Signal Processing and Optimization for Big-Data

Presentazione Progetto

Corso di Laurea Magistrale in

Ingegneria Informatica e Robotica

curriculum Data Science

Anno Accademico 2022/2023

ADDESTRAMENTO DI UN MODELLO DI REGRESSIONE LOGISTICA MEDIANTE DIFFERENTI ALGORITMI DI OTTIMIZZAZIONE

Professore: Paolo Banelli

Studente: Simone Maiorani

Matricola: 346627

A.D. 1308

unipg

UNIVERSITÀ DEGLI STUDI
DI PERUGIA

INDICE

1) Introduzione

Il focus di questo progetto riguarda l'implementazione di tecniche algoritmiche mirate alla risoluzione di problemi di ottimizzazione. In esame è stato preso l'addestramento di Logistic Regression, un modello di machine learning volto a classificazioni di tipo binarie date delle osservazioni in ingresso.

1.1) Dataset

Il dataset utilizzato per la realizzazione del progetto è reperibile al sito https://www.kaggle.com/datasets/adityakadiwal/water-potability/data e prevede la classificazione di acqua in potabile o non potabile.
Nello specifico, il dataset è composto da 3276 campioni, ciascuno dei quali avente 10 features di cui 9 numeriche e 1 categorica (target da predire):

- **pH**: valore del pH;
- **Hardness**: durezza dell'acqua;
- **Solids**: solidi totali (ppm);
- **Chloramines**: clorammine (ppm);
- **Sulfate**: solfato (mg/L);
- **Conductivity**: conducibilità (µS/cm);
- **Organic_carbon**: carbonio organico (ppm);
- **Trihalomethanes**: trihalometani (µg/L);
- **Turbidity**: torbidità (NTU);
- **Potability**: acqua non potabile (0) o potabile (1).

Prima di entrare nella fase di ottimizzazione, è stato essenziale condurre *un'analisi esplorativa dei dati (EDA)* e applicare tecniche di *Feature Engineering* al dataset. Questa fase preliminare si è resa necessaria per affrontare sfide come la gestione dei valori mancanti, il trattamento dello sbilanciamento dei dati e altre operazioni di preparazione al fine di garantire una base solida per l'ottimizzazione del modello.

2) Analisi Teorica Logistic Regression

Nonostante il nome contenga la parola "regressione" la Logistic Regression è un algoritmo di classificazione che mira a stimare la probabilità che un'osservazione appartenga a una classe specifica, comunemente indicata come classe positiva (1), rispetto alla classe negativa (0). Logistic Regression appartiene alla famiglia dei modelli discriminativi, pertanto non utilizza in alcun modo la conoscenza statistica a priori dei dati *(prior)*.
Con i modelli discriminativi, l'obiettivo è identificare il confine decisionale tra le classi per applicare etichette di classe affidabili alle istanze di dati.
Logistic Regression utilizza una funzione logistica *(sigmoide)* per mappare un valore lineare, basato sulle variabili indipendenti, in un valore compreso tra 0 e 1 che può essere interpretato come una probabilità.
La formula è la seguente:

$$P(Y_i = 1 \mid \underline{x_i}) = h(\underline{x_i}) = \frac{1}{1 + e^{-\underline{w}^T \underline{x_i}}}$$
$$P(Y_i = 0 \mid \underline{x_i}) = 1 - h\left(\underline{x_i}\right) = 1 - \frac{1}{1 + e^{-\underline{w}^T \underline{x_i}}}$$

Andiamo adesso a considerare i *log-odds*, ovvero il logaritmo naturale del rapporto tra la probabilità di un evento di appartenere alla classe 1 e alla classe 0.
Questa quantità è cruciale nella Logistic Regression poiché è usata per modellare la relazione tra le variabili indipendenti e l'output del modello.
È molto utile perché consente di modellare la relazione tra le variabili indipendenti e la probabilità di appartenenza a una classe in maniera lineare.

$$\log\left(\frac{P(Y = 1 \mid \underline{x})}{P(Y = 0 \mid \underline{x})}\right) = \underline{w^T x} + w_0$$

L'obiettivo dell'addestramento della Logistic Regression è trovare i coefficienti ottimali che massimizzino la verosimiglianza dei dati osservati. Questo processo coinvolge l'uso di tecniche di ottimizzazione come il *Gradient Descent*.

Una volta addestrato il modello, è possibile utilizzarlo per classificare nuove osservazioni. Di solito, si stabilisce una soglia di probabilità γ e si assegna il campione alla classe positiva se la probabilità stimata è maggiore della soglia, altrimenti alla classe negativa.

$$y = sigmoid(\underline{w^T x}) > \gamma$$

Una soglia pari a 0.5 è un modo per non assegnare un peso maggiore a una delle classi. Essendo una stima, il modello può commettere degli errori che, nell'ambito della classificazione, prendono il nome di *Miss Detection* e *Falso Allarme*.
Per stimare i parametri ottimi $\underline{w^*}$ è possibile utilizzare diversi approcci.
Il più comune legato alla Regressione Logistica è il *Maximum Likelihood Estimator (MLE)* che cerca di trovare i valori dei coefficienti che massimizzano la probabilità di osservare i dati effettivi dati i parametri stimati. In altre parole, cerca di trovare i parametri che rendono più probabile l'insieme di dati effettivamente osservato.
L'obiettivo è massimizzare la funzione di verosimiglianza *(likelihood)* rispetto ai dati appartenenti al *Training Set*.
Assumendo che le osservazioni in gioco siano tra loro indipendenti e identicamente distribuite *(i.i.d.)* il problema di ottimizzazione è il seguente:

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

Essendo il logaritmo una funzione monotona crescente, è possibile semplificare i calcoli andando a lavorare con quella che prende il nome di *Negative Log-Likelihood (NLL)*.
In questo caso il problema diventa di minimizzazione, la formula è la seguente:

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

Il problema di ottimizzazione per Logistic Regression non ammette una soluzione chiusa, ma può essere risolto efficacemente mediante algoritmi iterativi.
Questo è possibile grazie alla natura convessa del problema e all'assenza di termini di regolarizzazione, che ci permettono di convergere al minimo globale.

3) Analisi Teorica Algoritmi Di Ottimizzazione

Un algoritmo di ottimizzazione è un insieme di procedure matematiche e computazionali progettate per trovare la migliore soluzione possibile di un problema, cercando di minimizzare o massimizzare una funzione obiettivo, rispettando eventuali vincoli.

3.1) Gradient Descent

L'algoritmo Gradient Descent è utilizzato per trovare il minimo di una funzione costo, adattando iterativamente i parametri in direzione opposta al gradiente della funzione rispetto ai parametri stessi, con l'obiettivo di trovare il valore ottimale della funzione costo. La formula è la seguente:

$$\underline{w}^{k+1} = \underline{w}^{k} - \alpha \nabla J(\underline{w}^{k})$$

Dove:

- $\underline{w}^{k}$ rappresenta i parametri del modello al passo k;
- α rappresenta il tasso di apprendimento *(learning rate)* che è un valore positivo che controlla la dimensione del passo nell'aggiornamento dei parametri. È un iper-parametro critico in quanto determina la convergenza dell'algoritmo.
- $\nabla J(\underline{w}^{k})$ è il gradiente della funzione costo rispetto ai parametri $\underline{w}$ al passo k. Il gradiente rappresenta la direzione in cui la funzione di costo cresce più rapidamente, quindi sottraendolo dai parametri attuali, si muove in direzione opposta per cercare di raggiungere il minimo.

Per calcolare il gradiente della funzione costo è necessario conoscere la derivata della sigmoide, ovvero:

$$\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Ottenendo:

$$\nabla_{\underline{w}} J(\underline{X}, \underline{w}) = X^{T}\left(\sigma(X\underline{w}) - \underline{y}\right) = \frac{1}{N}\sum_{i=1}^{N} \underline{x}_i \left(\underline{y}_i - \sigma(\underline{w}^{T}\underline{x}_i)\right)$$

Sostituendo il gradiente nella formula del Gradient Descent si ottiene infine:

$$\underline{w}^{k+1} = \underline{w}^{k} - \alpha X^{T}\left(\underline{y} - \sigma\left(\underline{w}^{k}X\right)\right) * \frac{1}{N}$$

Questo processo viene ripetuto iterativamente fino a quando una condizione di stop, come un numero massimo di iterazioni o una soglia di convergenza, viene soddisfatta.

3.2) Algoritmo di Newton

L'ottimizzazione mediante l'algoritmo di Newton è una tecnica avanzata che mira a trovare il minimo di una funzione obiettivo, utilizzando informazioni sul secondo ordine della funzione (derivata seconda), sfruttando la matrice Hessiana. La formula è la seguente:

$$\underline{w}^{k+1} = \underline{w}^{k} - \alpha\left(H\left(J(\underline{x})\right)^{-1}\right)\nabla J(\underline{w}^{k})$$

L'algoritmo di Newton è noto per la sua convergenza rapida (sfrutta la curvatura della funzione anziché solo la direzione) quando la funzione obiettivo è strettamente convessa e la matrice Hessiana è definita positiva. Tuttavia, può essere computazionalmente costoso calcolare e invertire la matrice Hessiana, quindi in alcune situazioni potrebbe non essere il metodo più efficiente. Nel contesto dell'ottimizzazione mediante l'algoritmo di Newton, il concetto di funzione strettamente convessa è importante perché garantisce che la funzione obiettivo abbia un unico minimo globale e che l'algoritmo converga a questo minimo globale in modo efficiente e affidabile.
Quando una funzione è strettamente convessa, la matrice Hessiana è definita positiva in ogni punto, il che significa che tutti gli autovalori della matrice Hessiana sono positivi. Logistic Regression garantisce questa condizione.
Dalla derivata seconda si genera la matrice Hessiana i cui elementi sulla diagonale sono i prodotti delle probabilità a posteriori delle classi, ovvero:

$$diag(H) = P = \sigma\left(\underline{x}_1\underline{w}\right) * \left(1 - \sigma\left(\underline{x}_1\underline{w}\right)\right), \ldots, \sigma\left(\underline{x}_N\underline{w}\right) * \left(1 - \sigma\left(\underline{x}_N\underline{w}\right)\right)$$

Poiché questi prodotti sono sempre positivi, ne segue che la matrice Hessiana è definita positiva, il che rende possibile l'applicazione dell'algoritmo di Newton per l'ottimizzazione della Logistic Regression.

La formula finale per l'aggiornamento dei pesi è la seguente:

$$\underline{w}^{k+1} = \underline{w}^{k} - \alpha(X^T P X)^{-1} \left(X^T \left(\underline{y} - \sigma\left(\underline{w}^{k} X\right) \right) * \frac{1}{N} \right)$$

3.3) ADMM Distribuito (Split by Data)

ADMM (alternating direction method of multipliers) distribuito con split dei dati è un algoritmo di ottimizzazione che consente di risolvere problemi di ottimizzazione distribuita suddividendo i dati in sottoinsiemi e distribuendo il calcolo tra diversi nodi o dispositivi di calcolo. È particolarmente utile quando si desidera addestrare un modello su un grande dataset distribuito tra più dispositivi, come in un ambiente di calcolo distribuito o in un'applicazione di edge computing.

Il dataset completo viene suddiviso in sottoinsiemi, ognuno dei quali è assegnato a un nodo o un dispositivo di calcolo separato. Ogni nodo è responsabile solo del calcolo sul suo sottoinsieme di dati. In ogni nodo, viene eseguita l'ottimizzazione locale, spesso utilizzando un algoritmo come il Gradient Descent o il metodo di Newton per addestrare il modello sui dati locali. L'obiettivo è minimizzare la funzione costo relativa ai dati di quel nodo. Periodicamente, i nodi condividono informazioni tra loro. Questo scambio di informazioni può includere i parametri del modello o altre variabili rilevanti.

L'ADMM utilizza queste informazioni per garantire la coerenza tra i nodi.

L'algoritmo ADMM utilizza i moltiplicatori di Lagrange per imporre dei vincoli, condivisi tra i nodi. Questi moltiplicatori vengono aggiornati in base alle informazioni condivise e utilizzati per garantire la convergenza dell'algoritmo. L'algoritmo viene ripetuto iterativamente fino a quando non converge a una soluzione ottima o soddisfa un criterio di arresto.

ADMM è efficace quando il problema di ottimizzazione può essere suddiviso in sotto-funzioni minimizzabili separatamente, cioè quando il problema può essere formulato in modo che la funzione costo globale sia una somma di funzioni separabili, ciascuna associata a un sottoinsieme dei dati.

La Regressione Logistica soddisfa questa condizione.

$$f(\underline{w}) = \sum_{i=1}^{N} f_i(\underline{w})$$

Quando si applica l'ADMM in versione centralizzata, si introduce spesso una *slack-variable* e un vincolo di uguaglianza per separare l'ottimizzazione della funzione obiettivo principale da quella di un eventuale parte regolarizzante del problema.
La slack variable, spesso indicata con il termine *variabile globale z,* viene introdotta per "assorbire" il termine di regolarizzazione della funzione obiettivo.
In altre parole, se la funzione obiettivo principale include un termine di regolarizzazione, la slack variable è utilizzata per rappresentare questo termine in modo separato.
Quindi, la funzione obiettivo principale contiene solo la parte non regolarizzata.
Il vincolo di uguaglianza (generalmente $w - z = 0$) assicura che w e z siano coerenti tra loro durante il processo di ottimizzazione. La formula è la seguente:

$$\min_{\underline{w_1}, \underline{w_2}, .., \underline{w_N}, \ \underline{z}} \sum_{i=1}^{N} f_i\left(\underline{w_i}\right)$$
$$s.t. \underline{w_i} = \underline{z} \ \forall \ i = 1, \dots, N$$

ADMM appartiene alla *famiglia degli algoritmi Primali-Duali* e pertanto sfrutta la formulazione duale del problema.
Ad ogni variabile locale viene associata una sotto-funzione costo e tutte dipendono dalla stessa slack variable z, in modo da far tendere i sotto-problemi locali alla stessa soluzione.
Questo approccio è conosciuto anche come *ottimizzazione al consenso.*
La forma del Lagrangiano aumentato è la seguente:

$$L\left(\underline{w}_1, \dots, \underline{w}_N, \underline{z}, \underline{\mu}_1, \dots, \underline{\mu}_N\right) = \sum_{i=1}^{N} f_i(\underline{w_i}) + \sum_{i=1}^{N} \mu_i \left(\underline{w_i} - \underline{z}\right) + \rho \sum_{i=1}^{N} \left\| \underline{w_i} - \underline{z} \right\|_2^2$$

Mentre le equazioni di aggiornamento delle variabili *(versione scalata)* sono le seguenti:

$$1) \ \underline{w_i}^{(k+1)} = \min_{\mathrm{w_i}} \left(-\frac{1}{N} \left(\underline{y_i} \ log\left(\sigma\left(X^j \underline{w}\right)\right) + \left(1 - \underline{y_i}\right) \log\left(1 - \sigma\left(X^j \underline{w}\right)\right)\right) + \rho \left\| \underline{w_i - z^k + u_i^k} \right\|_2^2 \right)$$

$$2) \ \underline{z}^{(k+1)} = \frac{1}{N} \left(\sum_{i=1}^{N} \underline{w_i}^{(k+1)} + \sum_{i=1}^{N} \underline{u_i^k} \right)$$

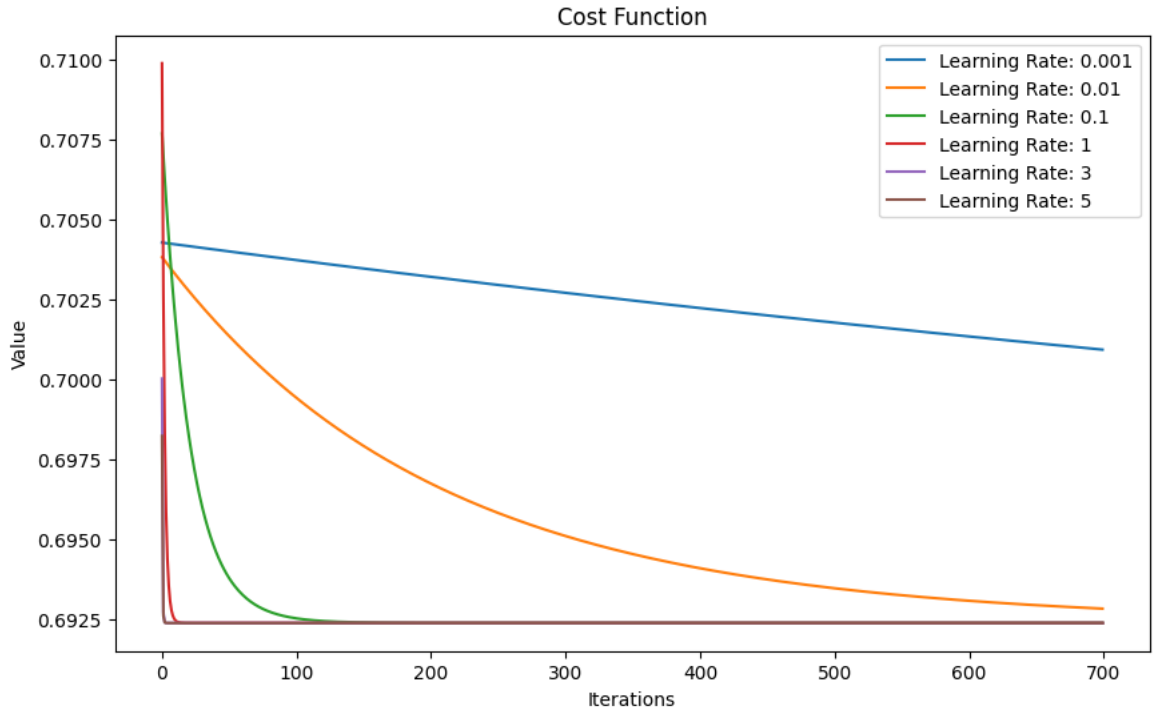
$$3) \ \underline{u_i}^{k+1} = \underline{u_i^k} + \underline{(w_i^{k+1} - z^{k+1})}$$

Nel primo passo ogni nodo *(detto anche agente)* calcola localmente il proprio problema, utilizzando la conoscenza delle variabili di ottimizzazione aggiornate al passo precedente. È anch'esso un problema di ottimizzazione, risolvibile tramite una soluzione come il Gradient Descent. Questa fase è distribuita.
Nel secondo passo avviene l'aggiornamento della variabile globale che altro non è che una media delle variabili primali e duali locali. Questo passaggio è centralizzato poiché richiede la conoscenza di tutte le soluzioni dei vari agenti locali. Tale aggiornamento viene dunque affidato ad un altro nodo di calcolo che si occuperà successivamente di diffondere il risultato a tutti gli agenti in gioco. Tale nodo prende il nome di *Fusion Center*.
Nel terzo passo avviene l'aggiornamento della variabile duale. Per la Teoria della Dualità, massimizzare il problema duale equivale a minimizzare il problema primale.

4) Spiegazione dei Risultati Ottenuti

4.1) Risultati Gradient Descent

Sono stati provati diversi valori di learning rate in modo da poter osservare le differenze in termini di velocità di convergenza. La scelta dell'iper-parametro learning rate (step-size) è importante in quanto controlla la dimensione del passo con cui vengono aggiornati i parametri del modello durante l'iterazione per minimizzare la funzione di costo.
Una learning rate grande può far convergere l'algoritmo più rapidamente, ma potrebbe causare oscillazioni o problemi di divergenza. Al contrario, una learning rate troppo piccola può rendere l'addestramento molto lento.




```
# Selecting the best learning rate with the lowest cost function value
best_learning_rate = learning_rates[np.argmin([entry[-1] for entry in cost_function_values])]
print(f"Best learning rate: {best_learning_rate}")
```

[17] Python

```
Best learning rate: 3
```

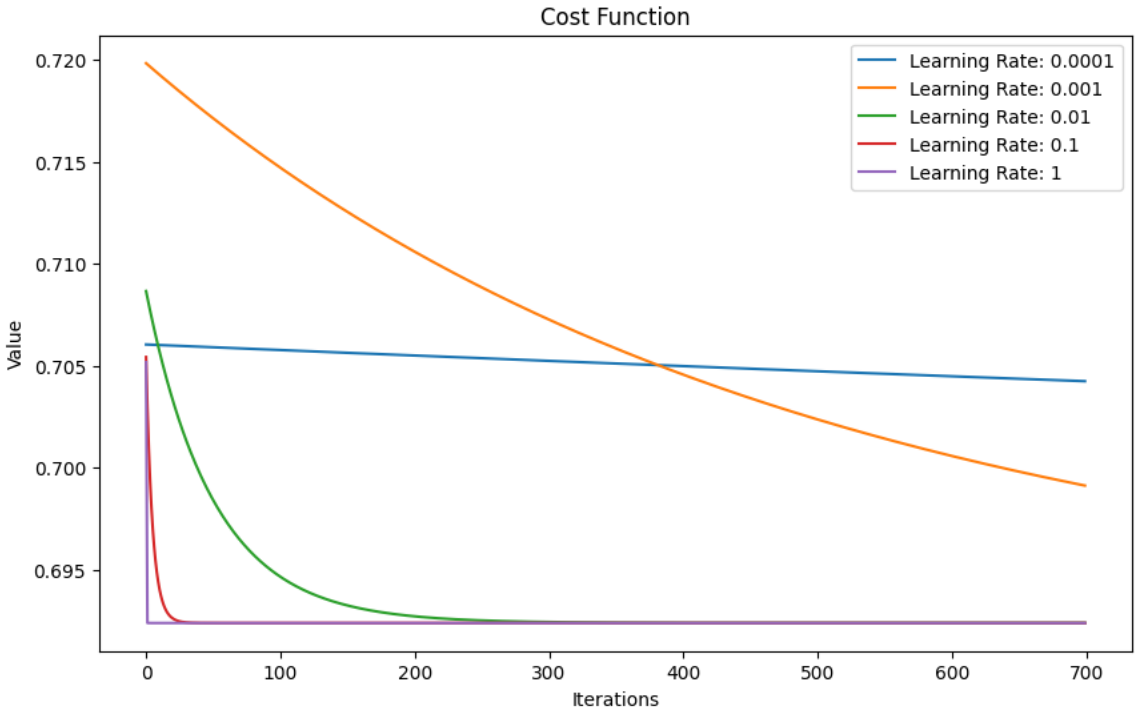
Il massimo numero di iterazioni è stato impostato a 700, in quanto il dataset testato non presenta un numero elevato di campioni. È stata impostata una threshold pari a 10^{-7} per arrestare in anticipo l'algoritmo, in caso di suo raggiungimento.

4.2) Risultati Algoritmo di Newton

Le condizioni che circondano l'uso dell'algoritmo di Newton sono leggermente diverse rispetto al Gradient Descent. Mentre è vero che l'algoritmo di Newton tende a convergere verso la soluzione ottimale in meno iterazioni rispetto al Gradient Descent, presenta una contropartita computazionalmente più onerosa.
Questo significa che, nonostante la sua convergenza più veloce, l'esecuzione complessiva dell'algoritmo di Newton richiede più tempo rispetto a quella del Gradient Descent.

Per mitigare il rischio di problemi di non invertibilità della matrice Hessiana, è stata introdotta una regolarizzazione. Nella pratica consiste nell'aggiungere una matrice d'identità moltiplicata per un piccolo valore, in questo caso 0.01, alla matrice Hessiana. Questa regolarizzazione aiuta a garantire che la matrice Hessiana rimanga invertibile e che l'algoritmo di Newton possa essere applicato in modo stabile senza problemi numerici.

```
# Computing the Hessian matrix
hessian = np.dot(X_train.T, np.dot(np.diag(y * (1 - y)), X_train)) / n_samples + l1_reg * np.eye(n_parameters)
```




```
# Selecting the best learning rate with the lowest cost function value
best_learning_rate = learning_rates[np.argmin([entry[-1] for entry in cost_function_values])]
print(f"Best learning rate: {best_learning_rate}")
```

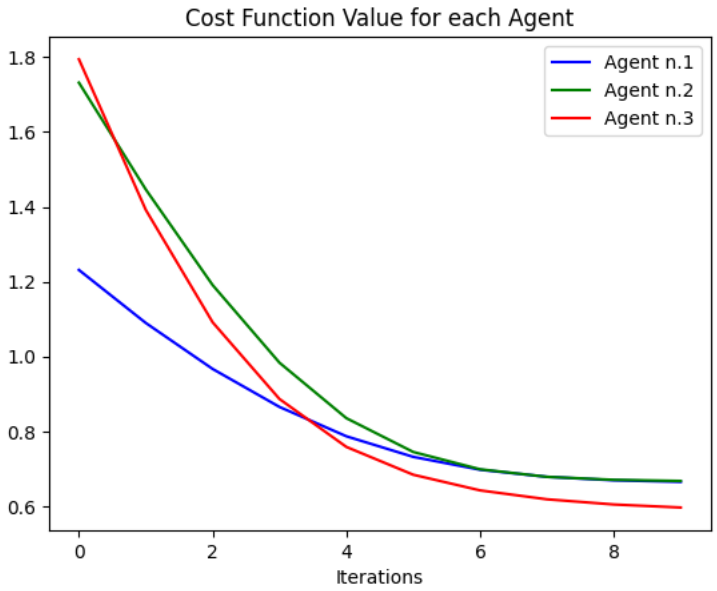
[5] Python

```
Best learning rate: 1
```

4.3) Risultati ADMM

Per l'algoritmo ADMM sono stati utilizzati 3 agenti, in quanto il numero di campioni contenuti dal dataset testato non è molto elevato.

Così facendo viene simulata la distribuzione dei dati e l'ottimizzazione locale.

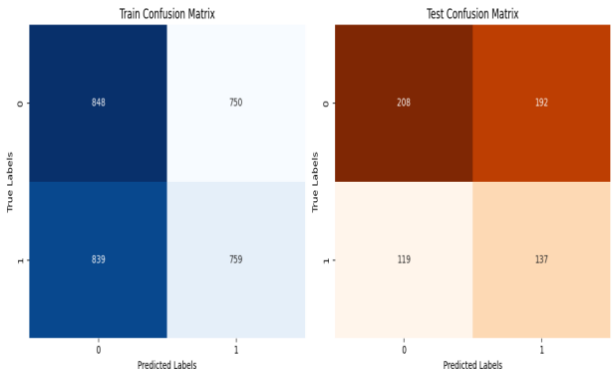


Ogni agente al termine delle iterazioni dispone del proprio set di parametri.

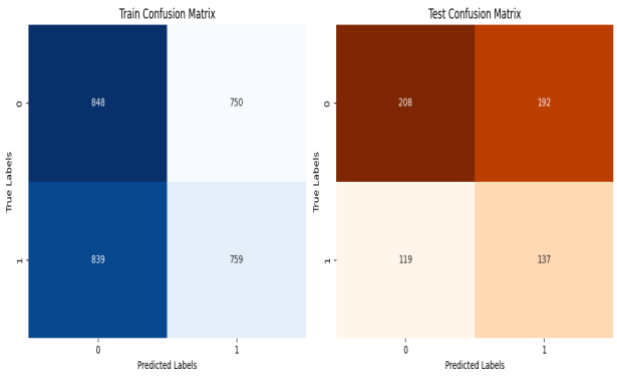
	Agent 1	Agent 2	Agent 3	z
0	−0.481807	−0.434910	0.981161	0.021481
1	0.046405	−0.002012	0.099646	0.048013
2	−0.006264	0.032219	−0.040828	−0.004958
3	0.030963	0.088087	0.024644	0.047898
4	0.004818	0.135682	−0.032156	0.036115
5	−0.027558	0.006207	−0.039457	−0.020269
6	0.013379	0.039174	−0.029244	0.007770
7	0.059513	−0.076544	−0.109856	−0.042295
8	0.120183	−0.077903	−0.000742	0.013846
9	0.068753	−0.007785	−0.101807	−0.013613

5) Considerazioni Finali

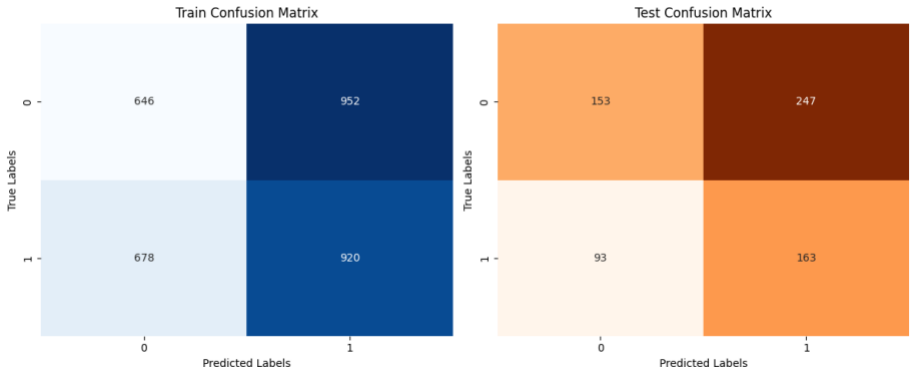
Sfortunatamente, in questo specifico caso, i risultati ottenuti non sono stati soddisfacenti. Il problema principale nasce dal fatto che il dataset utilizzato è sbilanciato, ovvero presenta una differenza significativa tra il numero di campioni appartenenti alla classe 0 e quelli appartenenti alla classe 1. Ciò ha reso difficile l'addestramento di un modello accurato e nonostante sia stata applicata una tecnica di bilanciamento dei dati (SMOTE), i miglioramenti sono stati limitati e non hanno portato a una soluzione soddisfacente.
C'è il sospetto che i dati utilizzati per la prova siano stati generati sinteticamente, con dati non reali, come evidenziato anche da altri utenti del sito (https://www.kaggle.com/datasets/adityakadiwal/water-potability/discussion/248871) il che ha reso praticamente impossibile sviluppare un modello efficace in un contesto in cui la qualità dei dati è essenziale. Di seguito vengono riportati alcuni dei risultati ottenuti, altri sono disponibili all'interno del codice.



Confusion Matrix – Gradient Descent



Confusion Matrix – Newton Algorithm



Confusion Matrix – ADMM Algorithm