

# Regressione Logistica

lunedì 15 aprile 2024 11:13

È un problema di classificazione.

l'output è composto da un numero finito di elementi.

Classificazione Binaria e Multiclasse.

## CLASSIFICAZIONE BINARIA

Funzione Sigmoidale, restituisce un valore compreso tra 0 e 1 per ogni valore reale passato in input.

Vedi le slides dove ci sono gli esempi.

## FUNZIONE DI COSTO PER LA CLASSIFICAZIONE BINARIA

Se utilizzassimo la funzione di costo usata per la regressione lineare otterremo una funzione non convessa, cioè che non ha un minimo assoluto ma ha molti massimi e minimi relativi.

Dobbiamo usare una funzione di costo "comoda" per la classificazione binaria, basandoci sui seguenti vincoli:

$$\text{Cost}(h_{\theta}(x), y) = 0 \text{ if } h_{\theta}(x) = y$$

$$\text{Cost}(h_{\theta}(x), y) \rightarrow \infty \text{ if } y = 0 \text{ and } h_{\theta}(x) \rightarrow 1$$

$$\text{Cost}(h_{\theta}(x), y) \rightarrow \infty \text{ if } y = 1 \text{ and } h_{\theta}(x) \rightarrow 0$$

Il costo è nullo solo quando la funzione ipotesi ed il valore reale hanno lo stesso valore.

Il costo tende a infinito se il valore reale è nullo e la funzione ipotesi tende a 1.

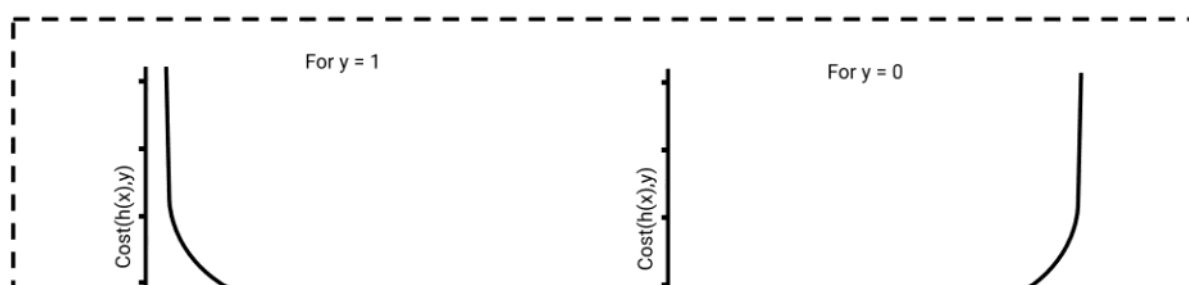
Il costo tende a infinito se il valore reale è uguale a 1 e la funzione ipotesi tende a 0.

La funzione costo che rispetta tali vincoli è :

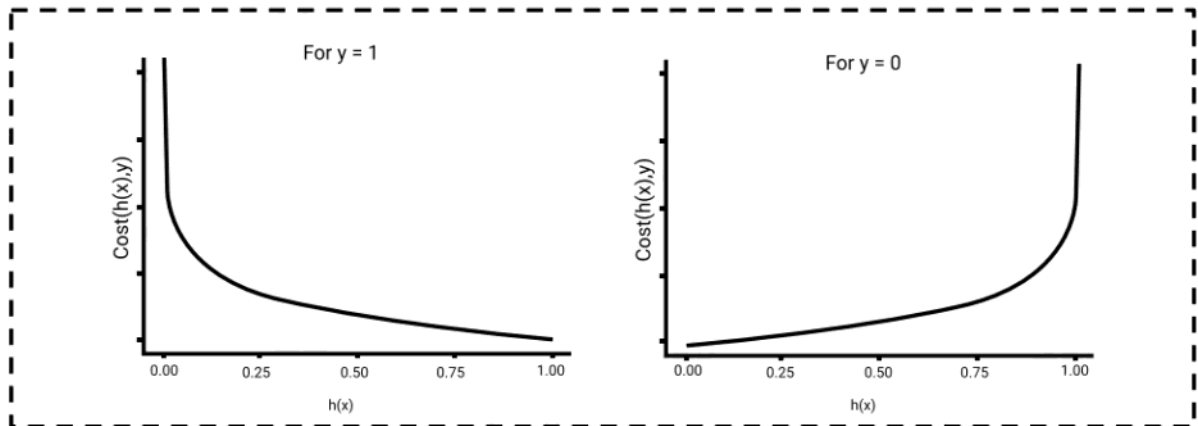
$$\text{Cost}(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x))$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(h_{\theta}(x)) \quad \text{if } y = 1$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$



$$\begin{aligned} \text{Cost}(h_{\theta}(x), y) &= -\log(h_{\theta}(x)) && \text{if } y = 1 \\ \text{Cost}(h_{\theta}(x), y) &= -\log(1 - h_{\theta}(x)) && \text{if } y = 0 \end{aligned}$$



# Overfitting e Regularizzazione

lunedì 15 aprile 2024 14:09

## OVERFITTING

Il modello non generalizza bene le predizioni relative ad input mai visti in precedenza.

Errore nel test-set molto maggiore dell'errore nel training-set.

Cause e soluzioni:

Il modello è troppo complesso per il problema analizzato, il modello si adatta troppo bene sui dati di addestramento.

Ci sono troppe Features per addestrare il modello.

## REGOLARIZZAZIONE

Per risolvere il problema dell'overfitting si fa la regolarizzazione.  
Mantiene tutte le Features ma ne riduce il relativi parametri Theta j.


Se impostiamo un limite massimo oltre al quale i nostri parametri theta non possono andare non permetteremo al modello di seguire appieno i dati che vengono dati in input al modello stesso e quindi il modello riesce a generalizzare molto meglio rispetto a llo stesso modello, con la stessa architettura ma senza alcun limite per i parametri theta.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Funzione di costo

$\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$

Inizia da 1 e non da 0



La funzione costo del modello darà un errore tanto più alto quanto più i parametri theta saranno grandi.

Tutto ciò durante la discesa del gradiente impedirà che i parametri possano crescere molto, quindi non seguire troppo i dati, per far sì che l'errore sia minimo.

In questo modo, durante l'addestramento si settano i parametri theta, per far sì che il modello segua i dati che gli si danno in input, ma senza farli crescere troppo per evitare un errore della funzione costo troppo elevato.

-Utilizzando il giusto parametro Lambda possiamo consentire al nostro modello di apprendere dai dati senza che il modello li impari a memoria (overfitting)

# Intro ANN

lunedì 15 aprile 2024 17:12

Può capitare di dover lavorare con molti parametri ed i modelli lineari non sono proprio i più adatti per farlo.

Le ANN offrono una strada alternativa nei casi di ipotesi complesse e di un gran numero di features.

Le ANN riescono ad operare anche su dataset con un alto numero di features senza dover per forza sviluppare un numero enorme di parametri da addestrare.

## Funzioni di attivazione

lunedì 15 aprile 2024 17:55

Utilizzate per inserire **non linearità** al nostro modello per poter rappresentare fenomeni complessi che non sono rappresentabili tramite semplici relazioni lineari.

Tali funzioni di attivazione riescono a propagare più o meno il segnale in base a come sono state impostate.

Hanno una funzione simile a ciò che fanno i neuroni biologici:

I neuroni biologici hanno al loro interno una membrana che serve per filtrare il passaggio del segnale elettrico da un neurone e verso l'altro.

Nel momento in cui un segnale elettrico abbastanza forte entra in un neurone, tale segnale riesce a superare la membrana interna e giungere al neurone adiacente.

Le funzioni di attivazione nei neuroni artificiali hanno lo stesso identico scopo della membrana, cioè permettono o meno il passaggio delle informazioni ai neuroni successivi.

Le funzioni non lineari sono fondamentali per rappresentare fenomeni complessi come il raggruppamento dei dati in cerchi, cosa che con le funzioni lineari non riusciremmo a rappresentare.

### Funzione di Attivazione Sigmoide

La derivata della funzione sigmoide ha valori molto piccoli per valori di  $x$  troppo grandi o troppo piccoli. Quindi poiché la derivata della funzione sigmoide tende a zero per questi valori gli aggiornamenti dei parametri sarebbero trascurabili, cioè il gradiente SVANISCE.

### Funzione di Attivazione Tangente iperbolica:

Passaggio graduale tra -1 ed 1.

Possiamo correggere anche i parametri che danno un output negativo nella funzione TI.

La sua derivata inoltre raggiunge valori più elevati della derivata della funzione sigmoide, quindi riusciamo ad addestrare il modello più velocemente.

Logicamente riusciamo a sfruttare ciò quando gli input sono tra -1 e 1 cioè dove danno valori più elevati per la derivata.

Anche qui abbiamo il problema della SVANIZIONE del gradiente per valori troppo grandi e troppo piccoli degli input.

### SCOMPARSA DEL GRADIENTE

È una difficoltà riscontrata nell'apprendimento durante la discesa del gradiente.

Ogni peso della rete viene aggiornato sfruttando la derivata parziale della funzione di costo rispetto al peso stesso. Se il gradiente risulta in un valore prossimo allo zero, il relativo peso non viene aggiornato.

Problema opposto: ESPLOSIONE DEL GRADIENTE.

### ReLU

Riduce la scomparsa del gradiente poiché riesce ad aggiornare i parametri per ogni  $x \geq 0$ , poiché la sua derivata per tali valori vale sempre 1.

### Leaky ReLU

$y = 0.01 \cdot x$  per  $x < 0$  creando una linea leggermente inclinata (con derivata piccola ma diversa da 0).

Permette l'addestramento del modello anche per input  $< 0$

**L'efficacia delle funzioni di attivazione deriva dal fatto che riescono a fornire un comportamento non lineare ai modelli in cui vengono proposte. Ovvero riescono a trovare delle relazioni tra gli input e gli output che normalmente non potrebbero essere riconosciute poiché sono relazioni altamente non lineari.**

**Aggiungendo layer al modello non facciamo altro che amplificare la non linearità del modello. La non linearità espressa in output dalle funzioni di attivazioni del primo layer verrà intensificata ancor di più alle funzioni di attivazione del secondo layer e così per tutti e quanti i layer della nostra rete.**

# Reti neurali artificiali

lunedì 22 aprile 2024

13:05

TF Playground ci consente di vedere graficamente ciò che accade all'interno delle reti neurali.

Il Batch Size: il numero di esempi che vengono sottoposti alla rete. Ad esempio con un Batch size di 30 sottoporremo alla rete 30 esempi del dataset per volta.

Una volta sottoposto un gruppo di 30 esempi alla rete vengono raccolti gli errori commessi dalla rete su ognuno di questi esempi e la media di questi errori servirà per aggiornare il modello.

Ovvero i parametri del modello non vengono aggiornati su ogni singolo esempio ma su una media calcolata su un certo numero di esempi .

## BACKPROPAGATION

È l'algoritmo che ci consente di addestrare le reti neurali artificiali.

Funzionamento:

Dato un input alla rete, questo input viene fatto propagare in avanti in layer in layer fino a raggiungere l'output. Una volta raggiunto l'output possiamo misurare di quanto ha sbagliato misurare, semplicemente confrontando l'output calcolato dalla rete con il label del nostro dataset. Una volta calcolato l'errore possiamo sfruttarlo tramite la discesa discesa del gradiente per poter correggere i parametri relativi all'ultimo layer.

Una volta corretti questi parametri sfruttiamo l'errore sull'ultimo layer per correggere i parametri che entrano nel penultimo layer e così fino ad arrivare a correggere i parametri del primo layer.

In questo modo riusciamo a minimizzare gli errori ed a minimizzare la funzione di costo

È disponibile direttamente all'interno di TF.

È un'interfaccia di alto livello su altri framework, ovvero consente di operare con un linguaggio comune su altri framework come tensorflow, toolkit di microsoft e theano.

Keras ci consente di prototipare ed addestrare efficacemente i nostri modelli, poiché si occupa lui automaticamente del calcolo parallelo su cpu e gpu.

Quindi piuttosto che reinventare la ruota e riscrivere tutti i passaggi da 0 necessari per implementare un'algoritmo di discesa del gradiente e creare una struttura dati capace di ospitare tutti i parametri necessari alla creazione di una rete, Keras ci consente di velocizzare tutte le operazioni relative alla definizione del modello, addestramento del modello e alla validazione del modello.

Vogliamo creare una rete che riconosca i numeri scritti a mano.  
Usiamo il dataset MNIST disponibile in Keras.

Le epoche (epochs=10) sta a significare quante volte il modello si deve addestrare su TUTTI i dati presenti nel dataset.

Il batch\_size (ad es batch\_size=250) invece sta a significare il numero di esempi del dataset che dobbiamo passare al modello per poi calcolare su di essi gli output, dopodiché si mediano gli errori degli output per calcolare un errore medio usato poi per correggere i parametri del modello. E così via batch dopo batch fino ad esaurire il dataset.

## **Dataset Titanic**

Raccogli i dati relativi ai passeggeri del Titanic e per ognuno di questi se è sopravvissuto o no.

Il nostro obiettivo è quello di creare un classificatore con una rete neurale che in base ai dati degli esempi ci dica se un passeggero è sopravvissuto o no, e così anche per nuovi esempi di passeggeri.

## **Dataset IMDB**

Recensioni su film, utile per confrontare vari algoritmi di NLP.

È già presente in Keras quindi no necessità formattazione come abbiamo fatto per il Titanic.

Con il parametro num\_words=10000 stiamo indicando che vogliamo un vocabolario di 10000 parole, le parole meno frequenti del dataset IMDB non saranno presenti in questo dizionario poiché si prendono quelle più frequenti.

Si vedrà che data è una lista di numeri ciascuno dei quali corrisponde a una parola del vocabolario.