Project Work in Optimization Techniques for Machine Learning
Almost multisecant quasi-Newton method

Gregorio Piqué
gregorio.pique@edu.unifi.it

April 9, 2024

1 Introduzione

Il seguente documento costituisce il resoconto relativo al project work del corso di *Optimization Techniques for Machine Learning* del curriculum di Laurea Magistrale in Intelligenza Artificiale presso l'Università degli Studi di Firenze. Il lavoro ha previsto lo studio di un paper scientifico e la riproduzione dei suoi risultati mediante esperimenti su delle istanze di problemi di regressione lineare e regressione logistica regolarizzata.

L'articolo preso in considerazione [1] propone un approccio alternativo per la risoluzione di problemi di ottimizzazione non vincolata mediante metodi Quasi-Newton. Questi metodi si distinguono per l'utilizzo di una matrice che approssima l'hessiana della funzione obiettivo per individuare la direzione di discesa. Tale matrice viene aggiornata ad ogni iterazione in modo da soddisfare la cosiddetta equazione Quasi-Newton.

2 Metodi Quasi-Newton

I metodi Quasi-Newton (QN) sono un particolare tipo di algoritmi di ottimizzazione che utilizzano l'informazione del primo ordine della funzione obbiettivo, ovvero il gradiente, e un'approssimazione di quella del secondo ordine, cioè l'hessiana. Questi algoritmi imitano il funzionamento del metodo di Newton (del secondo ordine) per cercare di sfruttarne le buone proprietà di convergenza. Nei metodi di Newton e QN, la direzione di discesa è ottenuta combinando l'inversa dell'hessiana (o della sua approssimazione) con l'antigradiente, ovvero:

$$d_k = -B_k^{-1}\nabla f(x^k), \quad B_k \approx \nabla^2 f(x^k) \tag{1}$$

La matrice B_k deve essere tale da soddisfare l'equazione Quasi-Newton, per la quale

$$B_k s_k = y_k \tag{2}$$

con $s_k = x^k - x^{k-1}$ e $y_k = \nabla f(x^k) - \nabla f(x^{k-1})$, $x^k \in \mathbb{R}^n$. La matrice B_k viene aggiornata ad ogni iterazione dell'algoritmo, con formule di aggiornamento che dipendono da s_k, y_k e da B_k stessa.

2.1 Multisecant Quasi-Newton

Una possibile estensione dei metodi QN è costituita dagli algoritmi *Multisecant Quasi-Newton* (*MSQN*), che sfruttano una maggiore quantità di informazioni (in termini di gradienti e soluzioni delle precedenti p iterazioni) con l'obbiettivo di ottenere una più accurata approssimazione dell'hessiana. In questo caso, l'aggiornamento di B_k avviene utilizzando le nuove matrici S_k e Y_k di dimensione $n \times p$, dove

$$S_k = \begin{bmatrix} | & | & & | \\ s_{k-p} & s_{k-p+1} & \dots & s_k \\ | & | & & | \end{bmatrix}, \qquad Y_k = \begin{bmatrix} | & | & & | \\ y_{k-p} & y_{k-p+1} & \dots & y_k \\ | & | & & | \end{bmatrix}$$

La formula esatta di aggiornamento della matrice B_k per algoritmi MSQN è

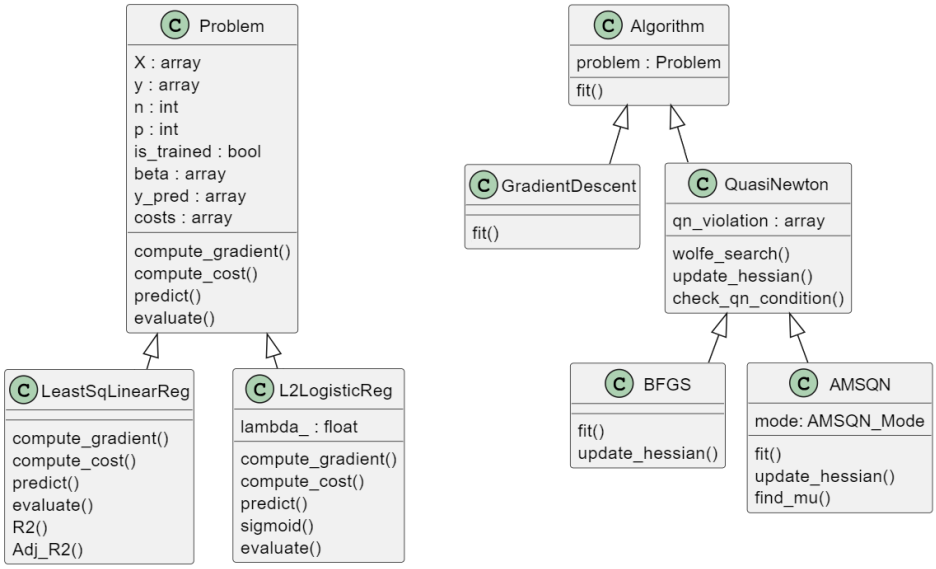

Figure 1: Classi per problemi e algoritmi.

$$
\begin{aligned}
B_{k+1} &= B_k - D_1 W^{-1} D_2^T \\
&= B_k + Y_k (Y_k^T S_k)^{-1} Y_k^T - B_k S_k (S_k^T B_k S_k)^{-1} S_k^T B_k
\end{aligned} \tag{3}
$$

In generale però, i metodi MSQN non garantiscono che la direzione ottenuta con l'equazione (1) sia di discesa [1]. Per questo motivo, viene proposto in [1] l'algoritmo *Almost Multisecant Quasi-Newton* (*AMSQN*), un ulteriore modifica al metodo MSQN, che impone che B_k sia (i) simmetrica e (ii) definita positiva, condizioni che permettono di ottenere una direzione di discesa, al potenziale costo di non riuscire a soddisfare l'equazione QN (2). Le perturbazioni proposte dal metodo AMSQN in [1] vengono applicate durante l'aggiornamento della matrice, come segue:

$$
\begin{aligned}
B_{k+1} &= B_k + \overline{\Delta} + \mu I \\
\overline{\Delta} &= -\frac{D_1 W^{-1} D_2^T + (D_1 W^{-1} D_2^T)^T}{2} \\
\mu &= max\{0, -\lambda_{min}(\overline{\Delta})\}
\end{aligned}
$$

Più nel dettaglio, $\overline{\Delta}$ rappresenta la perturbazione per forzare la simmetria di B_{k+1}, mentre μ è il più piccolo valore necessario affinché $B_{k+1} \succeq 0$ [1].

3 Implementazione

Il progetto ha richiesto l'implementazione di classi e metodi Python per la risoluzione di problemi di regressione lineare e di regressione logistica regolarizzata (ℓ_2) con diversi algoritmi, quali il metodo del gradiente, BFGS, e AMSQN in quattro varianti: (i) usando la formula esatta (3), (ii) applicando solo la perturbazione per imporre B_{k+1} simmetrica, (iii) solo la perturbazione per rendere B_{k+1} definita positiva, (iv) usando entrambe le perturbazioni. Le classi implementate e le relazioni tra queste sono mostrate nella Figura 1. Nella Appendice A è riportanto un caso d'uso del progetto implementato. Il codice sorgente è accessibile al repository *Git* `https://github.com/Pikerozzo/almost-multi-secant-quasi-newton`.

3.1 Problemi

I problemi di regressione lineare e logistica sono rappresentati dalle classi *LeastSqLinearReg* e *L2LogisticReg*, le quali ereditano dalla classe astratta *Problem*: questa definisce, tra gli altri, i metodi per il calcolo del

gradiente (*compute_gradient()*), del valore della funzione di costo (*compute_cost()*) e per fare predizioni usando una certa soluzione specificata rappresentata dal vettore dei pesi β (*predict()*); questi metodi vengono tutti implementati nelle due sottoclassi.

3.1.1 Regressione Lineare

Il problema di regressione lineare ai minimi quadrati ha funzione obbiettivo e gradiente come segue:

$$\min_{\beta \in \mathbb{R}^p} f(\beta) = \min_{\beta \in \mathbb{R}^p} \frac{||X\beta - y||^2}{2n}$$

$$\nabla f(\beta) = \frac{X(X\beta - y)}{n}$$

Le predizioni $\hat{y}$ (su un dataset X usando la soluzione β specificata) sono ottenibili come segue:

$$\hat{y} = X\beta$$

3.1.2 Regressione Logistica

Il problema di regressione logistica regolarizzata (ℓ_2-*regularized logistic regression*) identifica un problema di classificazione (in questo caso) binaria, dove la variabile esplicativa $x^{(i)}$ ha come target $y^{(i)}$, la quale è categorica e può assumere valore 0 o 1. La funzione obbiettivo e il gradiente assumono la forma seguente:

$$\min_{\beta \in \mathbb{R}^p} f(\beta) = \min_{\beta \in \mathbb{R}^p} -\frac{1}{n}[\sum_{i=1}^{n} y \log(\sigma(X\beta)) + (1-y)\log(1-\sigma(X\beta))] + \frac{\lambda}{2n}\sum_{j=1}^{p}\beta_j^2$$

$$\nabla f(\beta) = \frac{X^T(\sigma(X\beta) - y) + \lambda\beta}{n}$$

dove il coefficiente λ rappresenta il parametro di regolarizzazione e $\sigma(z)$ è una funzione non lineare, detta sigmoide:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Le predizioni $\hat{y}^{(i)}$ (su un dato $x^{(i)}$ usando la soluzione β specificata) sono ottenibili come segue:

$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } \sigma(x^{(i)}\beta) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 1 Main structure of `fit` method

Input: β^0, $\alpha_0, max\_iterations, tolerance$

for $i \leftarrow 1$ **to** $max\_iterations$ **do**
  if $\nabla f(\beta^k) \leq tolerance$ **then**
    break
  end
  Get direction d_k
  Get step α_k
  Update coefficients: $\beta^{k+1} \leftarrow \beta^k + \alpha_k \cdot d_k$
end

Output: Trained model with best coefficients β

3.2 Algoritmi

I problemi citati sono risolvibili con tre algoritmi diversi, attraverso il metodo *fit()* fornito per la loro risoluzione. La struttura di questo metodo è simile per i tre algoritmi (si veda l'algoritmo 1); ognuno di essi usa poi approcci diversi per la scelta della direzione e del passo α da effettuare ad ogni iterazione. Gli algoritmi implementati sono *gradient descent* (3.2.1), *BFGS* (3.2.2) e *AMSQN* (3.2.3) nelle sue quattro varianti.

3.2.1 Gradient Descent

Il metodo del gradiente è un algoritmo del primo ordine ed è il più semplice tra quelli implementati: effettua spostamenti con passo costante $\alpha_k = \alpha_0 \quad \forall k$, lungo la direzione dell'antigradiente $(-\nabla f(\beta^k))$ che per definizione è quella di massima discesa.

3.2.2 BFGS

Il secondo algoritmo implementato è BFGS, metodo QN che calcola la direzione usando la matrice H_k, approssimazione dell'inversa dell'hessiana. Il passo è scelto con il metodo di ricerca di linea di tipo *Wolfe* (debole o forte)[2], il quale consente di individuare un passo α_k tale da fornire un sufficiente decremento, in modo che la pendenza della curva nel nuovo punto si sia ridotta a sufficienza. L'aggiornamento di H_k, effettuato usando le componenti s_k e y_k (si veda la Sezione 2), avviene con le formule BFGS inverse.

$$d_k = -H_k \nabla f(\beta^k) \quad \forall k$$

$$H_{k+1} = H_k + \frac{(s_k^T y_k + y_k^T H_k y_k)(s_k s_k^T)}{(s_k^T y_k)^2} - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{(s_k^T y_k)}$$

3.2.3 AMSQN

L'ultimo algoritmo implementato è AMSQN nelle sue quattro varianti. Le prime tre utilizzano un passo costante, mentre l'ultima effettua, come BFGS, una ricerca di Wolfe. La direzione di discesa è determinata utilizzando l'inversa della matrice B_k, aggiornata secondo le formule previste dalla variante AMSQN utilizzata:

i formula esatta (3): $B_{k+1} = B_k - D_1 W^{-1} D_2^T$

ii B_{k+1} simmetrica: $B_{k+1} = B_k - \frac{D_1 W^{-1} D_2^T + (D_1 W^{-1} D_2^T)^T}{2}$

iii B_{k+1} definita positiva: $B_{k+1} = B_k - D_1 W^{-1} D_2^T + \mu I$

iv B_{k+1} simmetrica definita positiva: $B_{k+1} = B_k - \frac{D_1 W^{-1} D_2^T + (D_1 W^{-1} D_2^T)^T}{2} + \mu I$

Il valore μ è ottenuto calcolando gli autovalori della matrice $\overline{\Delta}$ utilizzata nell'aggiornamento, operazione effettuata usando le funzionalità della libreria Python `numpy`.

4 Risultati

In questa sezione vengono esaminati e commentati i risultati ottenuti. Vengono presentate diverse metriche di valutazione, tra cui il valore della funzione di costo, la violazione dell'equazione Quasi-Newton (2) per i metodi di tipo QN, e il tempo di esecuzione per una singola iterazione. I test sono stati effettuati con i diversi algoritmi su due istanze di regressione lineare ai minimi quadrati [3, 4] e regressione logistica regolarizzata [5, 6]. Le figure seguenti mostrano, in particolare, i risultati ottenuti con [3, 5].

I grafici nelle Figure 2 e 5 illustrano l'andamento del valore della funzione di costo per il numero di iterazioni effettuate (sotto-grafici a sinistra) e la quantificazione della violazione dell'equazione QN (sotto-grafici a destra). Per entrambe le tipologie di problema, le migliori performance in termini di errore della funzione obbiettivo sono ottenute con i metodi BFGS e *AMSQN - BOTH* (entrambe le perturbazioni), i quali mostrano una rapida convergenza in termini di iterazioni. Anche il metodo del gradiente converge al minimo trovato dai QN, presentando tuttavia un tasso di convergenza significativamente più lento.

Le figure 3 e 6 illustrano il tempo di esecuzione per singola iterazione, rispettivamente per la regressione lineare e quella logistica. Emerge chiaramente che la variante *AMSQN - BOTH* richiede i tempi di esecuzione maggiori per entrambi i problemi. Tra i possibili motivi, vi è il fatto che le varianti AMSQN che applicano la modifica per forzare B_k a essere definita positiva calcolano autovalori e autovettori di una matrice $n \times n$, un'operazione relativamente costosa, generalmente dell'ordine di $O(n^3)$. Inoltre, tutte le varianti dell'algoritmo AMSQN utilizzano l'approssimazione dell'hessiana B_k, e ciò richiede di invertire la matrice (altra operazione di complessità $O(n^3)$). Questo passaggio non è per

esempio necessario nel BFGS, poiché questo algoritmo usa direttamente H_k, ovvero l'approssimazione dell'inversa dell'hessiana. Infine, sia il BFGS che la variante *AMSQN - BOTH* utilizzano una ricerca di Wolfe per identificare il passo da effettuare ad ogni iterazione, un passaggio che richiede ulteriore tempo per ogni iterazione rispetto all'utilizzo di un passo costante.

Le Figure 4 e 7 mostrano infine l'andamento del valore della funzione di costo in relazione al tempo di esecuzione. Date le considerazioni fatte in precedenza, emerge chiaramente come i metodi QN (BFGS e AMSQN - BOTH) conducano in poche iterazioni a un maggiore miglioramento. Tuttavia, tali iterazioni sono più onerose e costose rispetto ad altri metodi (come quello del gradiente), risultando in tempi di esecuzione più elevati. Inoltre, tra i due metodi QN, il AMSQN risulta essere più lento per entrambi i problemi, dovuto anche a passaggi dell'algoritmo caratterizzati da un'elevata complessità computazionale; questi includono l'inversione di una matrice $n \times n$ e il calcolo di autovalori e autovettori, entrambe operazioni con complessità $O(n^3)$.

5 Conclusioni

Questo progetto ha richiesto di studiare [1] e di replicarne il metodo proposto; questo rappresenta un approccio alternativo ai metodi Multisecant Quasi-Newton che si avvale di aggiornamenti ad-hoc per rendere la matrice utilizzata simmetrica e definita positiva. Nonostante appaia, per i vari motivi citati, più computazionalmente pesante rispetto ad altri metodi, l'algoritmo AMSQN proposto e implementato permette di ottenere buoni risultati in poche iterazioni, evidenziando la rapida convergenza di questa tipologia di algoritmo.

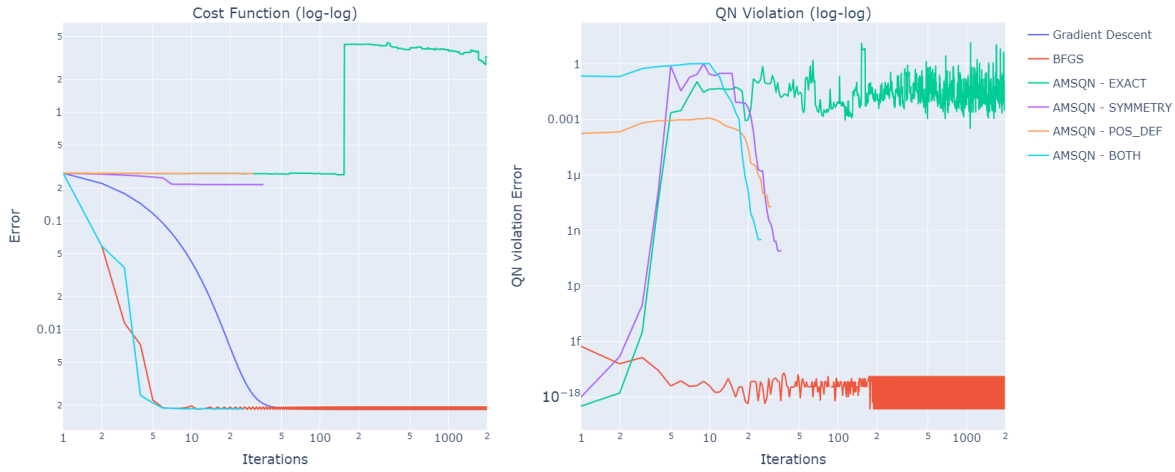

Figure 2: Risultati test regressione lineare.

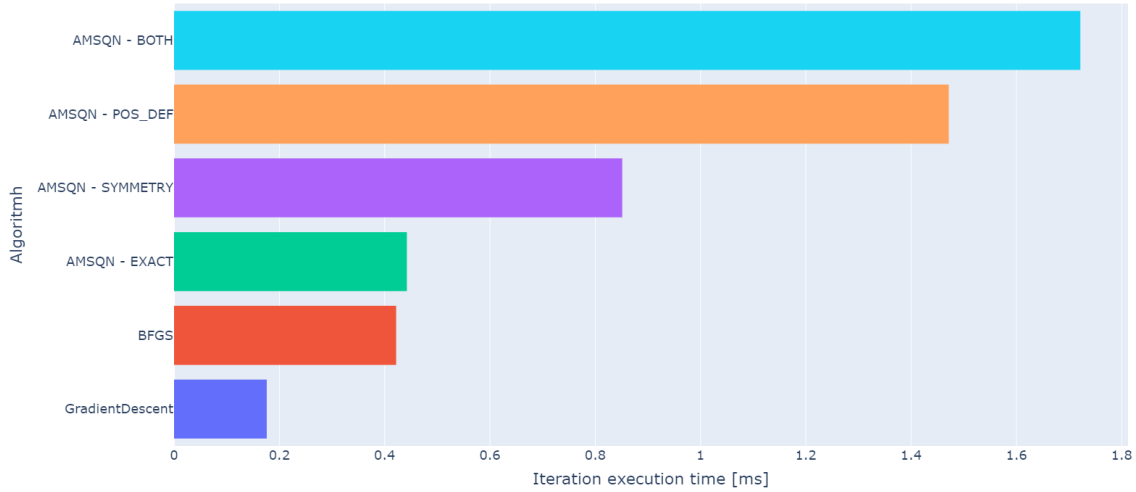

Figure 3: Tempi di esecuzione medi per iterazione, caso regressione lineare.

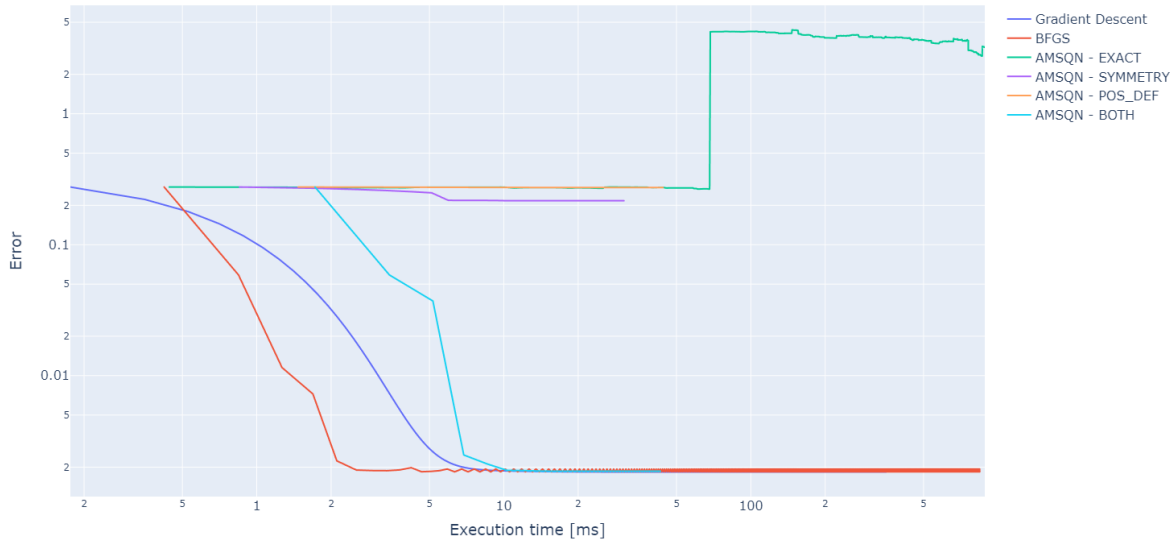

Figure 4: Errore per tempo di esecuzione, caso regressione lineare.

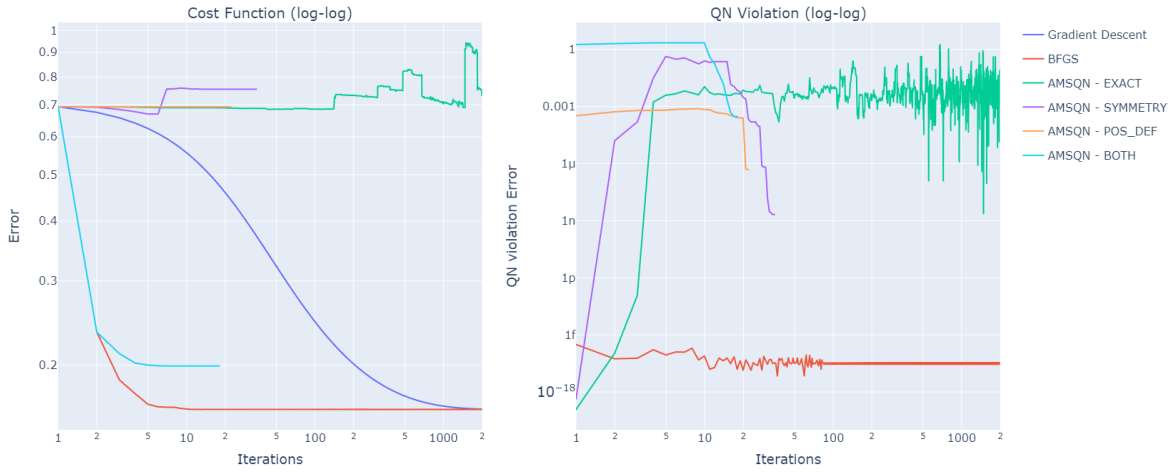

Figure 5: Risultati test regressione logistica.

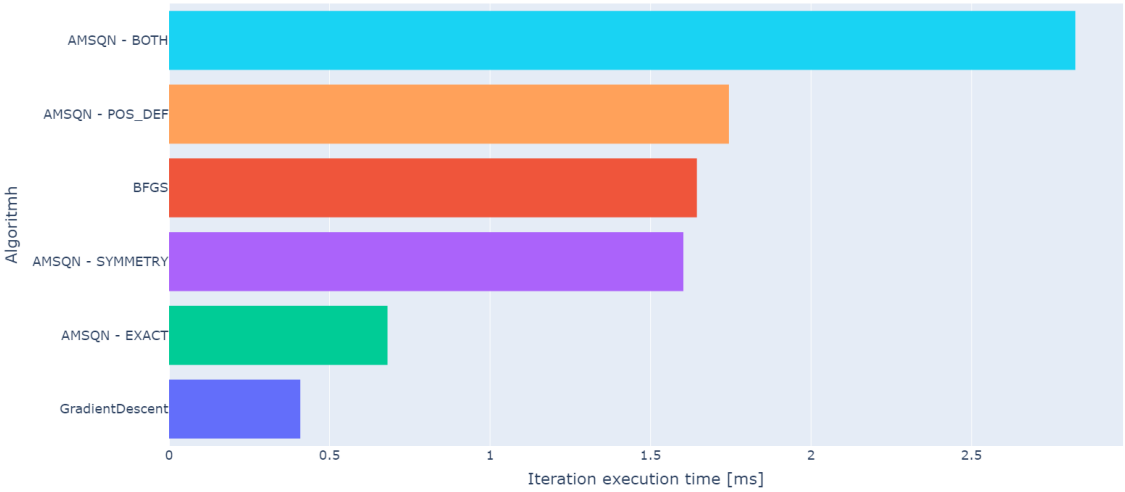

Figure 6: Tempi di esecuzione medi per iterazione, caso regressione logistica.

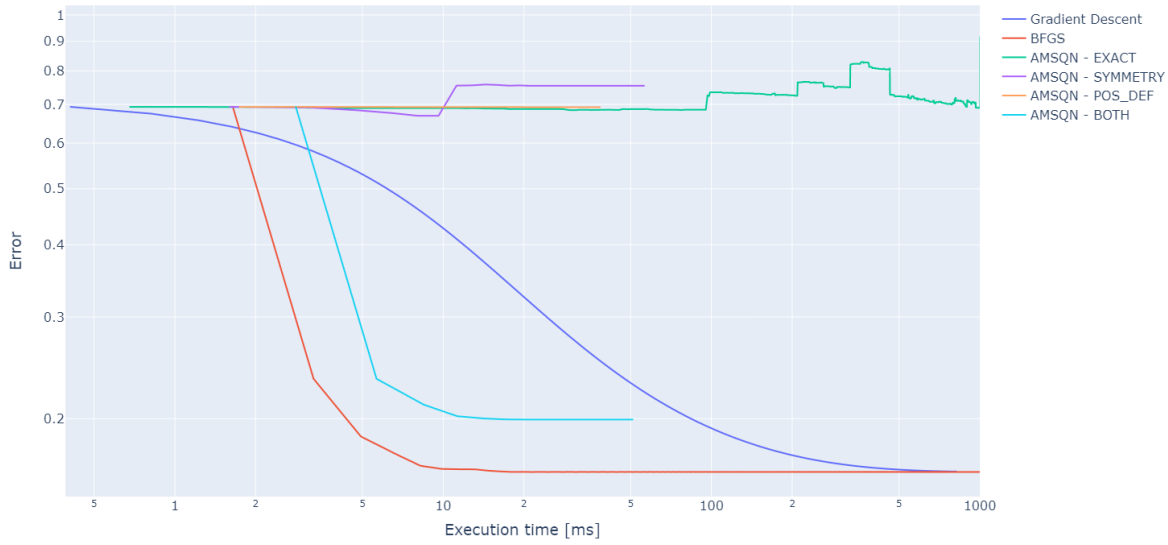

Figure 7: Errore per tempo di esecuzione, caso regressione logistica.

A Caso d'Uso

In questa sezione vengono mostrate le modalità di utilizzo delle funzionalità implementate; l'esempio che segue è relativo alla risoluzione di un problema di regressione logistica [5] usando *AMSQN - BOTH*, variante con entrambe le perturbazioni.

```
from amsqn import *

# Load the local dataset (source: https://www.kaggle.com/mirichoi0218/insurance)
insurance_df = pd.read_csv('..\\data\\insurance.csv')

# Convert the 'smoker' column to a binary column
insurance_df['smokes'] = (insurance_df['smoker']=='yes').astype(int)

# Split the dataset into features and target variable
X, y = insurance_df[['bmi', 'charges']].values, insurance_df['smokes'].values

# Split the dataset into training and testing sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1492)

# Scale the features
sc = StandardScaler()
sc = sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# Create the problem instance
problem = L2LogisticReg(X_train, y_train)

# Create the algorithm instance and train the model
alg = AMSQN(problem=problem, mode=AMSQN_Mode.BOTH)
alg.fit(learning_rate=0, max_iterations=1000)

# Evaluate the results on the test set
alg.problem.evaluate(X=X_test, y_true=y_test)
```

References

[1] M. Lee and Y. Sun. "Almost multisecant BFGS quasi-Newton method". In: *OPT 2023: Optimization for Machine Learning*. 2023. URL: https://openreview.net/forum?id=XW56zLleWK.

[2] L. Grippo and M. Sciandrone. *Introduction to Methods for Nonlinear Optimization*. Springer International Publishing, 2023. ISBN: 9783031267901.

[3] *Kaggle - Graduate Admissions 2*. URL: https://www.kaggle.com/datasets/mohansacharya/graduate-admissions. (accessed: 17.03.2024).

[4] *sklearn.datasets - California Housing*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html. (accessed: 15.03.2024).

[5] *Kaggle - Medical Cost Personal Datasets*. URL: https://www.kaggle.com/mirichoi0218/insurance. (accessed: 20.03.2024).

[6] *Kaggle - Diabetes Dataset*. URL: https://www.kaggle.com/datasets/ehababoelnaga/diabetes-dataset. (accessed: 16.03.2024).