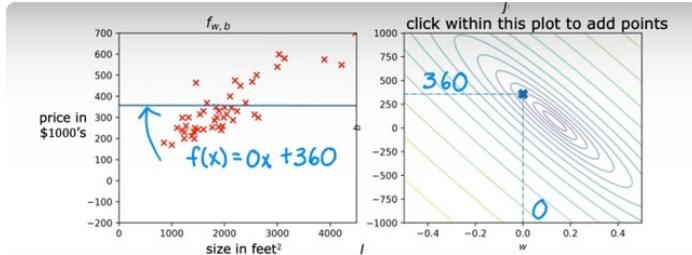
Costo Elevato

Sul contour plot, questa combinazione di parametri ci posiziona molto lontano dal centro (minimo). Il valore della funzione di costo $J(w,b)$ è estremamente alto, indicando predizioni molto imprecise.

Questo esempio dimostra quanto sia importante trovare i parametri corretti attraverso un processo di ottimizzazione sistematico.

Esempi di Rette di Regressione - Caso 2

Una retta migliore ma non ottimale



Il Modello

$$f(x) = 0x + 360$$

w = 0

Pendenza nulla

b = 360

Intercetta media

Analisi del Miglioramento

Questa configurazione rappresenta un passo avanti significativo rispetto al caso precedente. Vediamo perché:

Retta Orizzontale

Con pendenza w=0, la retta è perfettamente orizzontale e passa per y=360. Questo significa che il modello prevede sempre lo stesso valore, indipendentemente dall'input x. Non è ottimale, ma almeno non va nella direzione sbagliata.

Posizione sul Contour Plot

Il punto (w=0, b=360) sul contour plot è notevolmente più vicino al centro rispetto al Caso 1. Ci stiamo avvicinando al minimo della funzione di costo.

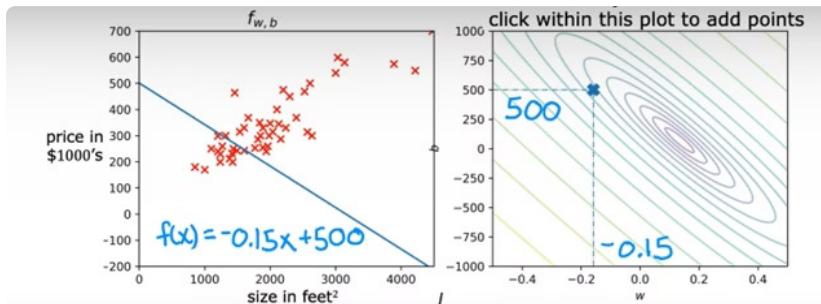
Funzione di Costo Ridotta

Il valore di $J(w, b)$ è significativamente più basso rispetto al caso precedente, ma ancora lontano dal minimo possibile.

- ☐ **Conclusione:** Stiamo migliorando nella direzione giusta, ma c'è ancora molta strada da fare per raggiungere la configurazione ottimale dei parametri.

Esempi di Rette di Regressione - Caso 3

Un passo indietro nel processo di ottimizzazione



Parametri del Modello

$$f(x) = -0.15x + 500$$

w = -0.15 (pendenza negativa)

b = 500 (intercetta media-alta)

Cosa È Andato Storto?

Pendenza Negativa Persistente

Ancora una volta la pendenza è negativa, mostrando una direzione opposta rispetto all'andamento naturale dei dati. Questo è un chiaro segnale che i parametri sono molto lontani dall'ottimo.

Peggioramento Rispetto al Caso 2

Rispetto alla configurazione precedente (Caso 2), ci siamo paradossalmente allontanati dal minimo invece di avvicinarci. Sul contour plot, questo punto è più distante dal centro.

Costo in Aumento

Il valore della funzione di costo $J(w,b)$ è aumentato rispetto al caso precedente, indicando che le nostre previsioni sono diventate meno accurate.

Lezione Importante: Non basta cambiare i parametri in modo casuale sperando di migliorare. Serve un metodo sistematico e matematicamente fondato per trovare l'ottimo. È qui che entra in gioco il Gradient Descent, l'algoritmo che ci guiderà in modo intelligente verso il minimo della funzione di costo.

La Retta Ottimale

Trovare il minimo della funzione di costo

★ BEST FIT

La Soluzione Perfetta

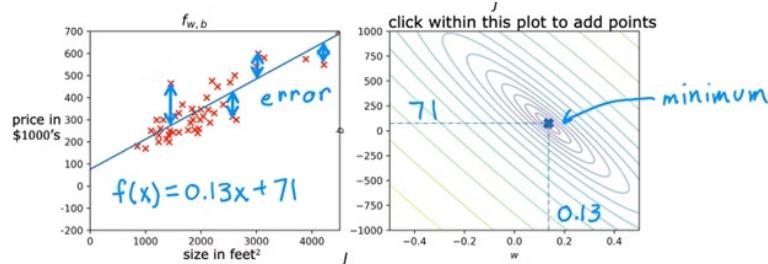
$$f(x) = 0.13x + 71$$

$w \approx 0.13$

Pendenza ottimale

$b \approx 71$

Intercetta ottimale



Caratteristiche dell'Ottimo

Minimo Errore Quadratico

La retta passa attraverso i dati in modo da minimizzare la somma delle distanze quadratiche tra i punti reali e le previsioni. Questo è esattamente ciò che ottimizza la funzione di costo MSE.

Centro del Target

Sul contour plot, il punto ($w=0.13, b=71$) si trova esattamente al centro delle ellissi concentriche. Abbiamo raggiunto il fondo della "ciotola" tridimensionale.

Valore Minimo di $J(w,b)$

Questa combinazione di parametri produce il valore più basso possibile della funzione di costo su tutto il dataset di training.

- **Il Problema Pratico:** Nei casi reali con migliaia o milioni di punti dati, non possiamo trovare questi valori ottimali "a occhio" o per tentativi. Serve un algoritmo automatico ed efficiente. È qui che entra in scena il Gradient Descent.

Introduzione al Gradient Descent

L'algoritmo che trova automaticamente l'ottimo

Il **Gradient Descent** (discesa del gradiente) è uno degli algoritmi di ottimizzazione più importanti e utilizzati nel Machine Learning. È il metodo che permette ai nostri modelli di "imparare" automaticamente i parametri ottimali dai dati.

Cos'è il Gradient Descent?

È un algoritmo iterativo che cerca il minimo di una funzione muovendosi nella direzione di massima discesa (il gradiente negativo). Invece di provare casualmente diversi valori dei parametri, usa le informazioni matematiche sulla pendenza della funzione per muoversi intelligentemente verso il minimo.

01

Inizializzazione

Inizializziamo i parametri con valori casuali o prestabiliti (es. $w=0$, $b=0$). Questo è il nostro punto di partenza sulla superficie della funzione di costo.

02

Iterazione

Aggiorniamo i parametri iterativamente, facendo piccoli passi nella direzione che riduce il costo. Ogni passo ci avvicina al minimo.

03

Convergenza

Continuiamo fino a quando i parametri non cambiano più significativamente, indicando che abbiamo raggiunto il minimo desiderato.

Perché È Importante?

Il Gradient Descent è fondamentale perché:

- Funziona automaticamente senza intervento manuale
- È computazionalmente efficiente anche con grandi dataset
- Si può applicare a funzioni di qualsiasi complessità
- È alla base dell'addestramento delle reti neurali

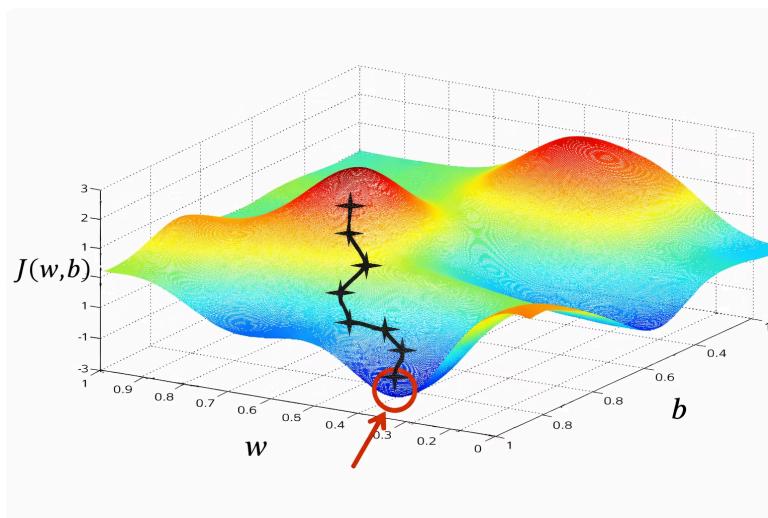
Metafora della Collina

Capire il Gradient Descent con un'immagine intuitiva

Per comprendere profondamente come funziona il Gradient Descent, usiamo una potente analogia visiva che rende il concetto matematico immediatamente comprensibile.

La Situazione

Immagina di essere sulla cima di una collina avvolta nella nebbia fitta. Non puoi vedere lontano, ma il tuo obiettivo è chiaro: raggiungere il punto più basso della valle sottostante nel modo più efficiente possibile. Come procedi?



La Strategia di Discesa

- Osservazione**
Guardati intorno a 360 gradi e valuta in quale direzione il terreno scende più ripidamente.
- Identificazione**
Trova la direzione dove la discesa è più ripida - questa è la direzione del gradiente (pendenza massima).
- Movimento**
Fai un piccolo passo controllato in quella direzione, senza correre il rischio di scivolare o saltare oltre la valle.
- Ripetizione**
Ripeti il processo: osserva, identifica, muoviti. Continua fino a quando non sei a valle (il terreno diventa piatto).

- Questo è esattamente ciò che fa l'algoritmo del Gradient Descent: calcola la pendenza (il gradiente) della funzione di costo, determina la direzione di massima discesa, e fa un passo controllato verso il basso. Ripete il processo iterativamente fino a raggiungere il minimo.

L'Algoritmo del Gradient Descent

Le formule matematiche che guidano l'ottimizzazione

Traduciamo la metafora della collina in formule matematiche precise.

L'algoritmo del Gradient Descent si basa su due equazioni fondamentali che aggiornano simultaneamente tutti i parametri del modello.

Le Formule di Aggiornamento

Queste equazioni vengono applicate ripetutamente, ad ogni iterazione dell'algoritmo, fino a quando i valori dei parametri si stabilizzano.

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

Componenti dell'Algoritmo



w, b - Parametri

I parametri del modello che stiamo ottimizzando. w rappresenta la pendenza della retta, b l'intercetta.



α - Learning Rate

Il tasso di apprendimento controlla la dimensione dei passi. È un iperparametro cruciale che determina quanto velocemente ci muoviamo verso il minimo.



$\frac{\partial J}{\partial}$ - Derivata Parziale

La derivata parziale della funzione di costo rispetto a ciascun parametro. Indica la direzione e la ripidità della pendenza in quel punto.

Logica Iterativa

L'algoritmo ripete l'aggiornamento dei parametri fino a quando non raggiunge la **convergenza**, ovvero quando i valori smettono di cambiare significativamente. A quel punto abbiamo trovato il minimo della funzione di costo.

Aggiornamento Simultaneo

Un dettaglio implementativo cruciale per il corretto funzionamento

La Regola d'Oro dell'Implementazione

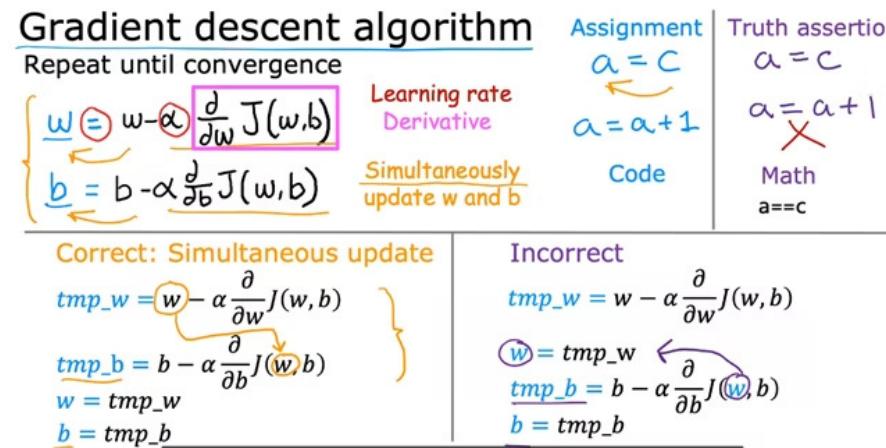
Quando implementiamo il Gradient Descent, c'è un aspetto fondamentale che può fare la differenza tra un algoritmo che funziona correttamente e uno che produce risultati errati.

Principio Fondamentale: Calcola i nuovi valori per **tutti** i parametri basandoti sui valori *attuali* (prima dell'aggiornamento), e *solo dopo* aver calcolato tutti i nuovi valori, aggiornali simultaneamente.

Perché È Importante?

Se aggiorniamo i parametri uno alla volta usando valori già modificati, stiamo di fatto cambiando le regole del gioco a metà iterazione. Questo porta a:

- Traiettoria di convergenza diversa e imprevedibile
- Possibile rallentamento della convergenza
- Risultati matematicamente non corretti
- Comportamento dell'algoritmo difficile da analizzare



Errore Comune

Se aggiorni prima w e poi usi il nuovo valore di w per calcolare l'aggiornamento di b , l'algoritmo non funzionerà correttamente secondo la teoria matematica del Gradient Descent.

Implementazione Corretta

1. Calcola $temp_w$ usando i valori attuali di w e b
2. Calcola $temp_b$ usando i valori attuali di w e b
3. Aggiorna: $w = temp_w$
4. Aggiorna: $b = temp_b$

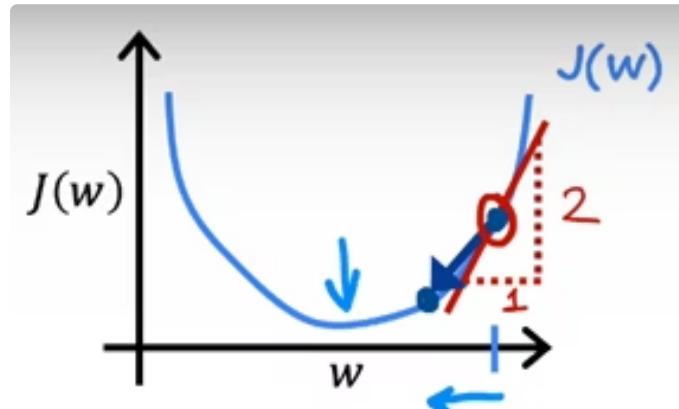
Il Ruolo della Derivata

Come la pendenza guida intelligentemente l'aggiornamento dei parametri

La derivata è il cuore matematico del Gradient Descent. Comprendere come funziona ci rivela la vera eleganza dell'algoritmo: la matematica stessa ci spinge automaticamente verso il minimo, indipendentemente da dove partiamo.

Caso 1: Pendenza Positiva

Immaginiamo di trovarci in un punto **a destra del minimo** sulla curva della funzione di costo.



In questo punto, la derivata è **positiva** (>0) perché la curva sta salendo verso destra.

La formula $w - \alpha(\text{positivo})$ esegue una sottrazione, facendo **diminuire** il valore di w .

Diminuendo w , ci spostiamo verso **sinistra** sulla curva, avvicinandoci al minimo.

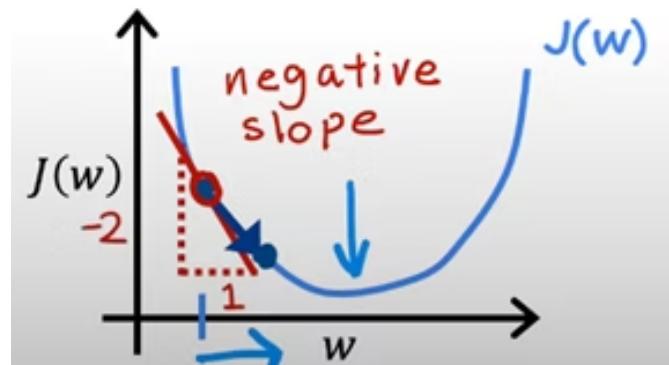
Caso 2: Pendenza Negativa

Ora immaginiamo di trovarci in un punto **a sinistra del minimo** sulla curva.

In questo punto, la derivata è **negativa** (<0) perché la curva sta scendendo verso destra.

La formula $w - \alpha(\text{negativo})$ diventa effettivamente un'**addizione**: sottrarre un numero negativo significa sommare.

Aumentando w , ci spostiamo verso **destra** sulla curva, ancora una volta avvicinandoci al minimo.



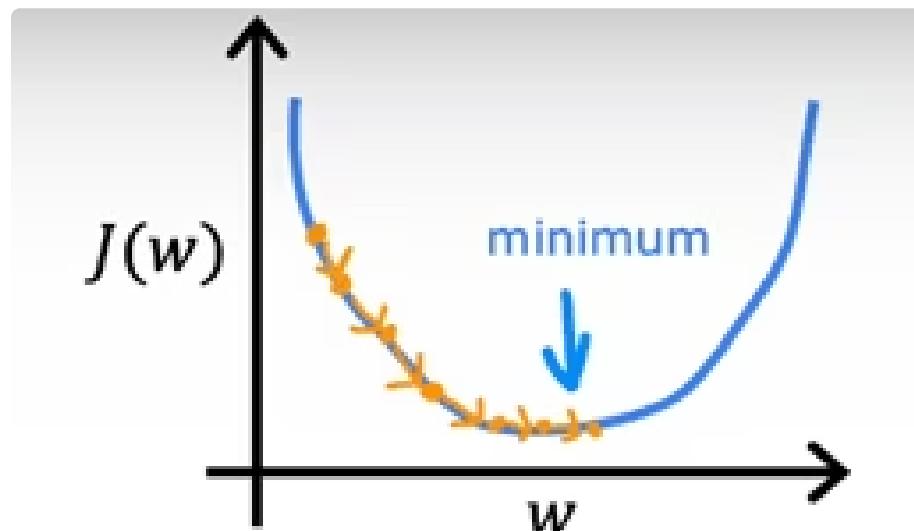
La Magia della Matematica: In entrambi i casi, indipendentemente da quale lato del minimo ci troviamo, la matematica ci spinge automaticamente nella direzione corretta verso il fondo della "ciotola". Non dobbiamo dire all'algoritmo se andare a sinistra o a destra: lo calcola da solo attraverso il segno della derivata.

Il Learning Rate α - Troppo Piccolo

Quando la prudenza diventa lentezza controproducente

Il Problema della Velocità

Se scegliamo un learning rate α estremamente piccolo (ad esempio 0.000001), ogni aggiornamento dei parametri sarà minuscolo. È come cercare di scendere da una montagna facendo passi da formica: tecnicamente funziona, ma richiede un tempo proibitivo.



Conseguenze Pratiche

Lentezza Estrema

L'algoritmo fa "passi da formica" verso il minimo. Ogni iterazione produce un miglioramento quasi impercettibile nella funzione di costo.

Costo Computazionale

Serviranno moltissime iterazioni (potenzialmente migliaia o milioni) per convergere al minimo, sprecando tempo prezioso e risorse computazionali.

Convergenza Garantita

Il lato positivo: con una funzione convessa, arriverà comunque al minimo globale. È solo una questione di pazienza (molta pazienza!).

Quando Potrebbe Essere Accettabile

Un α piccolo potrebbe essere giustificato solo in situazioni molto specifiche, come quando ci troviamo già molto vicini al minimo e vogliamo fare aggiustamenti finissimi senza rischiare di saltare oltre.

Il Learning Rate α - Troppo Grande

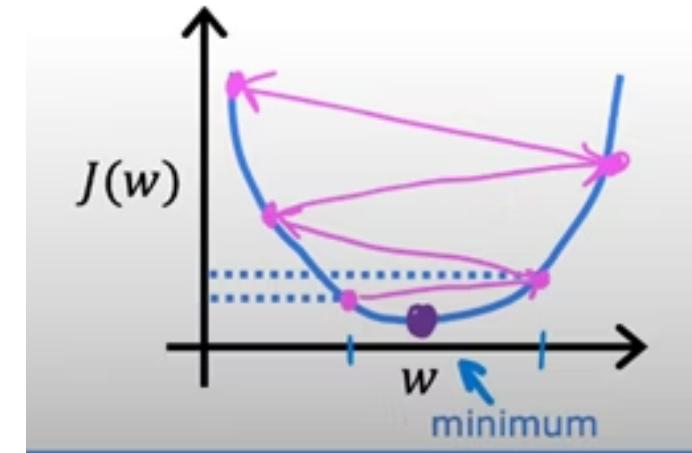
Quando l'impazienza porta alla divergenza catastrofica

All'estremo opposto dello spettro, un learning rate troppo grande può trasformare il nostro algoritmo di ottimizzazione in un processo caotico e controproducente. Invece di convergere verso il minimo, rischiamo di allontanarcene sempre di più.

Il Fenomeno dell'Overshoot

Immagina di scendere da una collina facendo passi giganteschi. Se il passo è troppo lungo, potresti superare completamente la valle e ritrovarti dall'altra parte, magari anche più in alto di prima. Questo è esattamente ciò che accade matematicamente con un α eccessivo.

- 1 **Salto Oltre il Minimo**
Il passo di aggiornamento è così grande che superiamo il punto ottimale, finendo dall'altra parte della "ciotola".
- 2 **Oscillazione Crescente**
Ad ogni iterazione, continuamo a saltare da una parte all'altra, con oscillazioni sempre più ampie.
- 3 **Divergenza Totale**
La funzione di costo **aumenta** invece di diminuire, allontanandoci sempre più dal minimo.



Segnali di Allarme

$J(w,b)$ Cresce

Se monitori la funzione di costo e vedi che aumenta invece di diminuire, α è quasi certamente troppo grande.

Valori Esplosivi

I parametri possono assumere valori sempre più grandi in valore assoluto, portando a overflow numerici.

NaN nel Codice

Nei casi estremi, i calcoli possono produrre "Not a Number" (NaN), segnalando un'instabilità numerica grave.

Trovare il Learning Rate Giusto

Strategie pratiche di debugging e tuning per ottimizzare α

Strategia 1: Monitorare la Convergenza

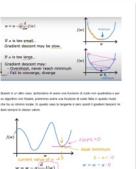
Il modo più efficace per valutare se il learning rate è appropriato è visualizzare graficamente l'andamento della funzione di costo nel tempo.

Grafico J vs Iterazioni

Disegna un grafico che mostra il valore di $J(\mathbf{w}, \mathbf{b})$ sull'asse verticale e il numero di iterazioni sull'asse orizzontale.

Interpretazione

Se J sale, α è troppo grande. Se J scende troppo lentamente (curva quasi piatta), α è troppo piccolo. L'ideale è una curva che decresce rapidamente all'inizio e poi si stabilizza.



Strategia 2: Valori di Prova

Un approccio sistematico consiste nel testare una sequenza di valori di α in scala logaritmica, per trovare l'ordine di grandezza corretto.

Sequenza Consigliata

Prova valori che differiscono di un fattore 3 o 10:

0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1

Confronto

Per ogni valore, allena il modello per un numero fisso di iterazioni e confronta le curve di convergenza.

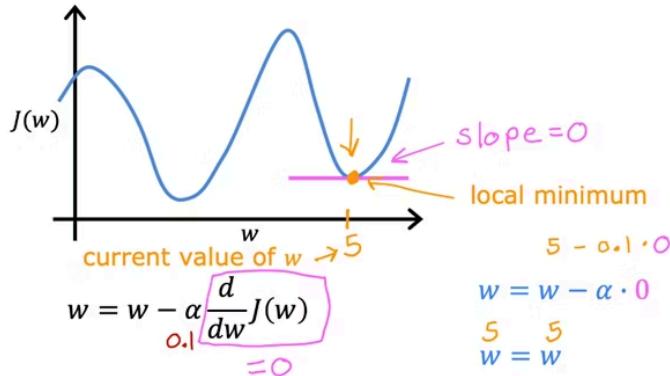
Selezione

Scegli il valore più grande che ancora produce una convergenza stabile e rapida.

☐ **Obiettivo:** Una curva di J che decresce rapidamente nelle prime iterazioni e poi si stabilizza dolcemente, indicando che abbiamo raggiunto il minimo.

Caso Speciale - Minimo Locale

Cosa succede quando siamo già arrivati alla destinazione?



Implicazioni Matematiche

01

Pendenza Zero

Quando siamo esattamente al minimo, la derivata (la pendenza) della funzione è esattamente uguale a zero: $\partial J / \partial w = 0$

02

Aggiornamento Nullo

Applicando la formula di aggiornamento del Gradient Descent:

$$w = w - \alpha \cdot 0 \Rightarrow w = w$$

Il nuovo valore di w è identico al vecchio valore!

03

Convergenza Raggiunta

Una volta raggiunto il minimo, il Gradient Descent non modificherà più i parametri. Questo è il segnale matematico che l'ottimizzazione è completa.

Questa proprietà elegante significa che non dobbiamo preoccuparci di "superare" il minimo una volta raggiunto. L'algoritmo si ferma naturalmente nel punto ottimale.

La Matematica del Minimo

In un minimo locale (o globale), la superficie della funzione di costo è perfettamente piatta. Dal punto di vista geometrico, la retta tangente alla curva in quel punto è orizzontale.

Stabilizzazione Automatica

Il parametro smette di cambiare completamente. L'algoritmo si stabilizza automaticamente senza bisogno di alcun criterio di stop esplicito.

Convergenza Automatica

Perché non serve ridurre α manualmente durante l'addestramento

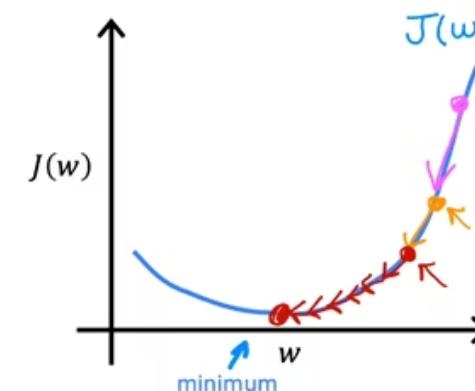
Una delle caratteristiche più eleganti del Gradient Descent è la sua capacità di "frenarsi" automaticamente mentre si avvicina al minimo. Questo comportamento emerge naturalmente dalla matematica dell'algoritmo, senza richiedere alcun intervento manuale.

Can reach local minimum with fixed learning rate α

$$w = w - \alpha \frac{d}{dw} J(w)$$

smaller
not as large
large

- Near a local minimum,
- Derivative becomes smaller
 - Update steps become smaller



Avvicinamento al Minimo

Man mano che i parametri si avvicinano ai valori ottimali, ci troviamo in regioni della superficie della funzione di costo sempre più piatte. La "ciotola" diventa meno ripida avvicinandosi al fondo.

Passi Automaticamente più Piccoli

Poiché la dimensione del passo è Passo = $\alpha \times$ Derivata, anche mantenendo α costante, i passi effettivi si riducono naturalmente. Il prodotto diventa sempre più piccolo man mano che la derivata tende a zero.

Conclusione Pratica: Nella maggior parte dei casi, puoi mantenere α costante per tutta la durata dell'addestramento. Tecniche più avanzate come il "learning rate decay" esistono, ma non sono necessarie per la convergenza di base del Gradient Descent su funzioni convesse come quella della regressione lineare.

Derivata Minore

La pendenza della funzione (la derivata) diminuisce progressivamente. Matematicamente, $|\partial J / \partial w|$ diventa sempre più piccolo, tendendo a zero al minimo.

Frenata Naturale

L'algoritmo "frena" da solo mentre arriva al minimo, senza bisogno di modificare α . È come una macchina che rallenta automaticamente avvicinandosi al semaforo rosso.

Gradient Descent per la Regressione Lineare

Inserire le derivate specifiche nell'algoritmo generale

Finora abbiamo parlato del Gradient Descent in termini generali, usando il simbolo generico ∂J per indicare la derivata. Ora è il momento di essere specifici: quali sono esattamente le formule per la regressione lineare?

Dal Generale allo Specifico

Per applicare l'algoritmo generale al nostro modello di regressione lineare con funzione di costo MSE (Mean Squared Error), dobbiamo calcolare le **derivate parziali** della funzione di costo rispetto a ciascun parametro.

Linear regression model

$$f_{w,b}(x) = wx + b \quad J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b) \rightarrow \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b) \rightarrow \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

}

Le Formule Specifiche

Derivata rispetto a w

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Questa formula calcola quanto dobbiamo modificare la pendenza w.

Derivata rispetto a b

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)})$$

Questa formula calcola quanto dobbiamo modificare l'intercetta b.

Queste formule vengono inserite nel passo di aggiornamento e ripetute iterativamente fino alla convergenza. Ricorda: $f(x) = wx + b$ è la nostra funzione di predizione.

Le Derivate Parziali Spiegate

Anatomia delle formule di aggiornamento

Scomponiamo le derivate parziali pezzo per pezzo per capire il significato di ogni termine. Questa comprensione ci aiuterà a intuire perché l'algoritmo funziona.

Struttura della Formula per w

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

1/m Σ - La Media

- 1 Consideriamo l'errore **medio** su tutto il dataset, non solo su un singolo punto. Questo rende l'algoritmo stabile e meno sensibile a outlier individuali.

(f(x) - y) - L'Errore

- 2 La differenza tra predizione e realtà ci dice quanto siamo lontani e *in che direzione* dobbiamo correggere i parametri. Errore positivo significa che stiamo sovrastimando, negativo che stiamo sottostimando.

x - L'Input

- 3 Questo termine collega l'errore al parametro w. Se x è grande, l'errore su quel punto avrà più influenza sull'aggiornamento di w. Questo ha senso: la pendenza w moltiplica x, quindi un x grande amplifica l'effetto di w.

Differenza nella Formula per b

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)})$$

- ▣ **Nota Importante:** Nella formula per **b**, manca il termine **x** finale. Perché?

Il parametro b (intercetta) è una costante che viene *aggiunta* a tutti gli esempi, indipendentemente dal valore dell'input x. Matematicamente, quando calcoliamo $\partial f / \partial b$ dove $f(x) = wx + b$, otteniamo semplicemente 1, perché la derivata di b rispetto a se stesso è 1.

Di conseguenza, l'aggiornamento di b dipende solo dalla media degli errori, senza la ponderazione per x che invece è necessaria per w.

Intuizione Geometrica

Modificare b sposta la retta verticalmente (su o giù) in modo uniforme. Modificare w ruota la retta attorno a un punto. Ecco perché w ha bisogno del termine x: l'effetto della rotazione dipende dalla posizione x in cui ci troviamo.

Funzioni Convesse vs Non Convesse

Perché la Regressione Lineare è "sicura" dal punto di vista dell'ottimizzazione

Funzione Convessa

Una funzione è detta **convessa** se ha la forma di una "ciotola" o "scodella" rivolta verso l'alto. Tecnicamente, il segmento che connette due punti qualsiasi sulla curva giace sempre sopra (o su) la curva stessa.

Un Solo Minimo

Possiede un unico minimo globale. Non ci sono "trappole" o avvallamenti locali.

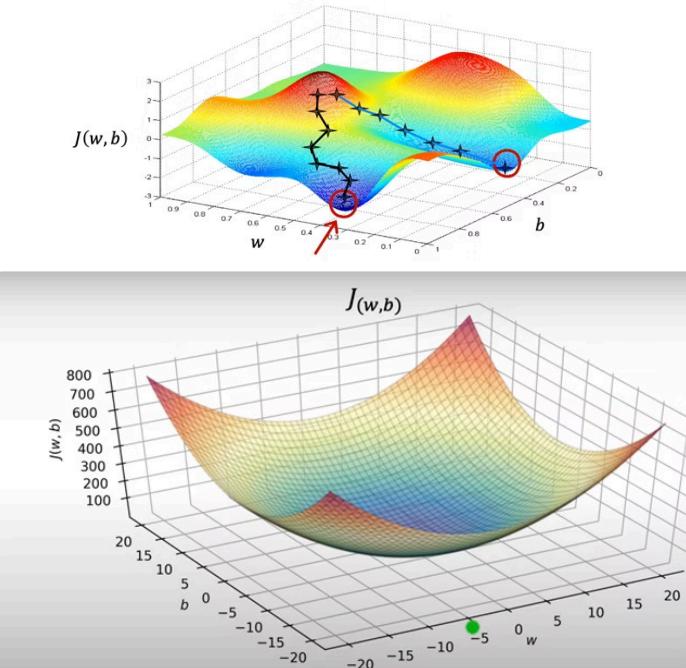
Convergenza Garantita

Non importa da dove partì (punto iniziale dei parametri), il Gradient Descent ti porterà sempre al minimo globale.

Discesa Monotona

Con un learning rate appropriato, la funzione di costo diminuisce costantemente ad ogni iterazione.

More than one local minimum



Funzione Non Convessa

Una funzione non convessa presenta una superficie complessa con picchi, valli, e molteplici minimi locali. È come un paesaggio montagnoso con molte vallate diverse.

Minimi Locali

Esistono punti "valle" che sembrano minimi ma non sono il punto più basso assoluto (minimo globale).

Dipendenza dall'Inizializzazione

Dove finisce l'algoritmo dipende da dove inizia. Due punti iniziali diversi possono portare a due minimi locali diversi.

Rischio di Blocco

Il Gradient Descent può "bloccarsi" in un minimo locale che non è la soluzione ottimale globale.

Buone Notizie per la Regressione Lineare

La funzione di costo MSE della Regressione Lineare è **sempre convessa**, indipendentemente dai dati. Questo garantisce che il Gradient Descent convergerà **sempre** al minimo globale ottimale. Non dobbiamo preoccuparci di minimi locali!

Batch Gradient Descent

Cosa significa esattamente il termine "Batch"?

Definizione e Significato

Il termine "Batch" nel contesto del Gradient Descent si riferisce a una caratteristica fondamentale dell'algoritmo che abbiamo studiato: **ad ogni singolo passo dell'algoritmo, utilizziamo l'intero set di dati di training.**

Questo può sembrare ovvio, ma è importante sottolinearlo perché esistono varianti dell'algoritmo che funzionano diversamente.

Caratteristiche del Batch Gradient Descent

1

Sommatoria Completa

La formula delle derivate include la somma su *tutti* gli m esempi: $\sum_{i=1}^m$. Nessun dato viene escluso dal calcolo.

2

Calcolo Esatto

Ad ogni iterazione, calcoliamo l'esatto gradiente della funzione di costo, considerando il contributo di ogni singolo punto del dataset.

3

Stabilità della Traiettoria

Poiché considera tutti i dati, il percorso verso il minimo è molto stabile e diretto. Non ci sono "zigzag" casuali.

"Batch" gradient descent

"Batch": Each step of gradient descent uses all the training examples.

	x size in feet ²	y price in \$1000's	$m = 47$
(1)	2104	400	
(2)	1416	232	
(3)	1534	315	
(4)	852	178	
...	
(47)	3210	870	

Vantaggi e Svantaggi

Pro: Convergenza Stabile

Percorso diretto e prevedibile verso il minimo.

Pro: Gradiente Esatto

Nessuna approssimazione nel calcolo della direzione di discesa.

Contro: Costo Computazionale

Con dataset enormi (milioni di punti), ogni iterazione può essere molto lenta.

Per dataset grandi si usano varianti come *Stochastic Gradient Descent* (un esempio alla volta) o *Mini-Batch Gradient Descent* (gruppi di esempi).

Visualizzazione Completa

Unire i puntini: dal grafico alla realtà del modello

Ora che abbiamo compreso tutti i pezzi individuali, mettiamo insieme l'intero puzzle per vedere come il Gradient Descent trasforma matematica astratta in risultati concreti.

Il Viaggio Completo dell'Ottimizzazione

Punto di Partenza

Inizializziamo i parametri (es. $w=0, b=0$). Questo ci posiziona in un punto sulla superficie 3D della funzione di costo, probabilmente lontano dal minimo.

Vista Dall'Alto - Contour Plot

Guardando il contour plot (vista 2D dall'alto), vediamo il punto muoversi attraverso le ellissi concentriche, spiralando verso il centro che rappresenta il minimo.

1

2

3

4

Il Percorso di Discesa

Ad ogni iterazione, calcoliamo le derivate e aggiorniamo i parametri. Questo sposta il punto sulla superficie tridimensionale, scendendo lungo la pendenza verso il basso.

Effetto sul Modello Reale

Nel mondo reale dei dati, ogni aggiornamento dei parametri modifica la retta di regressione $f(x) = wx + b$, facendola allineare sempre meglio ai punti dati effettivi.

Percorso di ottimizzazione (vista combinata)

La Connessione tra Astratto e Concreto

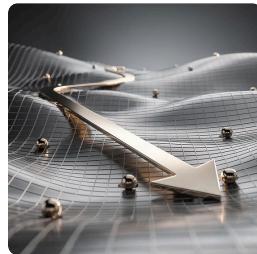
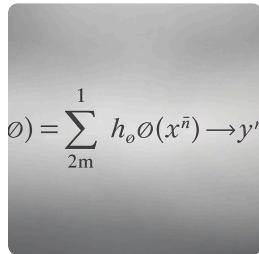
Questa visualizzazione completa rivela la bellezza del Gradient Descent: un processo matematico astratto (minimizzare $J(w,b)$ nello spazio dei parametri) produce un risultato tangibile e interpretabile (una retta che si adatta ai dati reali).

- Guardare queste tre rappresentazioni simultaneamente - la superficie 3D, il contour plot 2D, e il grafico dei dati con la retta - aiuta a costruire un'intuizione profonda di come funziona l'ottimizzazione nel Machine Learning.

Riepilogo

I 4 pilastri fondamentali di questa lezione

Abbiamo percorso un viaggio completo attraverso i concetti fondamentali dell'ottimizzazione nella regressione lineare. Ricapitoliamo i pilastri essenziali su cui si basa tutto ciò che abbiamo imparato.



01 - Funzione di Costo J

È la nostra bussola nell'oceano dell'ottimizzazione. La funzione di costo $J(w,b)$ misura l'errore quadratico medio (MSE) tra le predizioni del nostro modello e i dati reali. Rappresenta matematicamente quanto il nostro modello sia "sbagliato". L'obiettivo fondamentale di tutto il processo è minimizzarla, trovando i parametri che producono il valore più basso possibile di J .

02 - Gradient Descent

È il motore dell'ottimizzazione. Questo algoritmo iterativo usa le derivate parziali (che rappresentano la pendenza della superficie di costo) per scendere sistematicamente verso il punto di minimo globale. Ad ogni passo, calcola in quale direzione muoversi e quanto spostarsi, avvicinandoci progressivamente alla soluzione ottimale senza bisogno di esplorare casualmente lo spazio dei parametri.

03 - Learning Rate α

È l'acceleratore (o il freno) del processo. Il learning rate determina la grandezza dei passi che facciamo ad ogni iterazione. Va scelto con cura attraverso sperimentazione: un α troppo piccolo rende l'addestramento dolorosamente lento; un α troppo grande causa oscillazioni e divergenza, allontanandoci dal minimo invece di avvicinarci.

04 - Aggiornamento Simultaneo

È la regola d'oro dell'implementazione corretta. Dobbiamo calcolare i nuovi valori per **tutti** i parametri (w e b) basandoci sui valori attuali, e solo dopo averli tutti calcolati, sovrascrivere simultaneamente i vecchi valori. Aggiornare i parametri uno alla volta usando valori già modificati viola la matematica dell'algoritmo e produce comportamenti incorretti.

Applicazioni Pratiche

Dove si usa la Regressione Lineare nel mondo reale di oggi

La regressione lineare, nonostante sia uno dei modelli più semplici del Machine Learning, rimane incredibilmente utile e ampiamente utilizzata in innumerevoli settori. Vediamo alcuni esempi concreti di applicazioni che incontriamo quotidianamente.



1. Real Estate e Valutazione Immobiliare

Stimare il valore di mercato degli immobili basandosi su caratteristiche misurabili come metri quadri, numero di stanze, numero di bagni, anno di costruzione, e posizione geografica. Piattaforme come Zillow e Immobiliare.it utilizzano modelli di regressione (spesso più complessi della regressione lineare semplice, ma basati sugli stessi principi) per fornire stime automatiche di prezzo.



2. Marketing e Previsione Vendite

Prevedere il volume delle vendite future in base alla spesa pubblicitaria su diversi canali come TV, radio, social media, e campagne email. Le aziende usano questi modelli per ottimizzare l'allocazione del budget marketing, investendo di più nei canali che mostrano il ROI (Return on Investment) più alto secondo le previsioni del modello.



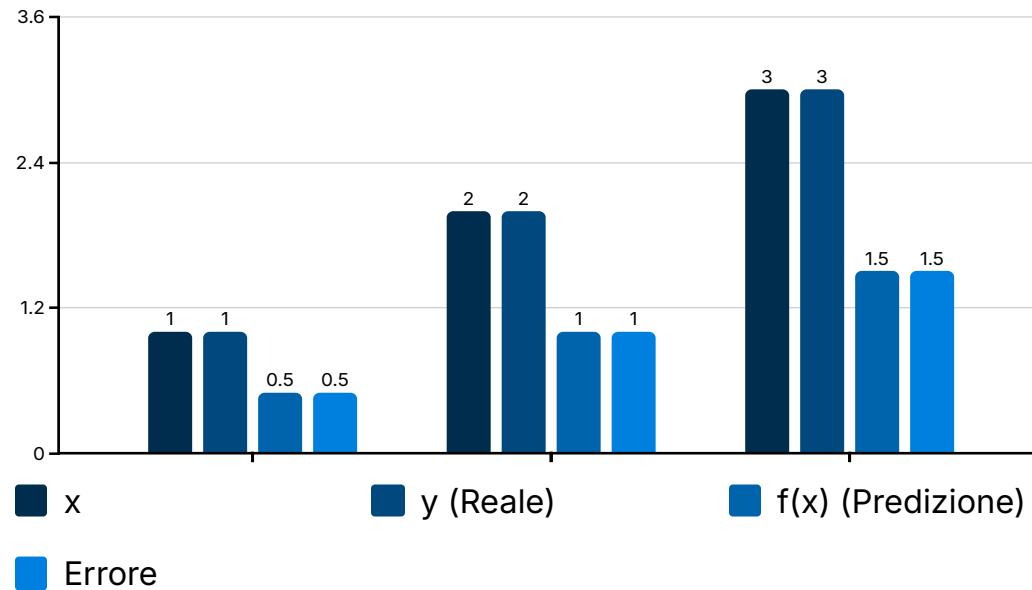
3. Produzione Industriale e Manutenzione

Stimare i costi di produzione, i tempi di manutenzione necessari, o il consumo energetico in base all'età dei macchinari, alle ore di utilizzo, alla temperatura di esercizio, e ad altri fattori operativi. Questo permette alle aziende manifatturiere di pianificare meglio la manutenzione preventiva e ottimizzare l'efficienza operativa.

- L'esempio dei prezzi delle case è il caso d'uso più classico e intuitivo per introdurre la regressione lineare, ed è esattamente quello che abbiamo usato in questa lezione per illustrare i concetti.

Esercizio Pratico 1: Calcolo della Funzione di Costo

Per mettere in pratica quanto appreso, calcoliamo manualmente il valore della funzione di costo $J(w,b)$ per un semplice set di dati e parametri specifici. Questo ci aiuterà a consolidare la comprensione di come l'errore del modello viene quantificato.



Parametri del Modello

Ipotizziamo i seguenti valori iniziali per i pesi (w) e il bias (b) del nostro modello di regressione lineare:

- w (peso) = 0.5
- b (bias) = 0

Il tuo compito: Calcola il valore della [funzione di costo \$J\(w,b\)\$](#) per questi parametri e questi dati.

Soluzione Esercizio Pratico 1

Calcoliamo passo dopo passo il valore della funzione di costo $J(w,b)$ utilizzando i dati forniti e i parametri del modello $w=0.5$, $b=0$.



Calcolo degli Errori

Per ogni punto dati, sottraiamo il valore reale (y) dalla predizione del modello $f(x_i) = wx_i + b$.

- $f(1) - y_1 = 0.5 - 1 = -0.5$
- $f(2) - y_2 = 1 - 2 = -1$
- $f(3) - y_3 = 1.5 - 3 = -1.5$



Quadrati degli Errori

Eleviamo al quadrato ciascun errore per eliminare i segni negativi e dare maggiore peso agli errori più grandi.

- $(-0.5)^2 = 0.25$
- $(-1)^2 = 1$
- $(-1.5)^2 = 2.25$



Somma degli Errori Quadratici

Sommiamo tutti i quadrati degli errori ottenuti.

- $0.25 + 1 + 2.25 = 3.5$



Calcolo della Funzione di Costo

Infine, dividiamo la somma per $2m$ (dove $m = 3$ è il numero di esempi).

- $J(0.5, 0) = (1/(2 * 3)) * 3.5 = 3.5/6 \approx 0.5833$

Il valore della funzione di costo per i parametri dati è circa **0.58**. Questo valore quantifica l'errore medio del modello rispetto ai dati reali con i parametri attuali.

Esercizio Pratico

Mettiamoci alla prova con i concetti appresi

Ora è il momento di applicare ciò che abbiamo imparato. Questi esercizi ti aiuteranno a consolidare la comprensione teorica attraverso l'applicazione pratica.

Esercizio 2: Debugging

Stai allenando il tuo modello di regressione lineare su un dataset di 10.000 esempi. Dopo 100 iterazioni del Gradient Descent, noti che il valore della funzione di costo J **sta aumentando** invece di diminuire. Qual è la causa più probabile e come risolvi il problema?

Diagnosi

Causa Probabile: Il Learning Rate (α) è troppo grande. L'algoritmo sta facendo "overshoot" - sta saltando oltre il minimo a ogni iterazione, finendo in punti sempre più lontani dall'ottimo e divergendo.

Soluzione

Azione Correttiva: Riduci significativamente α . Prova a dividerlo per 3 o per 10 (es. se $\alpha=0.1$, prova $\alpha=0.03$ o $\alpha=0.01$). Riavvia l'addestramento con il nuovo valore e monitora se J diminuisce costantemente.

Prevenzione

Best Practice: Testa sempre diversi valori di α prima dell'addestramento completo. Usa poche iterazioni su ciascun valore e scegli quello che produce la discesa più rapida senza instabilità.

Domande Frequenti

Dubbi comuni e risposte chiare per consolidare la comprensione

Durante lo studio del Gradient Descent e della regressione lineare, alcuni dubbi emergono frequentemente. Affrontiamoli con chiarezza.

Q1: Il Gradient Descent può bloccarsi in un minimo locale?

Per la Regressione Lineare con funzione di costo MSE, la risposta è **NO**. La funzione di costo è matematicamente convessa (forma a ciotola), garantendo l'esistenza di un unico minimo globale. Il Gradient Descent convergerà sempre a questo punto, indipendentemente dall'inizializzazione.

Tuttavia, per modelli più complessi come le Reti Neurali con funzioni di costo non convesse, **SÌ**, è un rischio reale. In quei casi si usano tecniche avanzate come inizializzazioni intelligenti, momenti, e learning rate adattivi per mitigare il problema.

Q2: Come scelgo i valori iniziali di w e b ?

Per la regressione lineare, la scelta dei valori iniziali è molto flessibile. Puoi tranquillamente iniziare da **$w=0$ e $b=0$** , o da piccoli numeri casuali. Poiché la funzione di costo ha un unico minimo globale, l'algoritmo convergerà sempre allo stesso punto ottimale, indipendentemente da dove partì.

In modelli più complessi (es. reti neurali), l'inizializzazione diventa cruciale e si usano tecniche sofisticate come Xavier o He initialization.

Q3: Cosa succede se le variabili hanno scale molto diverse?

Se le feature hanno scale molto diverse (es. una varia tra 0-1 e un'altra tra 0-10000), il Gradient Descent può rallentare significativamente. La superficie della funzione di costo diventa allungata come un'ellisse stretta invece di una ciotola circolare, rendendo la convergenza inefficiente.

Soluzione: Applica il **Feature Scaling** (normalizzazione o standardizzazione) alle tue variabili prima di iniziare l'addestramento. Questo trasforma tutte le feature in scale comparabili, accelerando notevolmente la convergenza.