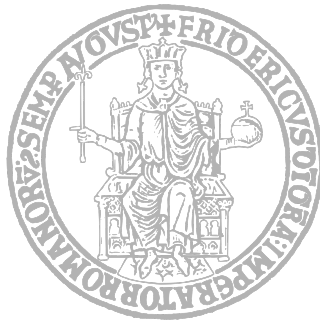


Elaborato d'esame

Calcolo Numerico A.A 2021/2022
Prof.ssa Luisa D'Amore



DIETI
Università Federico II di Napoli

Machine Learning:
Regressione Lineare e Logistica
e loro applicazioni

Autori:

Antonio Romano M63001315
Giuseppe Riccio M63001314
Raffaele Russo M63001325

Indice

1	Introduzione	3
2	Differenze tra apprendimento supervisionato e non supervisionato	4
2.1	Cenni sull'apprendimento supervisionato	4
2.2	Cenni sull'apprendimento non supervisionato	4
3	Regressione lineare	5
3.1	Discesa del gradiente	7
3.1.1	Batch Gradient Descent	8
3.1.2	Stochastic Gradient Descent	9
3.2	Normal Equations	12
3.2.1	Proprietà della derivazione matriciale	12
3.3	Metodo dei minimi quadrati rivisitato	15
4	Confronto tra discesa del gradiente e normal equations applicato ad un dataset per prevedere il costo dell'assicurazione medica	17
4.1	Complessità computazionale	18
4.1.1	Normal Equations	18
4.1.2	Discesa del gradiente	18
4.1.3	Confronto su MATLAB	19
4.2	Metriche di valutazione per la Regressione Lineare	20
4.2.1	Mean Squared Error	20
4.3	Considerazione finale	20
5	Interpretazione probabilistica	21
6	Classificazione e Regressione Logistica	23
6.1	Metriche di valutazione per la Regressione Logistica	25
6.1.1	Matrice di confusione	25
7	Applicazione MATLAB della Regressione Logistica applicato al set di dati per la predizione del SARS-CoV2 (Covid19)	27
7.1	Machine Learning steps	27
7.1.1	Raccolta e Preparazione dei dati	27
7.1.2	Scelta del modello e fitting dei parametri	28
7.1.3	Valutazione delle prestazioni del modello	29
7.1.4	La Prediction	31
8	Overfitting e Underfitting	33
8.1	Rilevazione dell'overfitting e dell'underfitting	34
9	Proof of Concept: da MATLAB App a Web App for Browser	36
10	Sviluppi futuri	38
11	Bibliografia e Sitografia	39

1 Introduzione

Un algoritmo di **Machine Learning** è un algoritmo in grado di apprendere dai dati. Per capire il significato di apprendere si fa riferimento alla definizione di Tom Mitchell:

”Un programma apprende dall’esperienza E rispetto a un task T e a una misura delle sue prestazioni P se le prestazioni del programma sul task T , misurate da P , migliorano con l’esperienza E ”

Il Machine Learning ci consente di risolvere problemi che sarebbero troppo complessi da trattare con le tecniche tradizionali di programmazione. Si supponga a tal proposito di voler realizzare un algoritmo di Machine Learning in grado di individuare email spam a partire da alcuni esempi di email spam e regolari, che costituiscono il nostro training set. In questo caso il task T è l’individuazione di email spam, l’esperienza E è data dal training set e una possibile misura per le prestazioni P è l’accuracy, ossia la frazione di email correttamente individuate dall’algoritmo. Realizzare lo stesso algoritmo con le tecniche di programmazione tradizionale richiederebbe la ricerca di pattern ricorrenti nelle email spam. Il tutto si riduce alla scrittura di regole lunghe e complesse, ma soprattutto difficili da mantenere. Un programma basato sul Machine Learning invece è in grado, attraverso un algoritmo di apprendimento, di imparare automaticamente quali sono le caratteristiche di una email tali da renderla spam. Riusciamo così a ottenere risultati più affidabili e un programma più breve e facile da mantenere. Eventuali cambiamenti dei pattern nelle email spam saranno appresi automaticamente dal modello di Machine Learning, senza intervento umano, mentre in un programma tradizionale sarebbe necessario individuare manualmente tali pattern per poi aggiornare le regole.

I modelli di Machine Learning hanno applicazione in numerosi ambiti: nella manutenzione predittiva permettono di reagire in maniera tempestiva alla rilevazione di eventuali anomalie nei dati o guasti. Possono anche essere utilizzati per prevedere eventi futuri come l’andamento del prezzo di un’azione, previsioni meteorologiche e persino i risultati delle elezioni presidenziali americane. Come nella vita, anche in queste situazioni prevale la logica che:

”Prevedere è meglio che curare”

Evitiamo così di dover sostenere maggiori costi in caso di complicazioni non pianificate.

Si riportano i casi di applicazione che saranno presi in esame:

- Previsione del costo di un immobile.
- Previsione del costo assicurazione.
- Previsione positività al SARS-CoV2 (COVID19) di un paziente.

In tutti e tre i casi, a partire da un insieme di dati iniziali x e di uscite corrispondenti y , viene addestrato un modello con un opportuno algoritmo di ottimizzazione per apprendere la relazione che sussiste tra l’ingresso e l’uscita. In tal modo sarà possibile, a partire da nuovi dati di ingresso \hat{x} , stimare l’uscita corrispondente \hat{y} .

In questo lavoro saranno approfonditi i modelli di **Regressione Lineare** e di **Regressione Logistica**, evidenziandone le differenze e la base teorica.

2 Differenze tra apprendimento supervisionato e non supervisionato

Nell'ambito del **Machine Learning** occorre fare una distinzione tra i modelli di apprendimento supervisionato e non supervisionato. La distinzione fondamentale tra i due modelli è basata sul concetto di etichettamento dei dati.

2.1 Cenni sull'apprendimento supervisionato

In un modello di **apprendimento supervisionato** forniamo al modello sia l'ingresso x che l'output desiderato, ossia l'etichetta y . Si consideri per esempio il caso in cui si voglia addestrare un modello a riconoscere diversi tipi di fiori. A tal proposito forniremo al modello un insieme di dati relativi alle caratteristiche del fiore e l'output corretto corrispondente, ossia il tipo di fiore in esame.

Successivamente il modello potrà predire l'output relativo a dati in ingresso non etichettati.

2.2 Cenni sull'apprendimento non supervisionato

In un modello di **apprendimento non supervisionato** i dati di training forniti al modello non sono etichettati. Tornando all'esempio della classificazione dei fiori, nell'addestramento di un modello non supervisionato, forniamo in ingresso soltanto le caratteristiche sui fiori, ma non il tipo di fiore corrispondente. Con un apposito algoritmo il modello potrà cercare di raggruppare i fiori in gruppi con caratteristiche simili. Questo tipo di apprendimento non supervisionato è comunemente chiamato **clustering**.

Nella parte restante dell'elaborato si farà riferimento a tecniche di apprendimento supervisionato. Per esempio, nel caso della predizione di positività al SARS-CoV2 (Covid19) forniremo per addestrare il modello sia i sintomi del paziente considerato, sia l'output corrispondente, ossia la sua positività o negatività.

3 Regressione lineare

Come riportato nel capitolo precedente, l'apprendimento supervisionato può essere usato per prevedere i prezzi degli immobili in una determinata area. A titolo d'esempio si supponga di disporre di un dataset contenente area e prezzo di m case nell'area di Napoli. A partire da questi dati ci si chiede come sia possibile predire il prezzo di altre case in funzione della loro area. Tale problema prende il nome di **Regressione Lineare Semplice**.

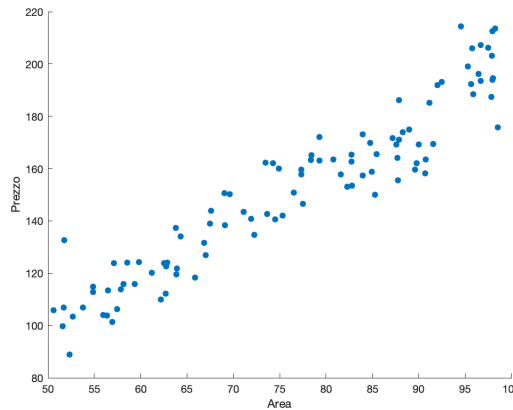


Figura 1: Dataset

Il codice MATLAB usato per generare il dataset nella Figura 1 è il seguente:

```
1 %Generazione dataset
2 %x rappresenta la superficie di una casa
3 %y rappresenta il prezzo della casa di area x
4
5 low = 50;
6 high = 100;
7 x = (high-low).*rand(200,1) + low;
8 y = 2 * x + 10 .* randn(200,1);
9
10 %Normalizzo i dati
11 x = normalize(x, 'range', [0 1]);
12 y = normalize(y, 'range', [0 1]);
```

Codice MATLAB 1: Generazione dataset casuale delle case nell'area di Napoli

Si definisce l'area di ogni casa del dataset $x^{(i)}$ come **variabile di ingresso del modello da costruire o feature**. Il prezzo di ogni casa rappresenta, invece, l'**output del modello** $y^{(i)}$, **anche detto variabile target**. La coppia $(x^{(i)}, y^{(i)})$ costituisce un **esempio di training** e l'insieme di tutte le coppie per $i = 1 \dots m$ costituiscono il **training set**.

Definito come X l'insieme in cui variano i valori degli input, Y l'insieme in cui variano i valori degli output, un problema di apprendimento supervisionato consiste, a partire da un training set, nel trovare una funzione $h : X \rightarrow Y$, detta **funzione ipotesi**, che fornisce una *buona stima* di $y^{(i)}$ a partire da $x^{(i)}$. Nell'esempio precedente $X = Y = \mathbb{R}$.

Se la variabile y che si sta *predicendo* è continua si parla di un problema di **regressione**, mentre nel caso in cui assuma un insieme discreto di valori si parla di problema di **classificazione**, affrontato in seguito.

Se l'insieme degli input varia nell'insieme \mathbb{R}^n nel caso **lineare** la funzione ipotesi assume la seguente forma:

$$h_{\theta}(x) = \theta_0 + \sum_{i=1}^n \theta_i x_i \quad (1)$$

Fissato $x_0 = 1$ e per semplicità di notazione, ignorando la dipendenza di h da θ , possiamo scrivere:

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x \quad (2)$$

A questo punto ci si chiede come sia possibile effettuare il **tuning dei parametri** θ del nostro modello, ovvero ci si chiede quali siano i valori di θ che minimizzano la distanza tra i valori predetti e quelli attesi. Una possibilità consiste nel rendere la funzione $h(x)$ vicina a y , almeno per gli esempi di training a nostra disposizione. Si definisce allora una funzione che misura quanto $h(x^{(i)})$ è vicino al corrispondente $y^{(i)}$ in funzione del parametro θ :

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 \quad (3)$$

Tale funzione costo è alla base del metodo dei **minimi quadrati**, la cui minimizzazione sarà affrontata nei paragrafi successivi.

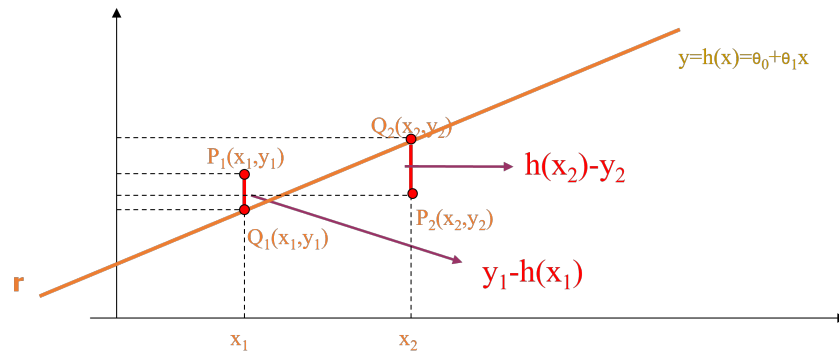


Figura 2: Retta dei minimi quadrati

L'idea alla base del metodo è intuibile dalla figura 2, in cui è possibile valutare lo scostamento tra la retta ed i punti. Possiamo osservare che la distanza (verticale) di $P_1(x_1, y_1)$ sarà $P_1Q_1 = d_1 = y_1 - h(x_1)$. Stessa cosa per la distanza (verticale) di $P_2(x_2, y_2)$ cioè $P_2Q_2 = d_2 = y_2 - h(x_2)$. Per evitare di utilizzare il valore assoluto (e quindi il segno) si considera il **quadrato di ciascuna distanza** d_i . Dunque non resta che valutare θ_0 e θ_1 in modo che sia **minima la somma delle distanze di tutti i punti P_i della retta**. Ovvero, ci si trova a dover valutare la seguente espressione nel caso di m punti:

$$\min(d_1^2 + d_2^2 + \dots + d_m^2) \quad (4)$$

Da questo concetto discende l'equazione 3. Per minimizzare tale sommatoria si applicheranno i metodi della **discesa del gradiente** e delle **normal equations**.

3.1 Discesa del gradiente

La discesa del gradiente è un algoritmo di ottimizzazione in grado di risolvere all'ottimo una vasta gamma di problemi. L'idea è variare i parametri θ iterativamente in maniera tale da minimizzare la funzione costo $J(\theta)$, come rappresentato in figura 3.

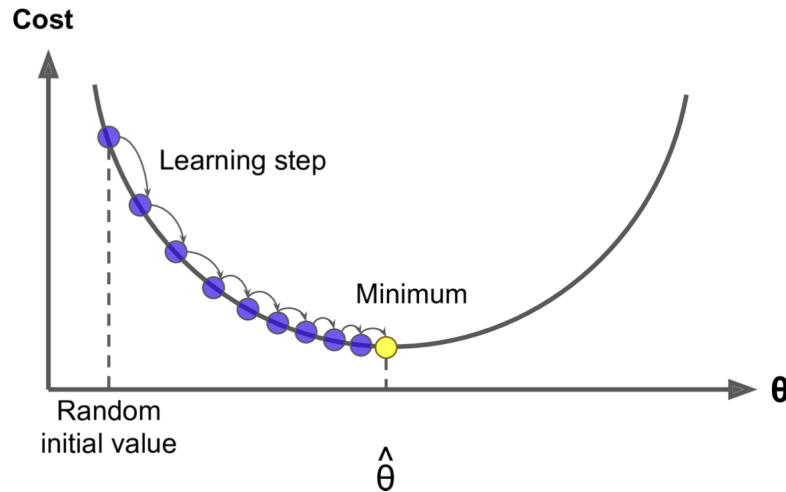


Figura 3: Discesa del gradiente

I parametri sono inizializzati casualmente e modificati simultaneamente $\forall j$ a ogni iterazione con la seguente regola fino a eventuale convergenza:

$$\theta_j = \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta) \quad (5)$$

$$\theta_j = \theta_j - \eta \frac{\partial}{\partial \theta_j} \left(\frac{1}{2} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 \right) \quad (6)$$

dove il parametro η viene detto **learning rate** e il cui valore deve essere **stabilito sperimentalmente**. Un valore troppo piccolo di questo parametro comporta una lenta convergenza, mentre un valore troppo grande potrebbe portare a un comportamento divergente della funzione costo come in figura 4.

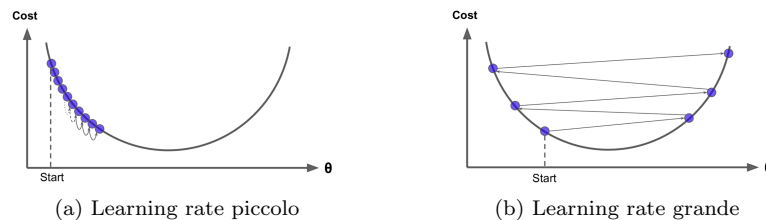


Figura 4: Convergenza discesa del gradiente al variare del learning rate

Non tutte le funzioni costo hanno un andamento così regolare e un solo punto di minimo assoluto. Potremmo lavorare con funzioni come in figura 5 in cui l'algoritmo potrà o meno convergere al punto di minimo assoluto a seconda di come sarà inizializzato il parametro θ .

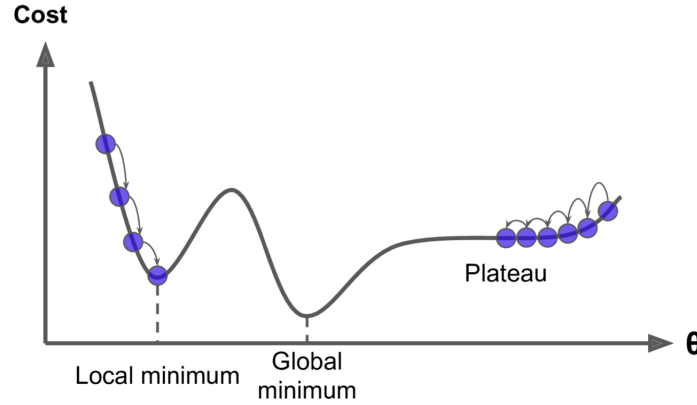


Figura 5: Convergenza discesa del gradiente con funzione irregolare

3.1.1 Batch Gradient Descent

La funzione costo definita nella 3 è una funzione quadratica convessa (per ogni coppia di punti sulla curva, il segmento che li congiunge non attraversa mai la curva), per la quale la convergenza al minimo assoluto è assicurata a patto che il **learning rate** sia scelto sufficientemente piccolo e si attenda il tempo necessario.

Per implementare questo algoritmo bisogna calcolare la derivata al secondo membro della 5. A tal proposito si suppone per semplicità di lavorare con un solo esempio di training (x, y) :

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_\theta(x) - y) x_j\end{aligned}\quad (7)$$

Allora per un singolo esempio di training si ottiene la seguente regola di aggiornamento **LMS** ("least mean squares"):

$$\theta_j = \theta_j - \eta (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (8)$$

Si osservi come l'ampiezza dell'aggiornamento è proporzionale al termine errore $(h_\theta(x^{(i)}) - y^{(i)})$. Infatti, a fronte di un esempio di training per il quale il valore predetto si avvicina al valore target $y^{(i)}$ non c'è bisogno di variare significativamente i parametri. Laddove l'errore è elevato bisognerà effettuare un cambiamento maggiore sui parametri.

Nel caso in cui si lavori con un training set di m elementi, la regola di aggiornamento per la linearità della derivata diventa:

$$\theta_j = \theta_j - \eta \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \forall j \quad (9)$$

Questo algoritmo, da eseguire fino a convergenza o per un massimo numero di iterazioni, viene detto **discesa del gradiente batch** poiché a ogni step **viene preso in considerazione l'intero training set**.

Lavorando con il dataset in figura 1, occorre dividere i dati in training set e testing set. In questo caso si è scelto di usare l'80% dei dati per il training ed il restante 20% per il testing. Tale divisione viene fatta per poter testare, a seguito dell'addestramento sul training set, le prestazioni del modello sul test set. Infine, si sceglie sperimentalmente un valore di η pari a 0.01.

Il codice MATLAB che implementa quanto descritto e l'algoritmo discesa del gradiente è il seguente:

```

1  %-----file:regressione_lineare.m-----%
2  %Divido dataset in training set e test set
3  n = size(x,1);
4  th = floor(0.8 * n);
5  x_train = x(1:th,:);
6  y_train = y(1:th,:);
7  x_test = x(th+1:n,:);
8  y_test = y(th+1:n,:);
9
10 %Inizializzo parametri theta casualmente
11 theta_batch = rand(1,length(x_train(1,:)));
12
13 %Fitting parametri con sgd e batch gradient descent
14 lr = 0.01;
15 it = 50;
16 [theta_batch, cost_hist_batch, n] = ...
    batch_gradient_descent(theta_batch,x_train,y_train,lr,it);
17
18 %-----file:batch_gradient_descent.m-----%
19 function [theta,cost_hist,n] = batch_gradient_descent(theta,x,y,lr,it)
20 cost_hist = zeros(it,1);
21
22 for n = 1 : it
23     for j = 1 : length(theta)
24         theta(j) = theta(j) - lr * ( h(x,theta) - y )' * x(:,j);
25         cost_hist(n) = cost_func(theta,x,y);
26     end
27 end
28 end

```

Codice MATLAB 2: Algoritmo di Batch Descent Gradient

Il modello risultante dall'esecuzione dell'algoritmo precedente è rappresentato in figura 6. In figura 7 viene rappresentato l'andamento della funzione costo nelle varie iterazioni. Si osservi come il valore minimo viene raggiunto dopo appena $n = 10$ iterazioni.

3.1.2 Stochastic Gradient Descent

Esiste un'alternativa alla discesa del gradiente batch, detta **discesa stocastica del gradiente (sgd)**, che prevede la seguente regola di aggiornamento:

$$\forall i = 1 \dots m \quad \theta_j = \theta_j - \eta (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \forall j \quad (10)$$

In questa versione **per ogni esempio di training del dataset** vengono aggiornati i parametri secondo il gradiente dell'errore **rispetto al singolo esempio** di training esaminato. Differentemente dall'algoritmo batch che deve valutare tutti gli m esempi del training set prima di effettuare un aggiornamento dei parametri, sgd effettua progressi velocemente a ogni esempio di training. Di conseguenza sgd spesso si avvicina al valore minimo di θ più velocemente, tuttavia potrebbe non convergere mai al minimo, ma oscillare attorno a esso.

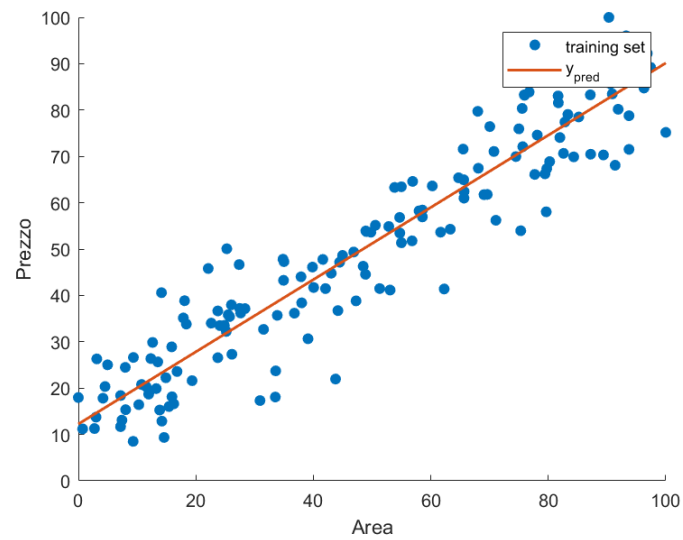


Figura 6: Esempio discesa del gradiente

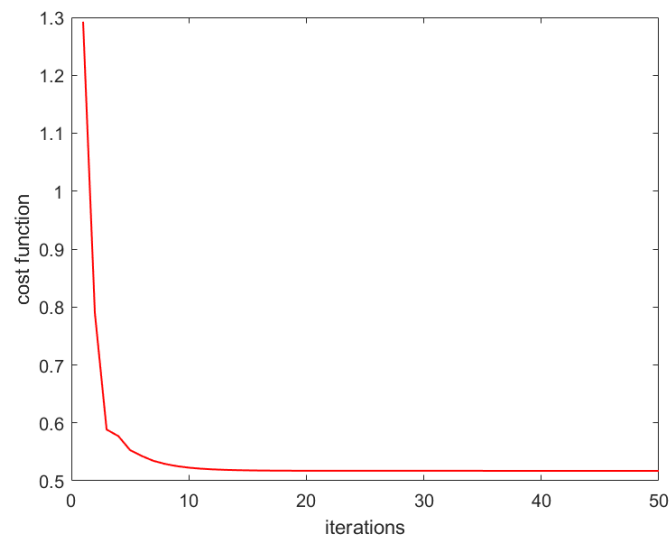


Figura 7: Andamento cost function

Il codice MATLAB che implementa la discesa stocastica del gradiente è il seguente:

```
1 %-----file:regressione_lineare.m-----%
2 %Inizializzo parametri theta casualmente
3 theta_sgd = theta_batch;
4
5 %Fitting parametri con sgd e batch gradient descent
6 [theta_sgd, cost_hist_sgd, n_sgd] = sgd(theta_sgd,x_train,y_train,lr,it);
7
8 %-----file:sgd.m-----%
9 function [theta,cost_hist,n] = sgd(theta,x,y,lr,it)
10 cost_hist = zeros(it,1);
11
12 for n = 1 : it
13     for i = 1 : length(x)
14         for j = 1 : length(theta)
15             theta(j) = theta(j) - lr * ( h(x(i,:),theta) -y(i) ) * x(i,j);
16         end
17     end
18     cost_hist(n) = cost_func(theta,x,y);
19 end
20 end
```

Codice MATLAB 3: Algoritmo di Stochastic Gradient Descent

Poiché spesso la soluzione fornita da sgd è vicina all'ottimo, esso viene preferito alla versione batch quando la dimensione del dataset è elevata.

In figura 8 è rappresentato l'andamento della funzione costo sul dataset precedente utilizzando le due versioni di discesa del gradiente. Si può osservare come il valore della funzione costo scenda più lentamente a ogni iterazione nel caso di sgd, che converge in un numero di iterazioni pari a 25, diversamente dalla versione batch che ne impiega 10.

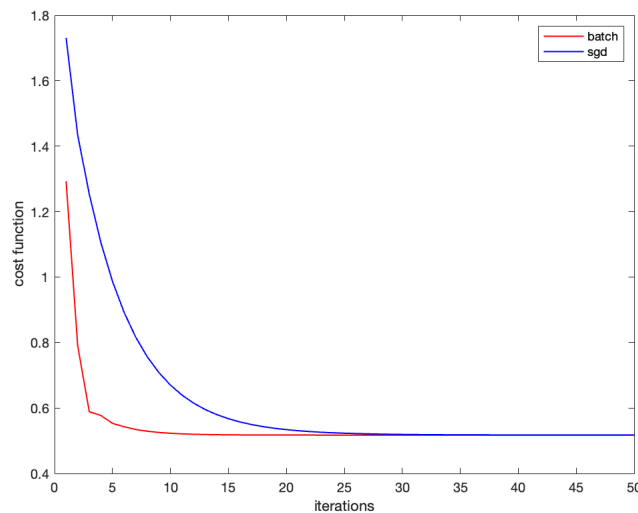


Figura 8: sgd vs batch nella regressione lineare semplice

3.2 Normal Equations

Il metodo della discesa del gradiente non è l'unico metodo per minimizzare la funzione costo $J(\theta)$. Infatti, è possibile ottenere la minimizzazione di tale funzione con metodi che non ricorrono ad algoritmi iterativi, ma che risolvono il problema all'esatto.

Questo metodo prende il nome di **normal equations** e minimizza $J(\theta)$ calcolando le derivate rispetto ai parametri θ_j e ponendole uguali a zero. Prima di procedere con tale derivazione sono richiamate alcune proprietà del calcolo matriciale.

3.2.1 Proprietà della derivazione matriciale

Per una funzione $f : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$ che è definita a partire dall'insieme delle matrici $m \times n$ verso l'insieme dei numeri reali, definiamo la derivata di f rispetto alla matrice A come:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \quad (11)$$

Dunque lo stesso **gradiente** $\nabla_A f(A)$ è una matrice $m \times n$ in cui il generico elemento (i, j) è $\frac{\partial f}{\partial A_{ij}}$.

A titolo di **esempio**, si supponga che $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ sia una matrice 2×2 e che la funzione $f : \mathbb{R}^{2 \times 2} \mapsto \mathbb{R}$ abbia la seguente forma:

$$f(A) = \frac{5}{2}A_{11}^2 + 3A_{22} + A_{12}A_{21} \quad (12)$$

Calcolando il gradiente di tale funzione rispetto ad A otteniamo:

$$\nabla_A f(A) = \begin{bmatrix} 5A_{11} & A_{21} \\ A_{12} & 3 \end{bmatrix} \quad (13)$$

Si introduce poi l'operatore **traccia**, scritto "tr". Per una matrice A (quadrata) di dimensioni $n \times n$, la traccia di A è definita **come la sommatoria degli elementi sulla diagonale principale**:

$$tr A = \sum_{i=1}^n A_{ii} \quad (14)$$

Se la matrice A è di dimensione 1×1 , cioè coincide con un numero reale a , vale che $tra = a$. La traccia di una matrice ha la proprietà che per due matrici A e B , il cui prodotto AB è una matrice quadrata, si può scrivere $trAB = trBA$. Come **corollario** di questa proprietà, abbiamo che:

$$\begin{aligned} trABC &= trCAB = trBCA, \\ trABCD &= trDABC = trCDAB = trBCDA. \end{aligned} \quad (15)$$

Per la dimostrazione della proprietà 15 si supponga di avere due matrici A e B di dimensioni rispettivamente di $K \times L$ e $L \times K$. Si effettuano i seguenti passi:

$$\begin{aligned}
 \text{tr}(AB) &= \sum_{k=1}^K (AB)_{kk} \\
 &= \sum_{k=1}^K \sum_{l=1}^L A_{kl} B_{lk} \\
 &= \sum_{l=1}^L \sum_{k=1}^K B_{lk} A_{kl} \\
 &= \sum_{l=1}^L (BA)_{ll} \\
 &= \text{tr}(BA)
 \end{aligned} \tag{16}$$

Occorre introdurre anche le seguenti proprietà:

$$\text{tr} A = \text{tr} A^T \tag{17}$$

$$\text{tr}(A + B) = \text{tr} A + \text{tr} B \tag{18}$$

$$\text{tr} \alpha A = \alpha \text{tr} A \tag{19}$$

Per la dimostrazione della proprietà 17 basta ricordare che la traccia di una matrice è pari alla sommatoria degli elementi sulla diagonale principale. Poichè sia A che A^T hanno gli stessi elementi diagonali, la proprietà risulta soddisfatta.

Prese due matrici A e B di dimensione $n \times n$, per la dimostrazione della proprietà 18:

$$\begin{aligned}
 \text{tr}(A + B) &= \sum_{i=1}^n (A + B)_{ii} \\
 &= \sum_{i=1}^n (A_{ii} + B_{ii}) \\
 &= \sum_{i=1}^n A_{ii} + \sum_{i=1}^n B_{ii} \\
 &= \text{tr}(A) + \text{tr}(B)
 \end{aligned} \tag{20}$$

Data una matrice A di dimensione $n \times n$, per dimostrare la proprietà 19:

$$\begin{aligned}
 \text{tr} \alpha A &= \sum_{i=1}^n (\alpha A)_{ii} \\
 &= \sum_{i=1}^n \alpha A_{ii} \\
 &= \alpha \sum_{i=1}^n A_{ii} \\
 &= \alpha \text{tr} A
 \end{aligned} \tag{21}$$

Si enunciano adesso alcune proprietà relative alla derivazione matriciale:

$$\nabla_A \text{tr} AB = B^T \quad (22)$$

$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T \quad (23)$$

$$\nabla_A \text{tr} ABA^T C = CAB + C^T AB^T \quad (24)$$

$$\nabla_A |A| = |A|(A^{-1})^T \quad (25)$$

Si noti che la proprietà 25 vale solo se la matrice quadrata A non è singolare, ovvero ha determinante diverso da zero ($|A| \neq 0$).

Per completezza si dimostrano adesso alcune delle precedenti proprietà. Per quanto riguarda la proprietà 22, date le matrici A e B di dimensione rispettivamente $n \times m$ e $m \times n$, si può scrivere:

$$\begin{aligned} \text{tr} AB &= \text{tr} \begin{bmatrix} \leftarrow & \vec{a}_1 & \rightarrow \\ \leftarrow & \vec{a}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \vec{a}_n & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ b_1 & b_2 & \dots & b_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \\ &= \text{tr} \begin{bmatrix} \vec{a}_1^T \vec{b}_1 & \vec{a}_1^T \vec{b}_2 & \dots & \vec{a}_1^T \vec{b}_n \\ \vec{a}_2^T \vec{b}_1 & \vec{a}_2^T \vec{b}_2 & \dots & \vec{a}_2^T \vec{b}_n \\ \vdots & & \ddots & \vdots \\ \vec{a}_n^T \vec{b}_1 & \vec{a}_n^T \vec{b}_2 & \dots & \vec{a}_n^T \vec{b}_n \end{bmatrix} \\ &= \sum_{i=1}^m a_{1i} b_{i1} + \sum_{i=1}^m a_{2i} b_{i2} + \dots + \sum_{i=1}^m a_{ni} b_{in} \\ &\implies \frac{\partial \text{tr} AB}{\partial a_{ij}} = b_{ji} \\ &\implies \nabla_A \text{tr} AB = B^T \end{aligned} \quad (26)$$

Per la proprietà 23, data una matrice A di dimensione $n \times n$, la dimostrazione si ottiene nel seguente modo:

$$\begin{aligned} \nabla_{A^T} f(A) &= \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{21}} & \dots & \frac{\partial f}{\partial A_{p1}} \\ \frac{\partial f}{\partial A_{12}} & \frac{\partial f}{\partial A_{22}} & \dots & \frac{\partial f}{\partial A_{n2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{1m}} & \frac{\partial f}{\partial A_{2m}} & \dots & \frac{\partial f}{\partial A_{nm}} \end{bmatrix} \\ &= (\nabla_A f(A))^T \end{aligned} \quad (27)$$

Si noti che aver preso una matrice A di dimensione $n \times m$ è un'assunzione valida se si suppone che $B \in \mathbb{R}^{n \times m}$. Infatti, il prodotto AB risulta essere una matrice quadrata a cui può essere applicato l'operatore traccia e per cui vale la seguente definizione $f(A) = \text{tr} AB$.

La proprietà 25 può essere ricavata a partire dalla definizione di **inversa** di una matrice. Sia A' la matrice dei cofattori, ossia la matrice in cui l'elemento (i, j) è dato dal prodotto di $(-1)^{i+j}$ e del determinante della matrice quadrata risultante dall'eliminazione della riga i e della colonna j da A . Poichè $A^{-1} = (A')^T / |A|$, ricordando che il determinante di A può essere riscritto come $|A| = \sum_j A_{ij} A'_{ij}$ e dato che $(A')_{ij}$ non dipende da A_{ij} si ottiene il risultato desiderato.

3.3 Metodo dei minimi quadrati rivisitato

Attraverso le proprietà illustrate nel paragrafo precedente, si è adesso in grado di procedere alla ricerca in forma chiusa dei valori di θ che minimizzano la funzione costo $J(\theta)$, che deve essere ridefinita in forma matriciale. A partire dal training set si definisce la matrice X di dimensione $m \times n$ che presenta sulle righe gli m esempi di training.

$$X = \begin{bmatrix} \dots & (x^{(1)})^T & \dots \\ \dots & (x^{(2)})^T & \dots \\ & \vdots & \\ \dots & (x^{(m)})^T & \dots \end{bmatrix} \quad (28)$$

Si definisce il vettore y di dimensione $m \times 1$ come il vettore contenente le variabili target corrispondenti agli m esempi di training.

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad (29)$$

Poichè vale $h_\theta(x^{(i)}) = (x^{(i)})^T \theta$ si può facilmente verificare che:

$$\begin{aligned} X\theta - y &= \begin{bmatrix} (x^{(1)})^T \theta \\ (x^{(2)})^T \theta \\ \vdots \\ (x^{(m)})^T \theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix} \end{aligned} \quad (30)$$

Poichè per un vettore z vale che $z^T z = \sum_i z_i^2$, si può riscrivere la funzione $J(\theta)$ come:

$$\begin{aligned} \frac{1}{2} (X\theta - y)^T (X\theta - y) &= \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned} \quad (31)$$

Combinando le proprietà 23 e 24 si ottiene che:

$$\nabla_{A^T} \text{tr} A B A^T C = B^T A^T C^T + B A^T C \quad (32)$$

Derivando J rispetto a θ si ottiene la seguente espressione:

$$\begin{aligned}
 \nabla_{\theta} J(\theta) &= \nabla_{\theta} \frac{1}{2} (X\theta - y)^T (X\theta - y) \\
 &= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \theta^T X^T y - y^T X \theta + y^T y) \\
 &= \frac{1}{2} \nabla_{\theta} \text{tr}(\theta^T X^T X \theta - \theta^T X^T y - y^T X \theta + y^T y) \\
 &= \frac{1}{2} \nabla_{\theta} (\text{tr} \theta^T X^T X \theta - 2 \text{tr} y^T X \theta) \\
 &= \frac{1}{2} (X^T X \theta + X^T X \theta - 2 X^T y) \\
 &= X^T X \theta - X^T y
 \end{aligned} \tag{33}$$

Nel terzo passaggio si è usata la proprietà per cui la traccia di un numero reale è uguale al numero reale stesso. Nel quarto passaggio si è usata la proprietà per cui $\text{tr} A = \text{tr} A^T$. Infine, nel quinto passaggio è usata la proprietà 32 con $A^T = \theta$, $B = B^T = X^T X$, $C = I$ e la proprietà 22.

Per minimizzare la funzione costo $J(\theta)$ si deve porre la sua derivata uguale a zero, ottenendo così l'espressione delle **normal equations**:

$$X^T X \theta = X^T y \tag{34}$$

Da cui deriva che il valore di θ che minimizza il valore di $J(\theta)$ è dato in forma chiusa dall'equazione:

$$\theta = (X^T X)^{-1} X^T y \tag{35}$$

Traducendo la formula 35 in codice MATLAB, facendo riferimento ai dati dell'esempio visto in precedenza, otteniamo:

```

1 %Fitting con normal equations
2 theta_ne = (x_train' * x_train) \ (x_train' * y_train);

```

Codice MATLAB 4: Algoritmo delle normal equations

In figura 9 è possibile notare come i parametri θ calcolati con i diversi metodi visti finora siano praticamente identici.

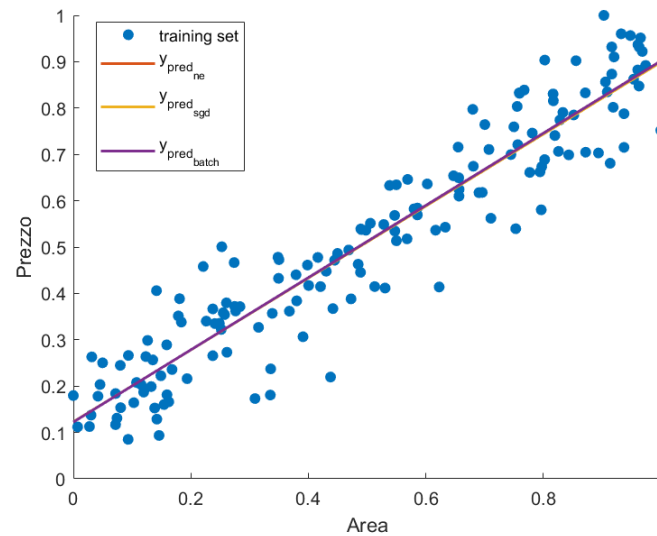


Figura 9: Regressione lineare semplice: sgd vs batch vs ne

4 Confronto tra discesa del gradiente e normal equations applicato ad un dataset per prevedere il costo dell'assicurazione medica

Per mostrare il comportamento dei due algoritmi precedentemente trattati si fa ora riferimento a un insieme di dati personali relativi alle spese mediche negli Stati Uniti. Le colonne del dataset sono:

- età: età del beneficiario principale
- sesso: sesso contraente assicurativo: femmina, maschio
- bmi: indice di massa corporea, fornisce l'indice oggettivo del peso corporeo (kg/m^2) utilizzando il rapporto tra altezza e peso, idealmente da **18,5** a **24,9**
- figli: numero di figli coperti dall'assicurazione sanitaria / numero di persone a carico
- fumatore: il soggetto fuma?
- regione: area residenziale del beneficiario negli Stati Uniti, nord-est, sud-est, sud-ovest, nord-ovest.
- spese: spese mediche individuali fatturate dall'assicurazione sanitaria

L'obiettivo è **prevedere** i costi assicurativi per le spese medicali.

I dati sono estratti da un file .csv e gli attributi testuali sono convertiti in formato numerico. Viene poi utilizzato l'80% del dataset per effettuare il training del modello su MATLAB, mentre il restante 20% si usa per il testing. Ciò si traduce nel seguente frammento di codice:

```
1 %Lettura dataset
2 dataset = readtable('insurance.csv');
3 %Conversione feature testuali in numeriche
```

```

4 dataset = convertvars(dataset,{'sex','smoker','region'},'categorical');
5 dataset = convertvars(dataset,{'sex','smoker','region'},'double');
6 dataset = dataset{:,:};
7
8 [r,c] = size(dataset);
9
10 %Separo input e output
11 x = [ones(r,1) dataset(:,1:c-1)];
12 y = dataset(:,c);
13
14 %Normalizzo prezzo
15 y = log10(y);
16
17 %Divido dataset in training set e test set
18 th = floor(0.8 * length(x));
19 x_train = x(1:th,:);
20 y_train = y(1:th,:);
21 x_test = x(th+1:r,:);
22 y_test = y(th+1:r,:);

```

Codice MATLAB 5: Preparazione dati per l'esempio dell'assicurazione medicale

4.1 Complessità computazionale

4.1.1 Normal Equations

Il metodo delle normal equations, descritto dalla 35:

$$\theta = (X^T X)^{-1} X^T y$$

richiede il seguente numero di operazioni:

1. $X^T X$ richiede $O(n^2 m)$ operazioni, essendo $X \in \mathbb{R}^{m \times n}$
2. L'inversa $(X^T X)^{-1}$ richiede $O(n^3)$ operazioni, essendo $X^T X \in \mathbb{R}^{n \times n}$
3. $X^T y$ richiede $O(nm)$ operazioni per produrre una matrice $\mathbb{R}^{n \times 1}$
4. $((X^T X)^{-1})(X^T y)$ richiede $O(n^2)$ operazioni

Quindi la complessità totale è pari a:

$$O(n^2 m) + O(n^3) + O(nm) + O(n^2) = O(n^2 m + n^3)$$

4.1.2 Discesa del gradiente

La discesa del gradiente può essere descritta in forma vettorizzata dalla seguente regola di aggiornamento:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta) = \theta - \eta (X^T X \theta - X^T y)$$

e richiede il seguente numero di operazioni:

1. $X^T X$ richiede $O(n^2 m)$ operazioni
2. Per ogni iterazione bisogna calcolare
 - $(X^T X)(\theta)$ che richiede $O(n^2 m)$
 - $(X^T)(y)$ che richiede $O(nm)$
 - L'aggiornamento di θ con queste matrici calcolate richiede $O(n)$

Quindi la complessità totale è pari a:

$$O(n^2m) + k(O(n^2m) + O(nm) + O(n)) = O(kn^2m)$$

dove k è il numero di iterazioni.

4.1.3 Confronto su MATLAB

Risulta che le normal equations hanno una complessità di $O(n^3)$, mentre la discesa del gradiente ha una complessità di $O(kn^2m)$. Viene qui riportata l'implementazione dei due algoritmi in MATLAB:

```

1  %-----File:main_regressione_lineare_insurance.m-----%
2  %Inizializzo parametri theta casualmente
3  m = length(x_train(1,:));
4  theta_batch_opt = rand(m,1);
5
6  %Fitting dei parametri attraverso discesa del gradiente batch
7  lr = 0.001;
8  it = 80;
9  [theta_batch_opt, cost_hist_batch_opt] = ...
    batch_gradient_descent_ottimizzato(theta_batch_opt, x_train, y_train, 0.0007, 500);
10
11 %Calcolo dei parametri attraverso le normal equations
12 theta_ne = (x_train' * x_train) \ (x_train' * y_train);
13
14 %-----File:batch_gradient_descent_ottimizzato.m-----%
15 function [theta, cost_hist, n] = ...
    batch_gradient_descent_ottimizzato(theta, x, y, lr, it)
16 cost_hist = zeros(it,1);
17 for n = 1 : it
18     theta = theta - lr.* ( x'*x*theta - x' * y );
19     cost_hist(n) = cost_func(theta, x, y);
20 end
21 end

```

Codice MATLAB 6: Calcolo dei parametri con discesa del gradiente e normal equations

Nel caso delle *normal equations* la complessità di $O(n^3)$ è data dall'uso **dell'algoritmo di eliminazione di Gauss** implementato tramite l'operatore backslash (\) in MATLAB.

Tuttavia, occorre fare alcune osservazioni in merito alle prestazioni dei due metodi. La discesa del gradiente, infatti, risulta essere il più efficiente tra i due nel caso in cui il numero di feature n sia elevato, per esempio $n = 10000$. Per valori più piccoli di n , come nel caso in esame, dove $n = 7$, le prestazioni migliori si ottengono con il metodo delle normal equations. Di quest'ultima affermazione si può trovare riscontro usando le funzioni *tic* e *toc* di MATLAB che ci permettono di misurare il tempo impiegato dai due metodi per calcolare il θ ottimale rispetto allo stesso training set. In particolare, si noti come **il metodo delle normal equations è quasi tre volte più veloce di quello della discesa del gradiente**:

```

1  tempo_batch_opt:
2      0.0382
3  tempo_ne:
4      0.0141
5  tempo_batch_opt/tempo_ne:
6      2.7121

```

Codice MATLAB 7: Tempi di esecuzione e speed-up algoritmo Batch vs Normal equations

Occorre però sottolineare come **i risultati ottenuti siano influenzati**, oltre che dalla complessità dei due algoritmi, da altri fattori come **le prestazioni della macchina** su cui girano, dal contesto di esecuzione, dal carico del sistema, etc., e quindi in esecuzioni successive potrebbero essere ottenuti risultati diversi.

4.2 Metriche di valutazione per la Regressione Lineare

Per valutare le prestazioni di un modello è necessario avere delle metriche di valutazione, cioè un numero reale che ci consenta di valutarne il comportamento.

4.2.1 Mean Squared Error

Il mean squared error è una metrica utilizzata solitamente per **valutare le prestazioni di un modello di regressione lineare**, fornendo un'idea sull'errore del modello nelle sue predizioni.

$$MSE(x, h) = \frac{1}{m} \sum_{i=1}^m \left(h(x^{(i)}) - y^{(i)} \right)^2$$

Nel caso di studio trattato in questo capitolo è stato proprio utilizzato il MSE per valutare le prestazioni del modello e sono stati ottenuti i seguenti risultati:

```
1 mse_batch_opt = mse(h(x_test, theta_batch_opt), y_test);
2 mse_ne = mse(h(x_test, theta_ne), y_test);
3 mse_batch_opt =
4     0.0095
5 mse_ne =
6     0.0095
```

Codice MATLAB 8: Valutazione del Mean Squared Error nel caso di discesa del gradiente e normal equations

Dall'output ottenuto nella simulazione MATLAB, si può apprezzare come i due metodi usati abbiano prestazioni identiche nell'applicazione considerata. La scelta sul metodo da utilizzare sarà quindi legata soltanto alla maggiore efficienza computazionale del metodo delle normal equations, essendo $n = 7$.

4.3 Considerazione finale

Volendo riassumere le considerazioni fatte nel corso di questo capitolo, in tabella sono sottolineate ulteriormente le differenze tra il metodo della discesa del gradiente e quello delle normal equations:

N.	Gradient Descent	Normal Equation
1	Scelta sperimentale del Learning Rate	Non c'è bisogno di scegliere il Learning Rate
2	Approccio Iterativo: potrebbe richiedere molte iterazioni	Approccio Analitico: non sono richieste iterazioni
3	Funziona bene quando il numero di feature n è elevato	Lento per n elevato ≈ 10000
4	Richiede di normalizzare le feature	Non richiede di normalizzare le feature

5 Interpretazione probabilistica

Si vuole ora giustificare l'espressione della funzione costo 3 del metodo dei minimi quadrati, utilizzata per la regressione lineare, attraverso un approccio probabilistico. Si assuma che le variabili target e gli input del modello siano legati dalla relazione:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \quad (36)$$

dove $\epsilon^{(i)}$ fa riferimento a un termine **errore** legato agli effetti non modellati del modello lineare come **rumore o eventuali feature trascurate nei dati**. Modelliamo il termine errore attraverso una variabile aleatoria gaussiana con media μ nulla, varianza σ^2 e una pdf uguale a:

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right) \quad (37)$$

Assumiamo inoltre che le variabili aleatorie $\epsilon^{(i)}$ siano tra loro indipendenti e identicamente distribuite (**IID**). La scelta di modellare l'errore come gaussiano viene fatta perché matematicamente conveniente e spesso **conforme con la realtà**. Esso è spesso dovuto alla somma di tanti effetti tra loro indipendenti e quindi per il teorema centrale del limite ha pdf gaussiana.

Dato x allora y è a sua volta una variabile aleatoria gaussiana con media pari $\theta^T x^{(i)}$, varianza σ^2 e una pdf uguale a:

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \quad (38)$$

dove θ non è una variabile aleatoria, ma un parametro.

Estendendo l'equazione 38 a tutto il training set otteniamo $p(y|X; \theta)$. Se viene evidenziata la dipendenza da θ , questa funzione viene detta funzione di **verosimiglianza**:

$$L(\theta) = \prod_{i=1}^m p(y^{(i)}|x^{(i)}; \theta) = L(\theta, X, y) = p(y|X; \theta) \quad (39)$$

Data X , per l'ipotesi di IID degli errori $\epsilon^{(i)}$ e quindi anche delle $y^{(i)}$, è possibile fattorizzare la pdf e la 39 diventa:

$$L(\theta) = \prod_{i=1}^m p(y^{(i)}|x^{(i)}; \theta) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \quad (40)$$

Per il principio di massima verosimiglianza il nostro obiettivo è scegliere i parametri θ in modo da massimizzare $L(\theta)$, ossia rendere la probabilità che la predizione sia corretta più alta possibile. Il ragionamento continua a valere se proviamo a massimizzare una qualunque funzione strettamente crescente di $L(\theta)$ come:

$$\begin{aligned}l(\theta) &= \log(L(\theta)) \\&= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\&= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2\end{aligned}\tag{41}$$

Massimizzare $l(\theta)$ equivale a minimizzare:

$$\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2\tag{42}$$

In questo modo si è ottenuta la stessa funzione costo $J(\theta)$ del metodo LMS 3.

6 Classificazione e Regressione Logistica

Il problema della classificazione è del tutto analogo a quello di regressione, con la differenza che la variabile target y può assumere un insieme discreto di valori. Nel caso in cui y assume solo il valore 0 o 1 si parla di classificazione binaria.

Si tenga a mente che il problema di classificazione può essere affrontato allo stesso modo della regressione, ignorando il fatto che y sia discreta. Tuttavia i risultati ottenibili sarebbero scadenti, non avendo più senso utilizzare una funzione ipotesi $h_\theta(x)$ che assuma valori al di fuori di $[0, 1]$ sapendo che $y \in \{0, 1\}$. Di conseguenza si modifica la forma della funzione ipotesi, che diventa:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (43)$$

dove:

$$g(z) = \frac{1}{1 + e^{-z}} \quad (44)$$

è chiamata **funzione logistica o sigmoide**, il cui grafico è mostrato in figura 10.

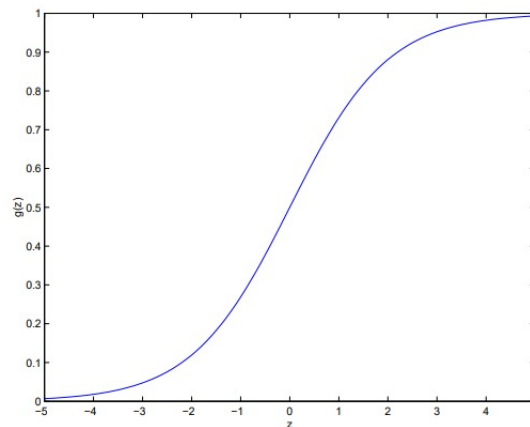


Figura 10: Sigmoide

Si noti che $\lim_{z \rightarrow +\infty} g(z) = 1$ mentre $\lim_{z \rightarrow -\infty} g(z) = 0$ e quindi $h_\theta(x)$ assumerà solo valori tra 0 e 1.

Si pone $x_0 = 1$ in modo che:

$$\theta^T x = \theta_0 + \sum_{j=1}^n \theta_j x_j \quad (45)$$

Dato il modello di **regressione logistica** ci si chiede come sia possibile calcolare i parametri θ . A tal proposito si procede in maniera analoga a quanto visto con la regressione ai minimi quadrati, in particolare dalla sua interpretazione probabilistica, cioè a partire dalla funzione di massima verosimiglianza.

Si assume che:

$$P(y = 1|x; \theta) = h_\theta(x) \quad (46)$$

$$P(y = 0|x; \theta) = 1 - h_\theta(x) \quad (47)$$

Sapendo che la densità $p(x)$:

$$p(x) = P(X = x) \quad (48)$$

Allora è possibile compattare la 46 e la 47 in:

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y} \quad (49)$$

Assumendo che gli esempi di training siano stati generati in modo indipendente, possiamo quindi scrivere la funzione di verosimiglianza:

$$\begin{aligned} L(\theta) &= p(y|X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)}|x^{(i)}; \theta) \\ &= \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned} \quad (50)$$

Invece della 50 possiamo massimizzare la **log likelihood**:

$$\begin{aligned} l(\theta) &= \log(L(\theta)) \\ &= \log \left(\prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \right) \\ &= \sum_{i=1}^m y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \end{aligned} \quad (51)$$

Analogamente alla derivazione nel caso della regressione lineare, per massimizzare la likelihood si utilizza l'ascesa del gradiente.

$$\theta := \theta + \eta \frac{\partial}{\partial \theta_j} l(\theta) \quad (52)$$

Si noti il segno + rispetto all'equazione 5, dovendo ora massimizzare una funzione.

A partire da un solo esempio di training (x, y) e derivando, si ottiene la regola dell'ascesa stocastica del gradiente:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} l(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \end{aligned} \quad (53)$$

$$\begin{aligned} &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_\theta(x)) x_j \end{aligned} \quad (54)$$

Dove nella 53 è stata utilizzata la derivata $g'(z)$ della funzione sigmoide:

$$\begin{aligned}
 g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\
 &= \frac{1}{(1 + e^{-z})^2} e^{-z} \\
 &= \frac{1}{(1 + e^{-z})} \left(1 - \frac{1}{1 + e^{-z}} \right) \\
 &= g(z)(1 - g(z))
 \end{aligned} \tag{55}$$

E dunque la 52 diventa:

$$\theta_j := \theta_j + \eta((y^{(i)} - h_{\theta}(x^{(i)}))x_j^{(i)}) \tag{56}$$

Il risultato ottenuto è identico alla regola LMS nella 8, ma non si tratta dello stesso algoritmo, essendo ora $h_{\theta}(x^{(i)})$ una funzione non lineare di $\theta^T x^{(i)}$.

Per l'implementazione dell'algoritmo si è tenuto in mente come massimizzare $l(\theta)$ equivalga a minimizzare con la discesa del gradiente: $J(\theta) = -l(\theta)$.

6.1 Metriche di valutazione per la Regressione Logistica

Nell'ambito della classificazione si definisce l'**accuracy** come:

$$accuracy = \frac{PredizioniCorrette}{PredizioniTotali}$$

Non sempre la scelta della metrica da utilizzare è immediata e in alcune situazioni diventa difficile trovarne una appropriata per il modello in esame. Questo è ciò che succede quando si lavora con classi distorte ("**skewed**"). Si supponga a tal proposito di avere un modello di regressione logistica che restituisce $y = 1$ quando il paziente considerato è positivo al Covid19 e 0 altrimenti. Il fatto di avere un accuracy elevato sul test set potrebbe non essere sufficiente a comprendere la bontà del nostro algoritmo. Si immagini infatti che soltanto lo 0.50% dei pazienti sia positivo e sia data la funzione *PredizioneCovid*:

```

1 function y = PredizioneCovid(x)
2     y = 0;
3     return

```

Codice MATLAB 9: Esempio di classe Skewed nel caso di predizione del Covid19

Osserviamo come quest'ultima ignori l'input x e predica sempre 0. In questo caso il fatto di effettuare il 99.5% delle diagnosi correttamente non è più impressionante

Quando il numero di esempi di training di una classe è molto più grande di un'altra classe si parla di classi distorte. In questi casi l'accuracy da sola non è più una metrica attendibile per valutare la bontà del modello. Nel caso preso in esame verrà affiancata alla "**Matrice di Confusione**" per la valutazione delle prestazioni del modello.

6.1.1 Matrice di confusione

Un modo migliore per valutare le prestazioni di un classificatore è osservare la **matrice di confusione**. Ciascuna riga della matrice di confusione rappresenta la classe attuale mentre ogni colonna rappresenta la classe predetta. Sulla diagonale principale è indicato il numero di elementi della classe attuale predetti correttamente. Nella generica posizione (i, j) con

$i \neq j$ è invece indicato il numero di elementi della classe i predetti erroneamente come appartenenti alla classe j .

Nel caso binario la matrice di confusione assume la seguente forma:

		Prediction class	
		Positive	Negative
Actual class	Positive	True Positive TP	False Negative FN
	Negative	False Positive FP	True Negative TN

Per quanto la matrice di confusione fornisca molte informazioni, ci sono situazioni in cui si preferisce una metrica più concisa, come **Precision** e **Recall**.

Si supponga che $y = 1$ in presenza di un paziente positivo al covid. La Precision indica di tutti i pazienti che sono stati predetti come positivi quale frazione di essi abbia realmente il covid:

$$Precision = \frac{TP}{\#PredictedPositive} = \frac{TP}{TP + FP}$$

La Recall indica qual è la frazione dei pazienti che hanno realmente il covid predetta come positiva.

$$Recall = \frac{TP}{\#ActualPositive} = \frac{TP}{TP + FN}$$

Spesso è conveniente combinare precision e recall in una singola metrica, detta **F1 score**, in modo da rendere possibile il confronto tra più classificatori. Quest'ultima è la media armonica di precision e recall:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall}$$

Differentemente dalla media che tratta i valori tutti allo stesso modo, la media armonica pesa maggiormente i valori piccoli in modo che il classificatore restituirà un F1 score elevato soltanto se sia recall che precisione sono alti. Si tenga a mente che spesso è necessario effettuare un trade-off tra precision e recall, che è strettamente legato al tipo di applicazione. Nell'ambito di diagnostica tumorale per esempio si preferisce avere una recall alta, volendo massimizzare la quantità di pazienti realmente malata diagnosticata, minimizzando i falsi negativi al costo di qualche falso allarme.

7 Applicazione MATLAB della Regressione Logistica applicato al set di dati per la predizione del SARS-CoV2 (Covid19)

Si consideri il seguente dataset che presenta come feature: sintomi, malattie critiche e comportamenti da SARS-CoV2 (Covid19):

- Breathing Problem: Difficoltà respiratorie, respiro affannoso, respiro pesante, sensazione di respiro incompleto
- Fever: Temperatura corporea $> 37.5\text{ }^{\circ}\text{C}$ ($> 99.5\text{ }^{\circ}\text{F}$)
- Dry Cough: Tosse secca, irritativa, persistente, non accompagnata dalla presenza di secrezioni catarrali
- Sore throat: Dolore alla gola, difficoltà a deglutire
- Running Nose: Perdita eccessiva di muco dal naso, o dai seni paranasali, verso l'esterno o verso la gola
- Asma: Fiato corto dopo piccolo sforzo fisico, tosse frequente e raschiamento della gola
- Headache: Dolore diffuso a tutto il cranio, sensazione di costrizione e peso al vertice del cranio, pesantezza sul seno frontale
- Fatigue: Fatica, stress e spossatezza
- Gastrointestinal Problem: Diarrea, nausea, vomito, fastidi allo stomaco e all'intestino

Altre feature: *Chronic Lung Disease, Heart Disease, Diabetes, Hyper Tension, Abroad Travel, Contact with COVID Patient, Attended Large Gathering, Visited Public Exposed Places, Family working in Public Exposed Places, Wearing Masks, Sanitization from Market.*

L' **obiettivo** del modello è quello di predire una certa probabilità di infezione al COVID-19 in base ai sintomi o alle malattie che il paziente manifesta e guidarlo a come comportarsi in caso di alta probabilità di infezione.

7.1 Machine Learning steps

A seguire sono descritti e implementati i vari *steps* a partire dalla preparazione dei dati fino alla realizzazione di un'app Matlab che implementa il modello addestrato.

7.1.1 Raccolta e Preparazione dei dati

Si inizia dalla *raccolta e preparazione dei dati*. Il dataset viene mischiato per essere sicuri che sia il training set che il test set siano rappresentativi dei dati. Infatti, nel caso in cui il dataset fosse ordinato per classe, potremmo ottenere un modello che non generalizza bene. Si separano poi le variabili di input e output in conformità al modello, **aggiungendo una colonna pari a 1 alle variabili di input** e si effettua la divisione dei dati in training set e test set, come mostrato di seguito:

```
1 %Lettura file txt
2 dataset = readmatrix('data_covid.txt');
3
```

```

4 %Shuffle dataset
5 dataset = dataset(randperm(size(dataset, 1)), :);
6
7 %Divido variabili di input e output. In conformita' al modello aggiungo ...
  colonna pari a 1 alle variabili di input
8 [r,c] = size(dataset);
9 x = [ones(r,1) dataset(:,1:c-1)];
10 y = dataset(:,c);
11
12 %Splitto il dataset in training set (80%), test set (20%)
13 th = floor(0.8 * length(x(:,1)));
14 x_train = x(1:th,:);
15 y_train = y(1:th,:);
16 x_test = x(th+1 : length(x(:,1))),:);
17 y_test = y(th+1 : length(x(:,1)));

```

Codice MATLAB 10: Preparazione dei dati

7.1.2 Scelta del modello e fitting dei parametri

In questa fase viene scelto il modello di Regressione Logistica, assumendo la variabile target y un valore pari a 0 o a 1. Viene fatto poi il "fitting" che consiste nel calcolare i *parametri* θ_j del modello, come mostrato nel seguente codice Matlab:

```

1 %-----File:main.regressione.logistica.m -----%
2 %Fitting parametri del modello di regressione logistica
3 n = 1000;
4 alpha = 0.0001;
5 m = length(x(1,:));
6 theta = rand(m,1);
7 [theta,hist] = regressione_logistica(x_train,y_train,theta,alpha,n);
8
9 %-----File:regressione.logistica.m -----%
10 function [theta,hist] = regressione_logistica(x,y,theta,lr,n)
11 m = length(x(1,:));
12 hist = zeros(n,1);
13
14 %Discesa del gradiente per il modello di regressione logistica
15 %A ogni iterazione salvo il valore corrente della funzione costo
16 for i = 1:n
17     for j = 1:m
18         theta(j) = theta(j) - lr .* (h(x,theta) - y)' * x(:,j);
19     end
20     hist(i) = cost(theta,x,y);
21 end
22 end
23
24 %-----File:cost.m -----%
25 function [J] = cost(theta, x, y)
26 J = sum((-y.*log(h(x,theta)))-(1-y).*log(1-h(x,theta)));
27 end

```

Codice MATLAB 11: Fitting dei Parametri e valutazione della *loss function*

Il modello è addestrato con la funzione *regressione_logistica(x_train,y_train,theta,alpha,n)*. Per valutare quanto bene l'algoritmo modella il set di dati, viene memorizzato il valore della funzione costo $J(\theta)$ nelle varie iterazioni dell'addestramento, che indica la distanza tra i dati predetti dal modello e quelli attesi.

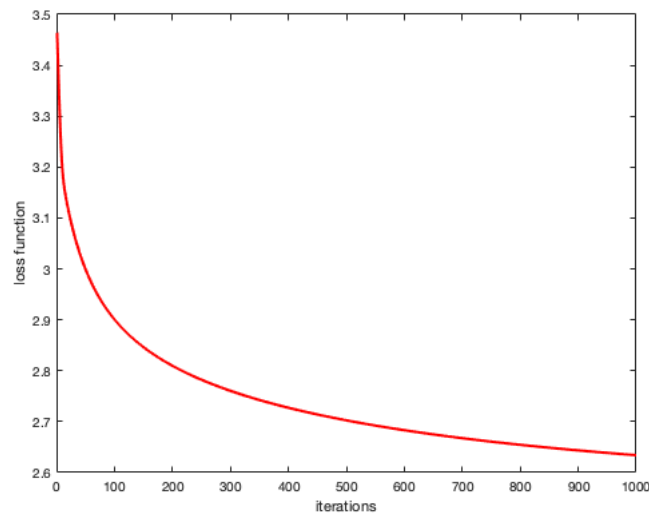


Figura 11: Andamento funzione costo

7.1.3 Valutazione delle prestazioni del modello

Si passa alla *valutazione delle prestazioni del modello sul test set* tramite le metriche di valutazione viste nel paragrafo 6.1 in termini di **accuracy** e delle altre metriche derivanti dalla **matrice di confusione**.

```
1 %Valuto le prestazioni del modello sul test set in termini di accuracy
2 p = predict(theta,x_test);
3 accuracy = sum(p == y_test)/length(x_test);
4
5 %Matrice di confusione
6 conf_matrix = confusionmat(y_test,double(p));
7 confusionchart(y_test,double(p)); %stampa della matrice di confusione
```

Codice MATLAB 12: Valutazione delle prestazioni del modello

```
1 Accuracy:
2     0.9724
3
4 Matrice di confusione:
5     187     23
6      7    870
```

Codice MATLAB 13: Output Accuracy e Matrice di Confusione

Dalla *Matrice di Confusione* in figura 12 otteniamo i seguenti valori:

- **Vero positivo (TP)** = 870; il che significa che 870 input positivi sono stati classificati correttamente dal modello.
- **Vero negativo (TN)** = 187; il che significa che 187 input negativi sono stati classificati correttamente dal modello
- **Falso positivo (FP)** = 23; il che significa che 23 input negativi sono stati erroneamente classificati come appartenenti alla classe positiva dal modello

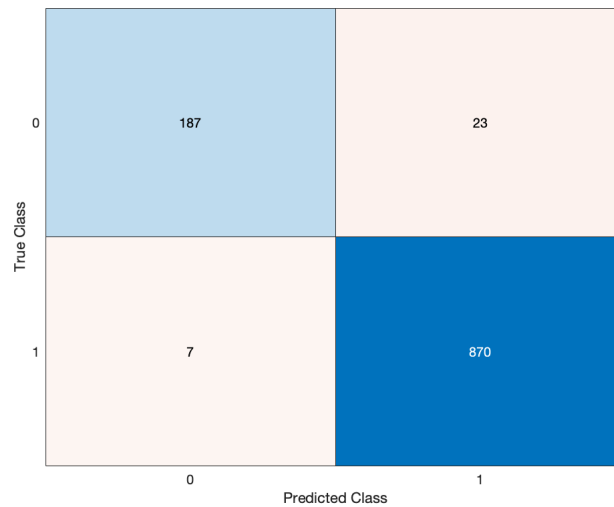


Figura 12: Matrice di Confusione

- **Falso negativo (FN) = 7**; il che significa che 7 input positivi sono stati erroneamente classificati come appartenenti alla classe negativa dal modello.

Sulla base dei risultati ottenuti possiamo affermare che il nostro classificatore ha buone prestazioni per il set di dati in esame, **tenendo conto del numero elevato di veri positivi e veri negativi**.

Dalla **matrice di confusione** è possibile allora valutare l'*Accuracy*, la *Precision*, la *Recall* e il *F1 Score*:

```

1  %Prelevamento di tp, tn, fp, fn
2  tp = conf_matrix(2,2);
3  fn = conf_matrix(2,1);
4  fp = conf_matrix(1,2);
5  tn = conf_matrix(1,1);
6
7  %Frazione di pazienti predetti positivi realmente positiva
8  precision = tp/(tp+fp);
9
10 %Frazione di pazienti predetti correttamente positivi
11 recall = tp/(tp+fn);
12
13 %Accuratezza del modello
14 accuracy = (tp + tn)/length(x_test(:,1));
15
16 f1_score = 2 * precision * recall / (precision+recall);

```

Codice MATLAB 14: Valutazione delle prestazioni attraverso la Matrice di Confusione

```

1  Accuracy:
2      0.9724
3
4  Precision:
5      0.9742
6

```

```
7 Recall:
8     0.9920
9
10 F1_SCORE:
11     0.9831
```

Codice MATLAB 15: Output Accuracy, Precision, Recall, F1-score calcolata con la matrice di confusione

Pertanto il 97% dei casi predetti positivi si è rivelato attualmente positivo e il 99% dei positivi è stato previsto positivo con successo dal modello.

La *Precision* è una metrica utile nei casi in cui i falsi positivi rappresentano un problema maggiore rispetto ai falsi negativi.

La *Recall* è una metrica utile nei casi in cui il falso negativo riveste un'importanza maggiore del falso positivo. Questo è vero specialmente nell'ambito medico in cui è preferibile dare un falso allarme, piuttosto che non diagnosticare casi positivi. Dunque la recall è la metrica più importante da considerare nella nostra applicazione, non volendo che un falso negativo possa diffondere il virus.

7.1.4 La Prediction

Si prova adesso a effettuare una predizione con il nostro modello. A tal proposito forniamo ad esso l'input, ossia i sintomi del paziente, per valutare la probabilità di infezione al COVID-19. Se il modello predice che il paziente *X* abbia una probabilità di infezione tra il 75% e il 100% allora verrà consigliato di consultare immediatamente un operatore sanitario e di rimanere presso il proprio domicilio. Se il sistema predice che il paziente *Y* abbia una probabilità di infezione tra il 50% e il 75% allora verrà consigliato di curarsi in casa e di chiamare il medico di fiducia. Di seguito sono proposti due esempi di pazienti con sintomi diversi:

```
1 %Esempio predizioni (aggiungo 1 in prima posizione in conformita' al
2 %modello)
3 %I sintomi e le malattie che un paziente P1 inserisce sono: Breathing ...
   Problem, Fever, Sore Throat, Fatigue, Contact with COVID Patient, ...
   Visited Public Exposed Places, Family working in Public Exposed Places, ...
   Wearing Masks
4
5 x_trial1 = [1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0];
6 p1 = h(x_trial1,theta);
7 disp('Probabilita' positivita' p1: ')
8 disp(p1)
9
10 %I sintomi e le malattie che un paziente P2 inserisce sono: Breathing ...
   Problem, Fever, Sore Throat, Diabetes, Fatigue, Contact with COVID ...
   Patient, Wearing Masks, Family working in Public Exposed Places
11 x_trial2 = [1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0];
12 p2 = h(x_trial2,theta);
13 disp('Probabilita' positivita' p2: ')
14 disp(p2)
```

Codice MATLAB 16: Due casi di predizione

```
1 Probabilita' positivita' p1:
2     0.5894
3
4 Probabilita' positivita' p2:
5     0.9613
```

Codice MATLAB 17: Predizione della P1 e P2

Si noti come il paziente P1 abbia il *58%* di probabilità di essere **potenzialmente positivo al SARS-CoV2(Covid19)** mentre il paziente P2 abbia il *96%* di probabilità di essere **potenzialmente positivo al SARS-CoV2(Covid19)**.

8 Overfitting e Underfitting

Dopo aver trattato teoricamente e implementato i modelli di Regressione Lineare e Regressione Logistica, vale la pena fare un accenno ai possibili problemi che possono deteriorarne le prestazioni.

L'**overfitting** si verifica quando un modello ha ottime prestazioni sul training set, ma non generalizza bene sul test set. Ciò accade per esempio quando utilizziamo un polinomio di grado elevato per modellare dati lineari come in figura 13.

L'**underfitting** si verifica invece quando il modello scelto è troppo semplice per rappresentare i dati come in figura 16.

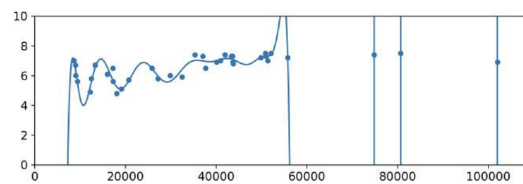


Figura 13: Esempio overfitting

Ad esempio, nel **deep learning**, in cui si lavora con **reti neurali** costituite da centinaia di migliaia di parametri, il modello è in grado di apprendere pattern molto complessi dei dati di training. Laddove però il dataset sia troppo piccolo o rumoroso, il modello probabilmente non generalizzerà bene su nuove istanze. Per comprendere la causa all'origine dell'overfitting e underfitting possiamo immaginare che l'informazione dei dati di training sia fatta da una parte utile, detta segnale, e una parte di rumore. Il segnale fa riferimento a quelle informazioni generali del training set che permettono al modello di generalizzare su dati non visti. Il rumore invece è specifico dei dati di training e fa riferimento a informazioni casuali derivanti dai dati del mondo esterno e che non aiuta il modello a generalizzare su nuovi dati. Il nostro obiettivo è addestrare un modello in maniera tale da minimizzare la funzione costo sul training set. In realtà però per evitare overfitting e underfitting dobbiamo **monitorare** l'andamento della prestazioni del modello su un set di dati ignoto, detto validation set.

Si utilizzano quindi le curve di apprendimento, ossia un grafico che rappresenta il valore della **loss** a ogni iterazione sul training e **validation set**, come in figura 14.

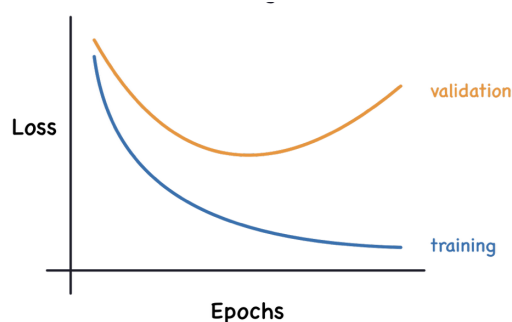


Figura 14: Curve di apprendimento

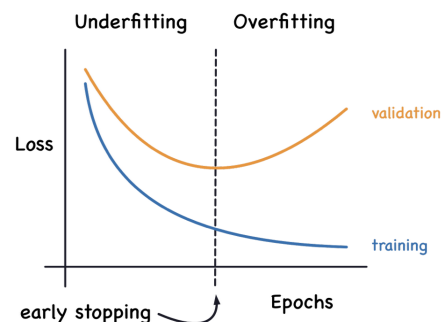


Figura 15: Early Stopping

In quest'ultima si può osservare come il valore della funzione costo sul training set scenda all'aumentare delle epoche, sia che il modello apprenda rumore, sia che apprenda segnale. La loss sul validation set diminuisce soltanto quando il modello apprende il segnale.

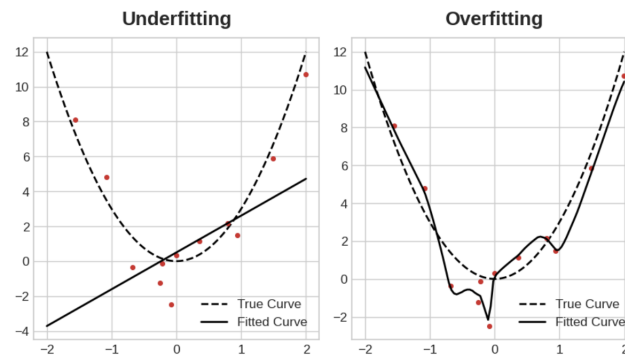


Figura 16: Underfitting e Overfitting

Quindi, quando il modello apprende segnale entrambe le curve si abbassano, mentre quando viene appreso rumore si crea un gap tra le due, che rappresenta la quantità di rumore imparata dal modello.

Idealmente vorremmo che il modello apprenda solo segnale e nessun rumore. Tuttavia nella pratica abbiamo bisogno di un trade-off. Possiamo aumentare la quantità di segnale al costo di maggior rumore, interrompendo però l'addestramento appena la loss sul validation set inizierà a salire.

L'overfitting è quindi legato all'apprendimento di troppo rumore, mentre l'underfitting è un problema che si verifica quando viene appreso troppo poco segnale e quindi la loss è alta sia nel validation set che nel test set. Bisogna capire, a partire dalle curve di apprendimento, il numero ottimale di epoche di addestramento a cui fermarsi come visibile in figura 15.

Sono infine proposte alcune soluzioni per risolvere questi due fenomeni:

- Overfitting:
 - Semplificare il modello, riducendo per esempio il numero di feature oppure di parametri
 - Aumentare la dimensione del dataset
 - Ridurre il rumore nel dataset, rimuovendo eventuali errori nei dati o outliers
 - Utilizzare un parametro di regolarizzazione λ per ridurre il valore dei parametri θ
- Underfitting
 - Scelta di un modello più complesso e con più parametri
 - Migliorare la qualità delle feature fornite al modello
 - Ridurre il valore del parametro di regolarizzazione λ

8.1 Rilevazione dell'overfitting e dell'underfitting

Dunque, riassumendo quanto detto prima, dato un dataset lo si suddivide in **training set** e in **test set** e si procede come nello schema in figura 17.

Si valuta l'errore sul training set e sul test set in modo tale che se:

- Se l'errore sul training set è alto, indipendentemente dall'errore sul test set c'è sicuramente un underfitting (problema troppo generalizzato)

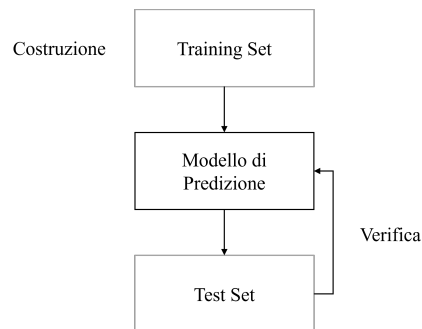


Figura 17: Costruzione e verifica del modello di predizione

- Se l'errore sul training set è basso mentre quello sul test set è elevato si tratta di overfitting
- Se l'errore è basso sia su training set che test set allora il test è superato

Proviamo a simulare una situazione di **underfitting** in un modello di regressione lineare addestrato con sgd:

```

1  %Generazione dataset
2  x = 50*rand(250,1);
3  y = x.^3 + 50^2 .* randn(250,1);
4  x = [ones(length(x),1) x];
5
6  %Parametri theta inizializzati casualmente
7  theta_sgd = rand(1,length(x(1,:)));
8  lr = 0.000001;
9  it = 100;
10
11 %Fitting parametri con sgd
12 [theta_sgd, cost_hist_sgd, n_sgd] = sgd(theta_sgd,x,y,lr,it);
  
```

Codice MATLAB 18: Esempio underfitting

Il risultato di tale simulazione può essere osservato nella Figura 18, in cui si nota che i valori predetti dal modello costruito approssimano in maniera errata i dati del training set:

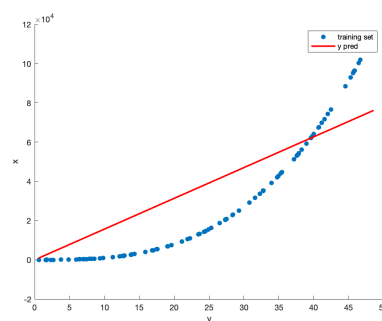


Figura 18: Esempio di Underfitting

9 Proof of Concept: da MATLAB App a Web App for Browser

Dopo l'analisi matematica ed il calcolo numerico su MATLAB del modello di predizione è possibile applicare in un caso d'uso nell'ambito Healthcare (sanità) quanto fatto finora. Infatti, con un'enorme quantità di dati medici generati, le organizzazioni sanitarie possono sfruttare questa mole di conoscenza per fornire risultati migliori con maggiore precisione. Soprattutto in questo periodo di pandemia mondiale del virus SARS-CoV2 (COVID-19), l'Intelligenza Artificiale può aiutare le organizzazioni sanitarie a fornire risultati in maniera più semplice per i pazienti, personalizzandone in modo significativo l'assistenza e riducendo i costi.

Il caso d'applicazione trattato risponde proprio a questa esigenza attraverso una prototipo di una Web App in fase di costruzione per l'esame del corso di **Software Architecture Design**.

Abbiamo inizialmente realizzato una piccola applicazione MATLAB in grado di calcolare la probabilità di infezione al SARS-CoV2 (COVID19), inserendo opportunamente i dati. (Figura 19)

The screenshot shows a MATLAB App window titled "MATLAB App" with a sub-header "FORM COVID". The form contains two columns of dropdown menus for inputting symptoms and risk factors. A red dashed box highlights the main input area. To the right of the form, an annotation reads: "Inserimento dei Sintomi e di altre informazioni relative al SARS-CoV2(Covid19) di un paziente." Below the form, another red dashed box highlights the "Predict" button and the "Probability" field, which displays "0.63". An annotation next to this box reads: "Calcolo della Probabilità di Infezione".

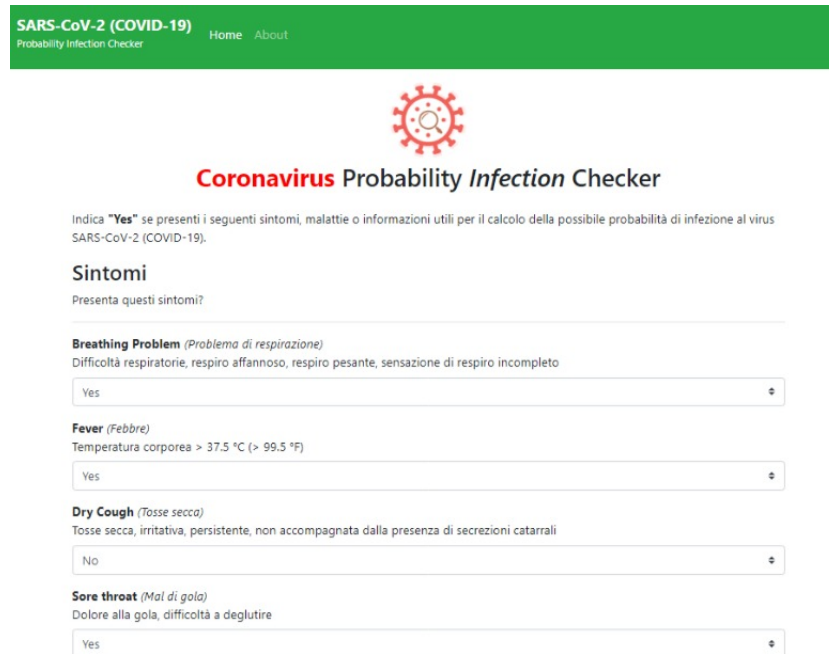
Symptom/Condition	Value
Breathing problem	No
Fever	No
Dry Cough	No
Sore Throat	Si
Running Nose	Si
Asthma	Si
Chronic Lung Disease	No
Headache	Si
Heart Disease	No
Diabetes	Si
HyperTension	Si
Fatigue	No
Gastrointestinal	No
Abroad travel	No
Contact with COVID Patient	No
Attended Large Gathering	Si
Visited Public Exposed Places	No
Family working in Public Exposed Place	No
Wearing Masks	Si
Sanitization from Market	No

Predict Probability 0.63

Figura 19: Inserimento dei sintomi attraverso una applicazione MATLAB

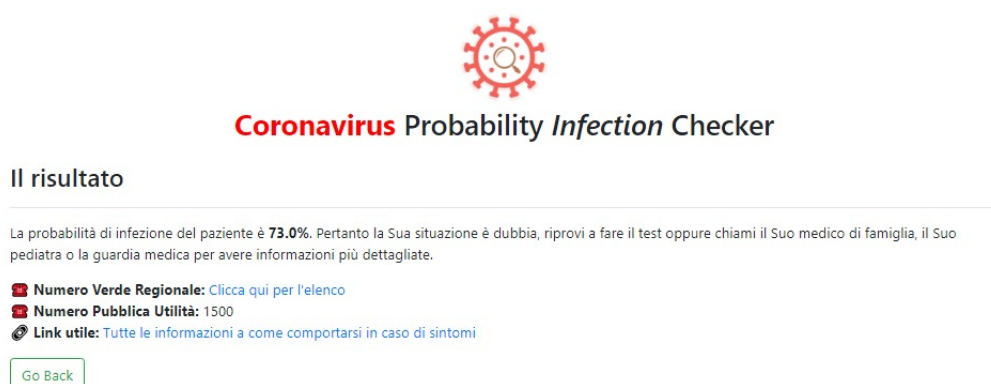
Dopo l'implementazione su MATLAB e uno studio di fattibilità del progetto futuro, si è realizzato un prototipo di Web App for Browser. In figura 20 viene mostrato il form del prototipo:

In breve, i dati inseriti dall'utente verranno inseriti in un form, e attraverso opportuni "script" verranno forniti a un algoritmo di Machine Learning "Logistic Regression" implementato in **Python**, che calcolerà la predizione (figura 21). I parametri inseriti sono: *Breathing Problem*, *Fever*, *Sore Throat*, *Fatigue*, *Contact with COVID Patient*, *Visited Public Exposed Places*, *Family working in Public Exposed Places*, *Wearing Masks*.



The screenshot shows the 'SARS-CoV-2 (COVID-19) Probability Infection Checker' web application. It features a green header with the title and navigation links 'Home' and 'About'. Below the header is a red coronavirus icon. The main title is 'Coronavirus Probability Infection Checker'. A note states: 'Indica "Yes" se presenti i seguenti sintomi, malattie o informazioni utili per il calcolo della possibile probabilità di infezione al virus SARS-CoV-2 (COVID-19)'. The 'Sintomi' section asks 'Presenta questi sintomi?' and contains four input fields: 'Breathing Problem (Problema di respirazione)' with a 'Yes' selection, 'Fever (Febbre)' with a 'Yes' selection, 'Dry Cough (Tosse secca)' with a 'No' selection, and 'Sore throat (Mal di gola)' with a 'Yes' selection.

Figura 20: Inserimento dei sintomi attraverso un classico form HTML



The screenshot shows the result page of the 'Coronavirus Probability Infection Checker'. It features a red coronavirus icon and the title 'Coronavirus Probability Infection Checker'. The section 'Il risultato' displays the result: 'La probabilità di infezione del paziente è 73.0%. Pertanto la Sua situazione è dubbia, riprovi a fare il test oppure chiami il Suo medico di famiglia, il Suo pediatra o la guardia medica per avere informazioni più dettagliate.' Below this, there are three links: 'Numero Verde Regionale: Clicca qui per l'elenco', 'Numero Pubblica Utilità: 1500', and 'Link utile: Tutte le informazioni a come comportarsi in caso di sintomi'. At the bottom is a 'Go Back' button.

Figura 21: Risultato della predizione

10 Sviluppi futuri

Le tecniche tradizionali di Machine Learning richiedono l'estrazione manuale e onerosa di feature dai dati. Inoltre, la scelta delle feature discriminanti da estrarre può essere complessa e limitata a un problema specifico. Il Machine Learning è stato quindi soppiantato dal Deep Learning, che ha impattato numerosi aspetti della società moderna, apportando progressi decisivi in problemi irrisolti nell'intelligenza artificiale. L'idea chiave del Deep Learning è fare in modo che sia un algoritmo di apprendimento a estrarre la migliore rappresentazione dei dati. La disponibilità di enormi quantità di dati, i progressi in hardware, software e computazione parallela hanno reso possibile l'addestramento di reti con milioni di parametri in poche ore. Le reti neurali convoluzionali, in particolare, hanno contribuito allo stato dell'arte nell'ambito della computer vision in attività come classificazione di immagini, segmentazione e rilevamento di oggetti, riuscendo rispetto ad altre architetture a trarre vantaggio dalla correlazione tra pixel adiacenti.

Si tenga a mente però che le performance dei modelli di Deep Learning siano strettamente legate alla presenza di grandi quantità di dati. Laddove il dataset a disposizione sia piccolo o l'hardware piuttosto limitato, i modelli di Machine Learning hanno ancora risultati migliori.

11 Bibliografia e Sitografia

- [1] MODELLI PREDITTIVI *blog.osservatori.net* https://blog.osservatori.net/it_it/modellazione-predittiva-come-funziona
- [2] APPRENDIMENTO SUPERVISIONATO E NON SUPERVISIONATO *CapTerra* <https://www.captterra.it/blog/2523/modelli-machine-learning-supervisionato-o-no>
- [3] OVERFITTING E UNDERFITTING *Andrea Minini* <https://www.andreaminini.com/ai/machine-learning/differenza-tra-overfitting-e-underfitting>
- [4] SUPERVISED LEARNING *Dispensa Andrew Ng*
- [5] DATASET SPESE MEDICALI NEGLI STATI UNITI *Kaggle* <https://www.kaggle.com/datasets/mirichoi0218/insurance>
- [6] DIFFERENZE TRA DISCESA DEL GRADIENTE E NORMAL EQUATIONS *GeeksforGeeks* <https://www.geeksforgeeks.org/difference-between-gradient-descent-and-normal-equation/>
- [7] MATRICE DI CONFUSIONE *www.analyticsvidhya.com* <https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/>
- [8] ML STEPS *www.simplilearn.com* <https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps>