Tesina Signal Processing and
Optimization for Big Data

Implementazione algoritmi di ottimizzazione per la risoluzione del problema di Logistic Regression

Cinti Alessandro **340317**

Indice

Introduzione

Nella seguente tesina sono andato ad implementare tre diversi algoritmi per risolvere il problema di ottimizzazione della Logistic Regression. In particolare è stato implementato il **gradient descent**, **l'algoritmo di Newton** e **ADMM** splittando le osservazioni. Per poter risolvere il problema di ottimizzazione è stato necessario l'utilizzo di un dataset reperibile al seguente link.

Dataset

Il dataset scelto è una collezione di informazioni demografiche e mediche di alcuni pazienti al fine di stimare la loro positività o negatività al diabete. Un singolo campione del dataset presenta le seguenti caratteristiche:

- **gender:** rappresenta il sesso biologico del paziente.
- **age:** età del paziente.
- **hypertension:** condizione medica per cui la pressione del sangue nelle arterie è sempre elevata.
- **heart_disease:** indica se il paziente soffre di malattie cardiache.
- **smoking_history:** descrive se il paziente è un fumatore o lo è stato.
- **bmi:** rapporto tra il peso di una persona e la sua altezza al quadrato
- **HbA1c_level:** media del livello di zuccheri nel sangue negli ultimi 2-3 mesi.
- **blood_glucose_level:** misura del glucosio nel sangue.
- **diabetes:** variabile da stimare. Ha valore 0 quando il paziente non è diabetico, 1 altrimenti.

E' stata necessaria una prima fase di **Exploratory Data Analysis** nella quale sono andato a modificare il dataset in modo da poter essere utilizzato per la stima del parametro. Le colonne **smoking_history** e **gender** sono state trasformate da categoriche a numeriche attraverso una funzione *map.* Il dataset si presenta con soltanto 8.5% dei campioni positivi al diabete, attraverso la funzione *SMOTE* è stato possibile ottenere una sua versione bilanciata. Infine ho diviso il dataset in modo da avere delle osservazioni da utilizzare nella fase di training e altre per la fase di test.

Analisi Teorica

Logistic Regression

La Logistic Regression è utilizzata per risolvere problemi di classificazione in cui la variabile d'uscita è discreta. In questo caso consideriamo il caso binario in cui l'output può assumere i valori 0 e 1.L'obiettivo di questo modello è quello di ricavare la distribuzione a posteriori delle classi condizionata dai dati osservati.
La logistic Regression utilizza come funzione di ipotesi la **sigmoide** la cui caratteristica è quella di mappare i valori di input, solitamente appartenenti ad un range ampio, in un intervallo compreso tra 0 e 1. Di seguito sono definite le probabilità a posteriori delle classi.

$$P(Y = 1|\, x) = h(x) = \frac{1}{1+e^{-w^T x}}$$

$$P(Y = 0|x) = 1 - P(Y = 1|x) = 1 - \frac{1}{1+e^{-w^T x}}$$

A questo punto andiamo a considerare i **log odds** cioè il logaritmo del rapporto tra le probabilità di successo e quelle di insuccesso. Quello che otteniamo è una dipendenza lineare con le features dei dati.

$$Log\ Odds\colon\ lg\left(\frac{P(Y=1|x)}{P(Y=0|x)}\right) = w^T x$$

Questa espressione mi è utile per ricavare i valori ottimali del vettore *w* in modo da poter fare predizioni andando a confrontare la sigmoide con una soglia.

$$y = sigmoid(w^T x) \geq \gamma$$

La soglia, indicata con γ, ha un ruolo importante poiché va a determinare la probabilità di classificare correttamente un osservazione ma anche la probabilità di classificazione non corretta (Falso Allarme). In questo caso vado ad utilizzare una soglia pari a 0.5 in modo da non assegnare un peso maggiore a nessuna delle decisioni.
Per stimare i parametri *w* posso utilizzare un approccio *Maximum Likelihood Estimator*. Assumendo di avere osservazioni indipendenti e identicamente distribuite vado ad impostare il seguente problema di ottimizzazione:

$$\max_{\underline{w}} f_{XY}(X, Y, \underline{w}) = \max_{\underline{w}} \prod_{i=1}^{N} (P(y_i = 1|\underline{x_i}))^{y_i} (1 - P(y_i = 1|\underline{x_i}))^{1-y_i}$$

Invece di massimizzare questa funzione, posso semplificare la risoluzione del problema andando a minimizzare la *Log Likelihood Negativa.* Si ottiene il seguente problema di ottimizzazione:

$$\min_{\underline{w}} -\frac{1}{m}\sum_{i=1}^{N}[y_i \log\Big(P(y_i=1|\underline{x_i})\Big) + (1-y_i)\log\Big(1-P(y_i=1|\underline{x_i})\Big)]$$

Il problema di ottimizzazione della Logistic Regression non ha soluzione in forma chiusa. E' possibile utilizzare algoritmi iterativi poiché il problema è convesso e non ha termini di regolarizzazione, quindi possiamo trovare il minimo globale.

Algoritmi di Ottimizzazione

Gradient Descent

Si tratta di un algoritmo iterativo che permette la minimizzazione della funzione costo. L'espressione iterativa è la seguente:

$$\underline{w}^{k+1} = \underline{w}^{k} - \mu \bigtriangledown J(\underline{w}^{k})$$

Il gradiente della funzione costo è utilizzato per stabilire la direzione in cui ci spostiamo nella ricerca del minimo mentre lo step size μ stabilisce la grandezza dello spostamento.
Per calcolare il gradiente della funzione costo è necessario conoscere la derivata della sigmoide che è uguale a $sigmoide'(z) = sigmoide(z) * (1 - sigmoide(z))$.
Il gradiente è il seguente.

$$\bigtriangledown_{\underline{w}} J(\underline{X}, \underline{w}) = \frac{1}{m}\sum_{i=1}^{N} \underline{x}_i(y_i - sigmoide(\underline{w}^T \underline{x}_i))$$

Sostituiamo il gradiente della funzione costo all'aggiornamento iterativo del gradient descent

$$\underline{w}^{k+1} = \underline{w}^{k} - \mu X^T(\underline{y} - sigmoide(\underline{w}^k X)))\frac{1}{m}$$

E' stato impostato un numero massimo di iterazioni pari a 1000 e l'aggiornamento dei pesi verrà eseguito finché questo limite non viene raggiunto. Un altra condizione per cui l'algoritmo si arresta la si ha quando il valore della funzione costo differisce dal valore precedente per un valore sufficientemente piccolo. La condizione è la seguente.

$$|J(X, \underline{w}^{k+1}) - J(X, \underline{w}^{k})| < \varepsilon$$

Algoritmo di Newton

L'algoritmo di Newton sfrutta sia l'informazione della derivata prima che l'informazione della derivata seconda. Questo algoritmo ha una velocità di convergenza maggiore rispetto a quella del gradient descent ma è computazionalmente oneroso poiché è necessario

calcolare l'inversa della matrice Hessiana. L'aggiornamento iterativo è dato dalla seguente espressione.

$$\underline{w}^{k+1} = \underline{w}^{k} - \mu(H(J(X, w^{k})))^{-1} \bigtriangledown_{\underline{w}} J(X, w^{k})$$

Come è stato già detto, la complessità di questo algoritmo deriva dal calcolare l'inversa della matrice Hessiana. Questa matrice la possiamo ricavare utilizzando la seguente espressione $H(J(X,w)) = X^{T}PX$ in cui la matrice P è una matrice diagonale fatta in questo modo.

$$P = \begin{bmatrix} s(x_1 w)(1 - s(x_1 w)) & & \\ & \cdots & \\ & & s(x_n w)(1 - s(x_n w)) \end{bmatrix}$$

A questo punto siamo in grado di scrivere l'espressione di aggiornamento dei pesi

$$\underline{w}^{k+1} = \underline{w}^{k} - \mu(X^{T}PX)^{-1}(X^{T}(\underline{y} - sigmoid(\underline{w}^{k}\underline{X}))\frac{1}{m})$$

Anche in questo caso il numero massimo di iterazioni è stato impostato a 500 e resta presente la condizione di arresto.
NOTA: A causa delle scarse prestazioni del mio pc sono stato costretto ad utilizzare soltanto una parte dei dati che avevo a disposizione.

ADMM distribuito rispetto ai dati

In generale l'algoritmo ADMM è un algoritmo che va a sfruttare sia l'informazione primale che l'informazione duale del problema di ottimizzazione.
Se ci troviamo nel caso in cui la funzione costo può essere espressa come somma di funzioni costo indipendenti, allora siamo nella situazione in cui è possibile dividere il calcolo su più agenti. Il problema di ottimizzazione deve essere espresso utilizzando una variabile di supporto *z* che, tramite un vincolo, è legata alla variabile di ottimizzazione principale.
Quanto detto lo possiamo esprimere con il seguente problema di ottimizzazione:

$$min_{\underline{w_1}...\underline{w_n},\underline{z}} \sum_{i=1}^{N} f_i(\underline{w_i})$$

$$st. \quad \underline{w_i} = \underline{z} \quad \forall i = 1, \ldots, N$$

La funzione costo della Logistic Regression può essere scritta come somma di funzioni costo rispetto ad uno o più campioni. Quindi possiamo formulare il problema di ottimizzazione della Logistic Regression utilizzando l'algoritmo ADMM.
La funzione costo è simile a quella definita precedentemente con la differenza che qui non andiamo a considerare tutte le osservazioni ma ogni agente, su cui distribuiamo il calcolo, ha una porzione dei dati. A questo punto andiamo a definire i tre aggiornamenti che caratterizzano l'algoritmo ADMM.

$$\underline{w}_i^{k+1} = min_{w_i}\{-\frac{1}{m}(y_i(sigmoid(X^j\underline{w})) + (1-\underline{y_i})(1-sigmoid(X^j\underline{w}))) + \rho||w_i - z^k + u_i^k||_2^2\}$$

Questo primo step è l'unico che ogni agente può calcolare localmente, il calcolo prevede la conoscenza delle variabili di ottimizzazione aggiornate al passo precedente e la propria porzione di dati. Questo step è un vero e proprio problema di ottimizzazione che per semplicità ho risolto utilizzando il gradient descent. E' importante notare che questo problema di ottimizzazione viene risolto dagli agenti ad ogni iterazione.

$$\underline{z}^{k+1} = \frac{1}{N}(\sum_{i=1}^{N}\underline{w}_i^{k+1} + \sum_{i=1}^{N}\underline{u}_i^k)$$

Nel secondo passo vado ad aggiornare la variabile globale z. In questo caso ho bisogno di tutte le soluzioni calcolate dagli agenti allo step precedente e quindi non posso distribuirlo. Infatti è necessario l'utilizzo di un Fusion Center oppure l'utilizzo di algoritmi al consenso. Nella pratica quello che andiamo a fare per aggiornare la variabile globale è una media delle variabili locali e duali.

$$\underline{u}^{k+1} = \underline{u}^k + (\underline{w}^{k+1} - \underline{z}^{k+1})$$

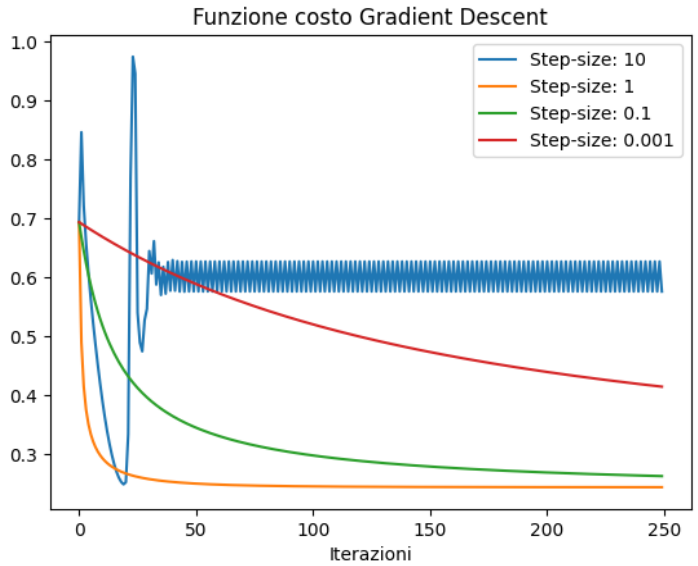
Nell'ultimo step quello che andiamo a fare è aggiornare la variabile duale. Utilizziamo il gradient ascent poiché il minimo del problema primale, se sono rispettate certe condizioni, coincide con il massimo del problema in forma duale.

Risultati

A questo punto andiamo ad osservare l'andamento della funzione costo in base ai diversi algoritmi di ottimizzazione utilizzati

Gradient Descent

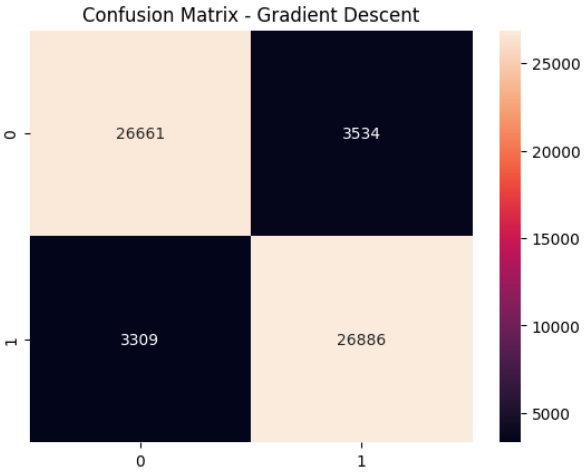
La scelta dello step-size va a condizionare la velocità con cui la funzione costo raggiunge il minimo. Uno step-size troppo piccolo comporta un aggiornamento ridotto dei pesi e di conseguenza la funzione costo si avvicinerà lentamente verso il minimo, in questo caso sarà necessario effettuare un alto numero di iterazioni per convergere.
Al contrario se utilizziamo un step-size molto grande la funzione costo potrebbe rimbalzare all'infinito senza mai arrivare al minimo.



Queste situazioni le possiamo osservare nella figura in cui con uno step-size pari a 50 la funzione costo non converge, mentre con uno step-size uguale a 0.01 la funzione costo converge ma molto lentamente. Infatti quando ho utilizzato lo step-size più piccolo il numero di iterazioni prefissato non è stato sufficiente a raggiungere il minimo.
Soltanto con lo step-size pari ad 1, l'aggiornamento della funzione costo è minore di $\epsilon = 10^{-7}$ e quindi l'algoritmo si arresta..

Prestazioni Classificatore

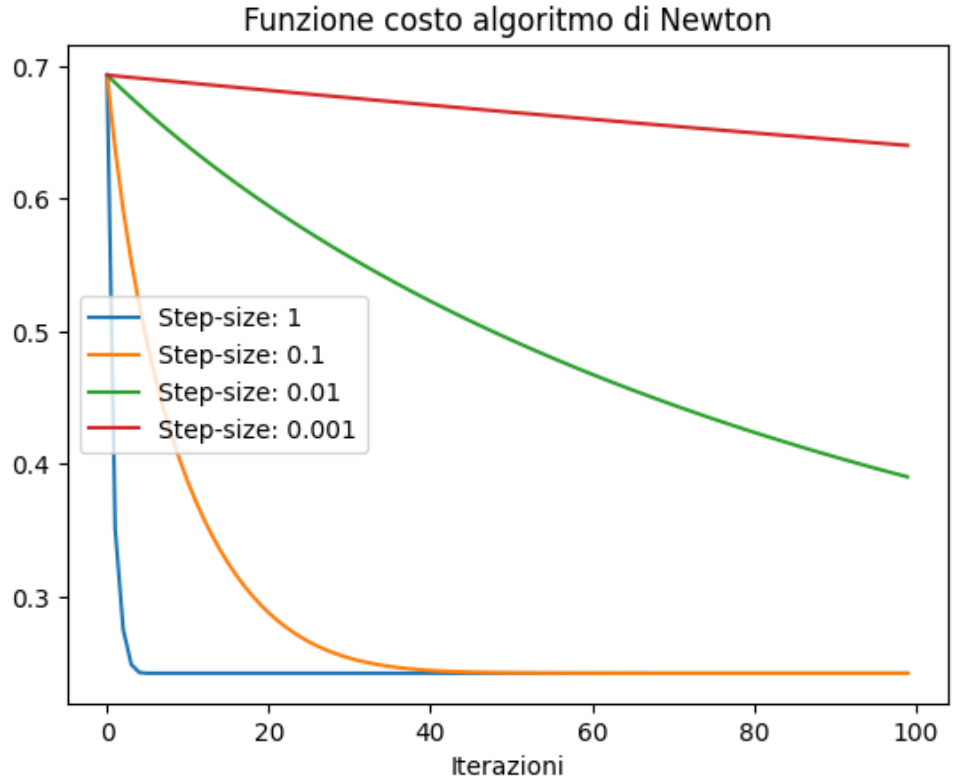
	Train Set	Test Set



Accuracy	0.886591632003914 9	0.887745170815394
F1 Score	0.886906165870956 6	0.8873157807584037

Algoritmo di Newton

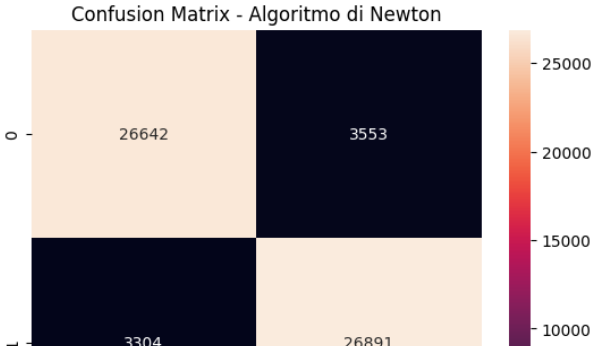
L'algoritmo di Newton sfrutta più informazioni per minimizzare la funzione per cui necessita di meno iterazioni per convergere al minimo. Il calcolo dell'inversa dell' Hessiana lo rende computazionalmente oneroso per cui la sua velocità potrebbe scontrarsi con le limitazioni dell' hardware.



In questo caso notiamo come con uno step-size pari ad 1 la funzione costo raggiunge il suo valore minimo con meno passi rispetto al gradient descent a parità di step-size. Con un numero di iterazioni pari a 500 e step-size pari a 0.001, la funzione costo non riesce a raggiungere il minimo.

Prestazioni Classificatore

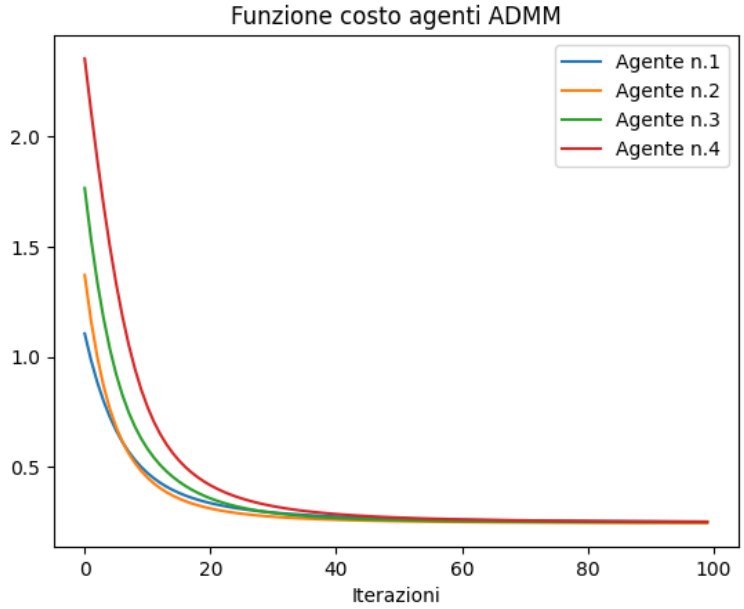
	Train Set	Test Set



Accuracy	0.885580295245086 1	0.8869209584590776
F1 Score	0.885875356919147 2	0.8864547110448749

ADMM

L'algoritmo ADMM implementato prevede l'utilizzo di 4 agenti che vanno a risolvere localmente il problema di ottimizzazione rispetto ad una porzione dei dati. Ogni agente utilizza il gradient descent per minimizzare la funzione costo e nel seguente grafico è possibile andare a visualizzare il valore della funzione costo per ogni agente.

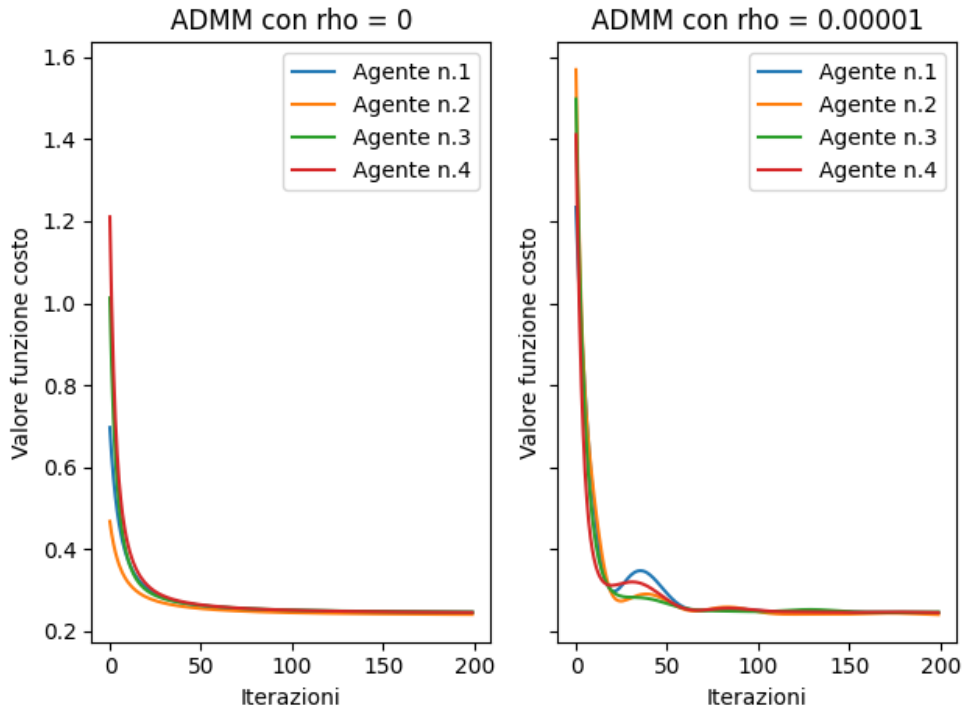


Ogni agente utilizza una porzione diversa di dati e quindi avremo valori di partenza della funzione costo differenti tra i vari agenti. Una volta che l'algoritmo ADMM ha terminato le iterazioni possiamo andare a verificare i pesi calcolati da ciascun agente che dovrebbero essere simili tra loro.

	Agente 1	**Agente 2**	**Agente 3**	**Agente 4**	**G.D.**	**Newton**
0	-0.024071	0.078133	0.029441	0.030469	0.030467	0.023768
1	1.280812	1.274694	1.260967	1.217120	1.184898	1.215563
2	-0.039966	-0.055657	-0.175129	-0.017504	-0.068694	-0.044202

3	-0.061522	-0.069261	-0.096107	-0.051955	-0.063460	-0.061312
4	0.727563	0.740028	0.733764	0.756791	0.699669	0.697914
5	2.990195	3.110417	3.072954	3.123934	2.594809	2.843252
6	1.731311	1.823538	1.747078	1.779353	1.650552	1.680628
7	-0.248602	-0.214496	-0.182054	-0.239818	-0.210712	-0.222244
8	-0.084477	-0.083877	-0.097232	-0.075667	-0.078349	-0.076213

In questo algoritmo abbiamo il parametro ρ necessario per la creazione del Lagrangiano Aumentato, il cui ruolo è quello di aumentare la convessità della funzione da minimizzare. La funzione costo della Logistic Regression è una funzione strettamente convessa, quando ρ è diverso da zero ottengo un peggioramento delle prestazioni quindi l'algoritmo ADMM va a risolvere il problema di ottimizzazione con $\rho = 0$.



Prestazioni Classificatore

	Train Set	Test Set
Accuracy	0.885099094690482	0.8862560026494453
F1 Score	0.8853515625	0.8866782149632929