Tesina di
Signal Processing and Optimization for Big Data

Corso di Laurea Magistrale in
Ingegneria Informatica e Robotica
curriculum data science

A.A. 2021/22

docente
Paolo Banelli

ADDESTRAMENTO DEL MODELLO DI CLASSIFICAZIONE LOGISTIC REGRESSION IMPLEMENTANDO VARI ALGORITMI DI OTTIMIZZAZIONE

334568 **Andrea Baldinelli** andrea.baldinelli@studenti.unipg.it

Indice

1 Introduzione

Lo scopo di questa tesina è l'implementazione alcune tecniche algoritmiche per la risoluzione di problemi di ottimizzazione, nel contesto specifico di addestramento di un modello di machine learning come *Logistic Regression*; un modello adibito alla classificazione binaria delle osservazioni date in input.

1.1 Descrizione dei dati

Al fine di poter implementare un modello di machine learning, c'è bisogno di un insieme di dati da cui partire. Per questa tesina è stato scelto un dataset in cui sono stati raccolti dati (anonimizzati) di persone, spaziando da parametri medici a informazioni più di carattere personale. Una generica *osservazione* all'interno del dataset è composta dalle seguenti informazioni (escludendo l'identificativo della stessa all'interno dei dati):

- **Età del paziente**: variabile di tipo numerico;
- **Gender**: variabile di tipo stringa;
- **Hypertension**: una variabile binaria che descrive se il paziente soffre di ipertensione(1) o meno (0);
- **Heart disease**: una variabile binaria che descrive se il paziente soffre di patologie cardiache (1) o no (0);
- **Ever married**: una variabile che descrive se il paziente si sia mai sposato o meno;
- **Work type**: variabile categorica con cui si indica, ad alto livello, l'impiego del paziente (nel dettaglio si indica se lavora: nel pubblico; nel privato; se è libero professionista; disoccupato oppure è un bambino);
- **Residence** : altra variabile categorica con cui si indica se il paziente vive in città o in zone rurali;
- **Avg glucose level**: viene indicato il livello medio di glucosio nel sangue. Va sottolineato come un valore medio di 200 mmol/L indica che il paziente è diabetico;
- **BMI:** *Indice di massa corporea.* una metrica che descrive il peso corporeo del paziente;
- **Smoking status**: una variabile che descrive se il paziente è un fumatore, se lo è stato o se non ha mai fumato (oppure se questa informazione non è disponibile);
- **Stroke**: variabile binaria che descrive se il paziente ha avuto o meno ictus cerebrali.

Come lasciato intendere nella descrizione precedente, ci sono alcune features che non assumono valore numerico. Affinché sia possibile addestrare un modello di machine learning, è necessario che essi siano convertiti tutti in valori numerici; pertanto si è vista necessaria una preliminare fase di *Exploratory Data Analysis* (in cui si cerca di capire quale relazione ogni feature ha con la variabile di uscita) seguita da una fase di *Feature Engineering* (in cui si vanno a trasformare le features "incriminate" nella migliore rappresentazione possibile). Questa fase preliminare è descritta più nel dettaglio successivamente, al capitolo **EXTRA**

1.2 Descrizione del problema

In questa tesina si è andato ad addestrare il modello Logistic Regression in modo da andare a classificare se un paziente può essere potenzialmente a rischio di contrarre un ictus cerebrale oppure no.
L' obiettivo "vero" della tesina però non si concentra sul classificatore in sé, bensì sugli algoritmi utilizzati per l'addestramento dello stesso, infatti la domanda a cui si vuol rispondere è: **c'è differenza prestazionale (in termini di accuratezza nella classificazione) tra l'utilizzare un certo algoritmo rispetto ad un altro? Tra un approccio centralizzato ed uno distribuito?**
Gli algoritmi presi in esame sono:

- Gradient descent (anche con regolarizzazione)
- Algoritmo di Newton
- ADMM distribuito (con *fusion center*) con una distribuzione dei dati fra gli agenti

2 Analisi teorica del problema

Prima di discutere i risultati ottenuti, è necessario fare una trattazione teorica di quello che è stato fatto e di ciò che è stato utilizzato.

2.1 Logistic Regression

Logistic Regression (binario) è un modello di machine learning appartenente alla famiglia dei modelli discriminativi, ovvero quella famiglia di modelli che nella fase di addestramento non vanno a sfruttare informazioni statistiche sulla distribuzione dei dati a priori (come la famiglia dei modelli generativi) ma nemmeno del tutto agnostici alla probabilità come la famiglia delle funzioni discriminative (ad esempio SVM). Logistic Regression va a modellare la distribuzione a posteriori delle classi (2, nel caso binario) in funzione dell' osservazione in esame.

$$P(Y_i = 1|\underline{x_i})$$
$$P(Y_i = 0|\underline{x_i}) = 1 - P(Y_i = 1|\underline{x_i})$$

Questo è quello che, in generale, si va a fare con un classificatore di tipo bayesiano, soltanto che in questo caso la modellazione avviene in modo *diretto* andando a sfruttare l'assunzione alla base di Logistic Regression:

$$\log(\frac{P(Y_i = 1|\underline{x_i})}{P(Y_i = 0|\underline{x_i})}) = \underline{w^T x_i} + w_0$$

dove il vettore $\underline{w}^T$ più il termine intercetta w_0 sono parametri non noti a priori, pertanto devono essere stimati dal nostro set di osservazioni.
L'assunzione che viene fatta è che si approssima il *log Maximum A Posteriori Detector* (una variante sul tema del *Log Likelihood Ratio Test* quando si lavora con approccio bayesiano) con un iperpiano (lavorando in $\Re^n$).
Da questa formulazione, sfruttando il fatto che la somma delle due probabilità a posteriori faccia sempre 1, possiamo ricavare le formule che ci permettono di modellare le probabilità nel seguente modo:

$$P(Y_i = 1|\underline{x_i}) = \frac{1}{1 + \exp(-\underline{w^T x_i} - w_0)}$$
$$P(Y_i = 0|\underline{x_i}) = \frac{\exp(-\underline{w^T x_i} - w_0)}{1 + \exp(-\underline{w^T x_i} - w_0)}$$

Quella funzione che modella $P(Y_i=1 \mid x_i)$, al variare dell'osservazione in esame, è una funzione nota, prende il nome di *sigmoide*. è una funzione che è definita su tutto $\Re$ mentre il codominio è un insieme di valori compreso tra [0,1]; pertanto è anche ragionevole che tale funzione vada a modellare la probabilità di un evento (in questo caso l'evento è la *classificazione* dell'osservazione).
La classificazione vera e propria, in funzione della probabilità a posteriori sulle classi, avviene andando a valutare il risultato della sigmoide in funzione di una soglia (che nel complesso vanno a definire la *funzione decisore*):

$$\phi(\underline{x_i}) = u_{-1}(sigmoid(\underline{w^T x_i}) \geq \gamma)$$

il valore di soglia γ viene impostato in funzione del problema di classificazione in esame, infatti il valore della soglia determina sia la probabilità di classificare correttamente un'osservazione ma anche la probabilità di classificare in maniera errata un campione (quando si classifica come un campione come classe 1 ma in realtà era un campione di classe 0) definita anche probabilità di *falso allarme*.
Per poter ottenere il set di parametri, si può pensare di andare ad utilizzare un approccio molto comune nel mondo del machine learning, ovvero stimarli per mezzo di un approccio *Maximum Likelihood Estimator* , rispetto alle osservazioni che appartengono al training set:

$$\max_{\underline{w}} f_{XY}(X, Y; \underline{w})$$

Sfruttando il teorema di Bayes e assumendo che le nostre osservazioni siano indipendenti e identicamente distribuite, si ottiene il seguente problema di ottimizzazione:

$$\max_{\underline{w}} \sum_{i=1} (P(y_i = 1|\underline{x_i}))^{y_i} (1 - P(y_i = 1|\underline{x_i}))^{1-y_i}$$

andando a riscrivere il problema come una minimizzazione e lavorando con la log-likelihood:

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

Così formulato il problema è non vincolato. convesso, quindi sappiamo che garantisce l'esistenza di un unico minimo globale. Rispetto ad altri problemi di ottimizzazione, non ammette una soluzione in forma chiusa, pertanto è possibile risolverlo per mezzo di algoritmi iterativi di ricerca. Di seguito verranno enunciati gli algoritmi andando a esplicitare qual è la soluzione iterativa di tale problema.

2.2 Algoritmi di ottimizzazione implementati

2.2.1 Gradient Descent

Gradient Descent è una tecnica che permette di ottenere in modo iterativo la soluzione di un problema di ottimizzazione.
In generale si ottiene che la politica di aggiornamento della soluzione si:

$$\underline{x^{k+1}} = \underline{x^k} - step^k \nabla J(\underline{x})|_{\underline{x=x^k}}$$

Ove *step* può essere concepito come il "peso" che viene dato all'informazione data dal passo di gradiente; non va sottovalutato in quanto determina la convergenza dell'algoritmo! (nel capitolo successivo verrà approfondito questo argomento). Il gradiente della funzione da ottimizzare valutata con la *soluzione corrente* permette di indicare la "direzione" lungo la quale ricercare il valore ottimo.
Nel caso di Logistic Regression, sfruttando il fatto che la derivata della funzione sigmoide risulta essere:

$$sigmoid(x) = sigmoid(x)(1 - sigmoid(x))$$

si ricava che il gradiente della funzione obiettivo definita precedentemente ,rispetto al vettore dei parametri, risulta essere:

$$\nabla_{\underline{w}} J(Xw) = X^T(sigmoid(X\underline{w}) - \underline{Y})$$

Ove X è quella che viene chiamata *design matrix*, una matrice che contiene per riga le osservazioni del dataset, **con in aggiunta una *feature dummy***, che serve per considerare il parametro w_0 all'interno del vettore dei parametri $\underline{w}$.
Pertanto si ottiene la seguente equazione di aggiornamento della soluzione:

$$\underline{w^{k+1}} = \underline{w^k} - stepX^T(sigmoid(X\underline{w^k}) - \underline{Y})$$

L'aggiornamento della soluzione avviene fino a quando non viene soddisfatto il criterio di stop:

$$\|J(X\underline{w^{k+1}}) - J(X\underline{w^{k}})\| \leq \epsilon$$

ove il valore di ϵ deve essere un valore sufficientemente piccolo, a scelta dello sviluppatore.
La semplicità di gradient descent viene "compensata" da una lentezza nel convergere alla soluzione ottima. Si può dimostrare che la velocità di convergenza alla soluzione ottima è governata dalla velocità a convergere della serie

$$\sum_{k=1}^{\inf} c^k$$

dove c è una costante definita nell'intervallo [0,1], che sintetizza in un unico valore tutte le scelte progettuali fatte (valore di partenza della soluzione w^0, lo step size e la dimensionalità del problema); tanto più c tende a 1, tanto più la convergenza sarà lenta.

2.2.1 Newton algorithm

Questo algoritmo è concettualmente simile al gradient descent, la differenza risiede nella direzione lungo la quale viene cercata la soluzione: infatti viene utilizzata anche la derivata seconda della funzione obiettivo, ovvero la matrice hessiana.

$$\underline{x^{k+1}} = \underline{x^{k}} - step(H(J(x))^{-1}\nabla J(\underline{x})|_{\underline{x=x^k}}$$

Rispetto al gradient descent, in questo caso è richiesta la convessità stretta della funzione costo, in quanto viene utilizzata la matrice inversa della matrice hessiana, che risulta definita positiva (condizione necessaria per l'invertibilità) se e soltanto se la funzione associata è strettamente convessa. Questa cosa risulta essere garantita dalla funzione costo di Logistic Regression, infatti andando a derivare il gradiente ottenuto per gradient descent si ottiene

$$H(J(Xw)) = \nabla_{\underline{w}}(\nabla_{\underline{w}} J(X, \underline{w})) = X^T P X$$

dove la matrice P è una matrice diagonale siffatta

$$P = diag[sigmoid(\underline{x_1 w})(1-sigmoid(\underline{x_1 w})), ..., sigmoid(\underline{x_N w})(1-sigmoid(\underline{x_N w})]$$

L'elemento i-esimo sulla diagonale non è altro che il prodotto delle probabilità a posteriori delle classi rispetto all'osservazione i-esima.
Indirettamente si è ottenuta la decomposizione in autovalori - autovettori della matrice hessiana; constatando che gli autovalori della matrice non sono altro che i valori sulla diagonale della matrice P, che sono tutti valori compresi tra]0,1[, si può asserire che la matrice hessiana è definita positiva, quindi si può applicare l'algoritmo di Newton al problema di ottimizzazione di Logistic Regression.

L'equazione di aggiornamento che si ottiene è la seguente:

$$\underline{x^{k+1}} = \underline{x^k} - step(X^T P(\underline{w^k})X)^{-1} X^T (sigmoid(X\underline{w^k}) - \underline{Y})$$

Rispetto a gradient descent, a parità di criterio di stop, l'algoritmo di Newton impiega meno iterazioni per ottenere la soluzione al problema (è ragionevole che ciò accada, visto che l'algoritmo di Newton decide la direzione di ricerca anche tenendo conto di informazioni sulla curvatura della funzione); di contro si ha che è più costoso (in termini di complessità algoritmica) il singolo aggiornamento della soluzione.

2.2.3 ADMM distribuito (*split by data)*

Per ora sono stati descritti algoritmi di tipo centralizzato, ovvero l'algoritmo viene eseguito su un solo calcolatore che ha a disposizione tutto il set di dati. Visto che la mole di dati in gioco in questi contesti è sempre più grande, sono state definite delle tecniche per ottenere la soluzione a problemi di ottimizzazione anche in modo distribuito, quindi problemi che vengono risolti *cooperativamente* da più calcolatori! Esistono due possibilità "di distribuzione": ***distribuzione dei dati*** fra i vari calcolatori; ***distribuzione delle variabili di ottimizzazione*** tra i vari calcolatori. In questa tesina si prende in esame la distribuzione dei dati.
Il framework algoritmico distribuito preso in esame ha senso di essere applicato soltanto se la funzione obiettivo del problema di minimizzazione è possibile scomporla in una somma di sotto-funzioni obiettivo.

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

Questo, implicitamente, significa anche che ogni sotto funzione sarà definita rispetto ad un sottoinsieme di dati dell'insieme di dati originale. Questa ipotesi risulta essere verificata per la funzione costo di Logistic Regression, visto che già nella sua formulazione centralizzata la funzione costo è stata definita come somma di singoli contributi.
Entrando nei dettagli, l'algoritmo distribuito di *Alternating Direction Method of Multipliers* è un algoritmo di ottimizzazione appartenente alla famiglia dei *primal-dual algorithms*, ovvero quei algoritmi che sfruttano la formulazione duale del problema (che sfrutta la funzione *lagrangiana*). La particolarità di ADMM (in versione centralizzata) è che permette di suddividere l'ottimizzazione della funzione obiettivo da quella di un eventuale parte regolarizzante del problema andando a definire quella che viene definita come *slack variable* e un vincolo di uguaglianza che lega la variabile di ottimizzazione principale e la variabile aggiunta.

Esplicitando le ipotesi sulla funzione obiettivo decomponibile, unite al concetto di *slack variable* di addm centralizzto, si può pensare di riscrivere ci permette di formulare un problema ottimizzazione distribuito rispetto ai dati nel seguente modo, ove vengono introdotte **copie della variabili di ottimizzazione**

$$min_{\underline{x_1} \dots \underline{x_N}, \underline{z}} \quad \sum_{i=1}^{N} f_i(\underline{x_i})$$
$$s.t. \qquad \underline{x_i} = \underline{z} \qquad \forall i = 1, ., N$$

Ogni copia delle variabili (chiamate anche *variabili locali*) è associata ad una sotto funzione costo, ma tutte dipendono dalla stessa *slack variable* (definita anche *variabile globale*).
Questa formulazione permette una risoluzione distribuita, andando ad esplicitare il framework ADMM, definendo la funzione lagrangiana aumentata associata nel seguente modo:

$$L(\underline{x_1}, ..., \underline{x_N}, \underline{z}, \underline{\mu_1}, ..., \underline{\mu_N}) = \sum_{i=1}^{N} f_i(\underline{x_i}) + \sum_{i=1}^{N} \mu_i(\underline{x_i} - \underline{z}) + \rho \sum_{i=1}^{N} \|\underline{x_i} - \underline{z}\|_2^2$$

si ottengono le seguenti equazioni di aggiornamento (nella versione *scalata*) delle variabili locali, di quelle *duali locali* e della variabile globale:

$$1) \quad \forall i = 1, \ ....., \ N \qquad \underline{x_i}^{k+1} = min_{x_i} \quad (f_i(\underline{x_i}) + \rho \|\underline{x_i} - \underline{z^k} + \underline{u_i^k}\|_2^2)$$

$$2) \quad \underline{z^{k+1}} = \frac{1}{N}(\sum_{i=1}^{N} \underline{x_i^{k+1}} + \sum_{i=1}^{N} \underline{u_i^k})$$

$$3) \quad \forall i = 1, \ ....., \ N \qquad \underline{u_i^{k+1}} \ = \ \underline{u_i^k} + \underline{x_i^{k+1}} - \underline{z^{k+1}}$$

Osservando le equazioni, si comprende perché definire le copie della variabile primale di ottimizzazione ci permettano di distribuire l'ottimizzazione, infatti si può evincere dalla prima equazione come l'aggiornamento della copia locale dipenda solo dalla "propria" funzione di ottimizzazione (quindi ottimizzata rispetto ad una **porzione del dataset originale**).
Questo non significa che i sottoproblemi locali siano del tutto indipendenti fra di loro, bensì sono tutti dipendenti alla variabile globale **z**, il cui aggiornamento non è altro che una media delle variabili primali e duali locali. **Questo tipo di vincolo induce i sottoproblemi locali a tendere, per quanto ottimizzate su dati diversi, ad ottenere la stessa soluzione**! Per questo motivo, questa tecnica di ottimizzazione prende anche il nome di *ottimizzazione al consenso*, in quanto alla fine tutte le soluzioni dei sottoproblemi convergono al valore della variabile globale.

Osservando nel dettaglio l'aggiornamento della variabile globale, è evidente che quello non può essere, così come è scritto, un calcolo distribuibile su più calcolatori in quanto è necessario conoscere tutte le copie locali di tutte le variabili; questo tipo di calcolo viene fatto da un calcolatore (che prende il nome di *fusion center*) che riceve tutte le soluzioni locali e ne calcola la media e successivamente restituisce il risultato a tutti i nodi del cluster di calcolatori. E' possibile definire un approccio di risoluzione del problema *completamente distribuito*, ovvero che non sfrutta direttamente una variabile globale, ma non rientra nelle tematiche trattate da questa tesina.

Applicando quanto detto allo scenario di ottimizzazione di Logistic Regression, si è cercato di simulare questo tipo di strategia di ottimizzazione definendo "virtualmente" 10 agenti su cui viene distribuito il calcolo delle copie locali delle variabili primali e duali. In questo modo; pertanto, osservando che la funzione di costo è decomponibile perfettamente in somma di sotto funzioni

$$J(X, \underline{w}) = -\sum_{j=1}^{nagents} [\underline{Y^j} \log(\underline{sigmoid(X^j \underline{w})}) + (\underline{1} - \underline{Y^j}) \log(\underline{1} - \underline{sigmoid(X \underline{w})}]$$

ove $\underline{Y^i}$ è un vettore con (n_dati/n_agenti) componenti mentre X^j è una porzione di *design matrix* contenente (n_dati/n_agenti) righe (osservazioni) e tutte le colonne della matrice originaria.

Le equazioni di aggiornamento del problema di ottimizzazione, secondo ADMM, diventano:

$$1) \quad \forall i = 1, \ldots\ldots, N$$

$$\underline{w_i}^{k+1} = \quad min_{w_i}(-[\underline{Y^j} \log(\underline{\sigma(X^j \underline{w})}) + (\underline{1} - \underline{Y^j}) \log(\underline{1} - \underline{\sigma(X^j \underline{w})}] + \rho \|\underline{x_i} - \underline{z^k} + \underline{u_i^k}\|_2^2)$$

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

$$3) \quad \forall i = 1, \ldots\ldots, N \qquad \underline{u_i^{k+1}} = \underline{u_i^k} + \underline{w_i^{k+1}} - \underline{z^{k+1}}$$

In questo caso quindi, ad ogni step dell'algoritmo, ogni calcolatore deve andarsi a risolvere un sottoproblema di Logistic Regression per ottenere la soluzione locale ottima al passo corrente! Non avendo una soluzione in forma chiusa, come già sottolineato in precedenza, in questa tesina viene applicato l'algoritmo *gradient descent* ad ogni step della soluzione.

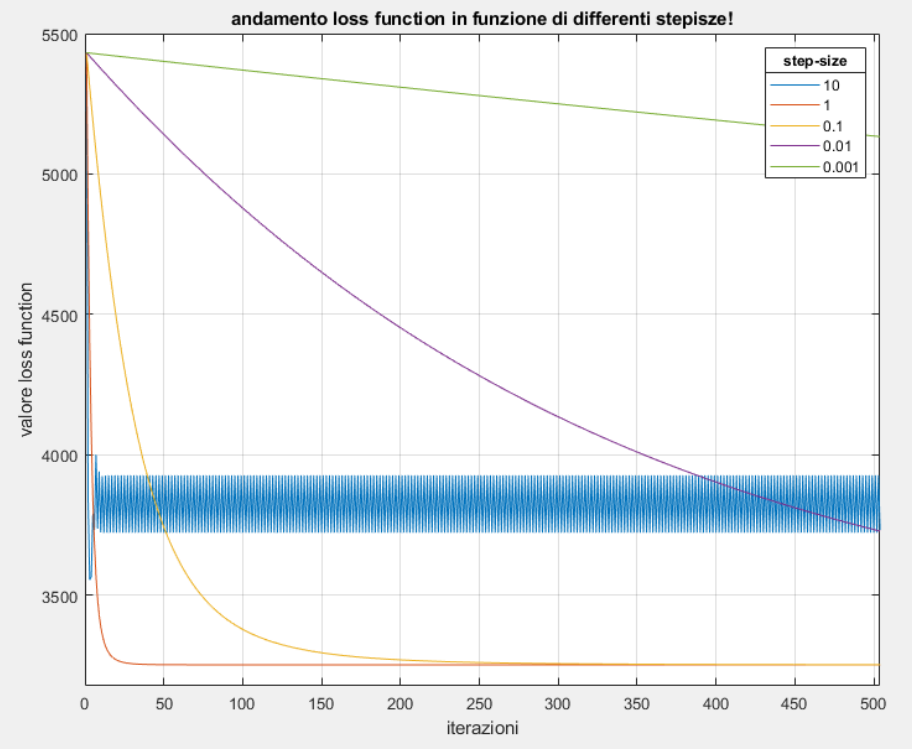
La convergenza di ADMM è garantita nel caso di funzione differenziabile (come nel caso in esame), dipendente dallo step di aggiornamento delle variabili locali. In generale non è molto veloce a convergere alla soluzione esatta, però dopo pochi step di aggiornamento già si tende ad avere un buon vettore di soluzioni.

3 Risultati sperimentali

In questa parte di tesina vengono discussi i risultati prestazionali dell'addestramento di Logistic Regression in funzione dei vari algoritmi commentati.

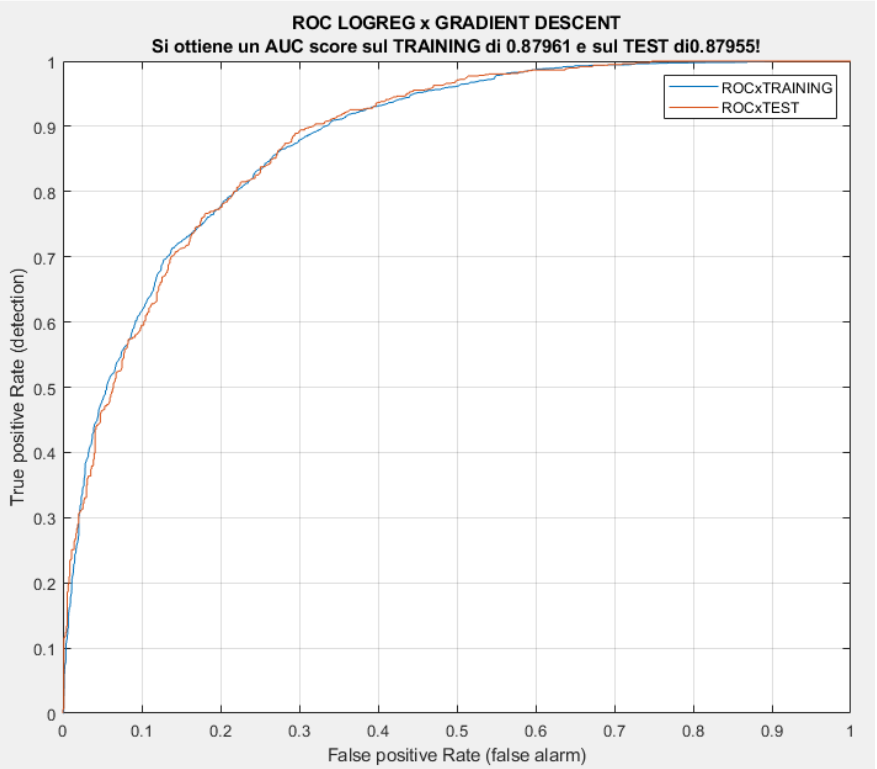
3.1 Gradient Descent

L'addestramento di Logistic Regression è stato effettuato in funzioni di step size differenti (con una politica di step size fissato durante l'esecuzione dell'algoritmo), in modo da osservare come cambia la velocità di decrescita della loss function verso il minimo e anche andando a vedere eventuali anomalie dovute da step size non adeguati. In questo caso l'insieme di step size in esame è composto dai seguenti valori: 10; 1; 10^{-1}; 10^{-2}; 10^{-3} (tutti *normalizzati* per il numero di osservazioni che compongono il training set).
Impostando un numero di iterazioni di aggiornamento pari a 2000 e una politica di "stop" all'aggiornamento con ϵ = 10^-4, si sono ottenuti i seguenti andamenti della funzione di costo



Si può osservare come non è sempre vero che avere uno step size importante garantisca una velocità di convergenza buona: questo è il caso in cui step size = 10. Si nota infatti come la funzione si avvicini al plateau di 3000, per poi risalire ed effettuare un *effetto rimbalzo* che non porta l'algoritmo a convergere. Si può anche evidenziare la netta differenza tra uno step size =1, che praticamente arriva in convergenza dopo pochissimi passi di esecuzione (eseguendo il codice si evince come la convergenza avvenga dopo circa 100 iterazioni), e uno step size di = 0.001, che nemmeno dopo 500 iterazioni si avvicina minimamente ai risultati ottenuti dagli altri.

Andando a valutare le prestazioni del classificatore, si è optato per utilizzare la metrica *Area Under the Curve*, una metrica che indica quanta area è sottesa dalla Receiver Operating Characteristic, si sono ottenuti come risultati sul training test di 0.8796 mentre sul test set di 0.8795. Ottimi risultati, praticamente uguali!. Di seguito viene riportato il grafico che mostra le ROC ottenute.

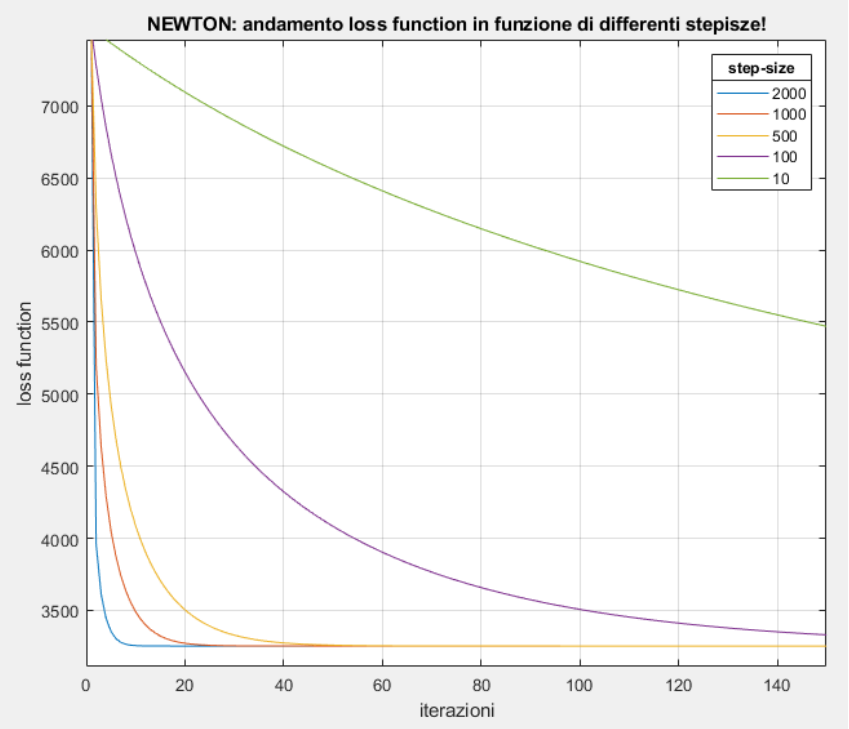


3.2 Newton Algorithm

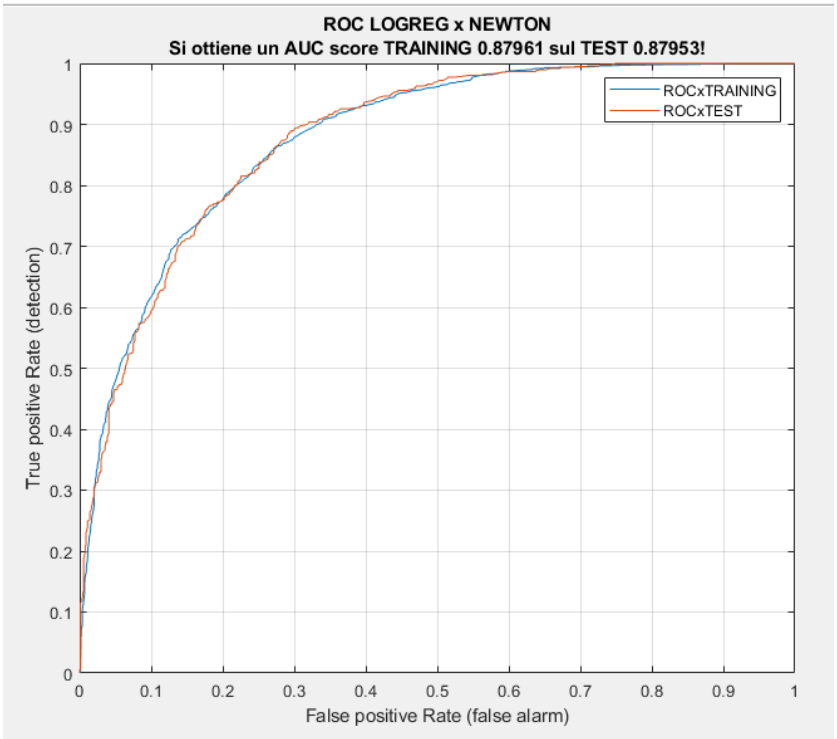
Le condizioni al contorno all'implementazione dell'algoritmo di Newton sono leggermente diverse, in quanto è vero che tale algoritmo impiega meno iterazioni di gradient descent per convergere alla soluzione ottima, però complessivamente è computazionalmente più costoso eseguirlo (soprattutto sull'hardware a disposizione per i test); quindi come esecuzione impiega molto più tempo di quello che non richieda gradient descent. Inoltre, per evitare problemi di non invertibilità della matrice hessiana, si aggiunge una regolarizzazione alla matrice, ovvero una matrice identica moltiplicata per un epsilon di circa 0.01. Gli step size considerati, sempre normalizzati per il numero di osservazioni del training set, sono: 2000; 1000; 500; 100; 10. Per quanto riguarda la politica di stop dell'algoritmo, si usa lo stesso epsilon usato per gradient descent

Il grafico, riportato sotto, mostra l'andamento della funzione obiettivo al variare degli step size (normalizzati). Si può notare come per lo step size 2000 si ottiene una funzione obiettivo che raggiunge il plateau dopo pochissimi step (eseguendo il codice si verifica che il plateau è raggiunto dopo circa 26 passi), mentre per tutti gli altri ci si mette proporzionalmente di più. Rispetto a gradient descent, non si arriva mai al limite massimo di iterazioni (come la teoria ci "garantiva") se

non per lo step di valore 10, che risulta essere così piccolo da non avvicinarsi nemmeno al minimo della funzione.



In termini di prestazioni di Logistic Regression, si ottengono due buone ROC sia sul training che sul test, con valori di AUC score molto simili che si attestano attorno a 0.87



3.3 ADMM distribuito

Implementare il training di Logistic Regression per mezzo di ADMM distribuito secondo i dati, ha bisogno di definire il parametro *numero agenti*, in modo da andare a simulare quanto è stato "distribuito" il calcolo, e il parametro di convessificazione del lagrangiano aumentato ρ. Per la tesina, si è scelto come numero di agenti 10, così che ogni sottoproblema di ottimizzazione avesse 750 osservazioni per ottimizzare il sottoproblema di Logistic Regression che si genera; mentre per i valori di convessificazione, si è scelto di far variare rho in un insieme discreto di valori: 0; 1; 10; 50; 100. Per quanto riguarda l'ottimizzazione delle variabili locali, visto che ad ogni step di ADMM si ha, per ogni agente, un "sotto addestramento di Logistic Regression", regolarizzato (con il termine regolarizzante ρ del lagrangiano aumentato), si è pensato di ottenere le singole soluzioni locali applicando gradient descent. Quindi ad ogni step e per ogni agente viene applicato l'algoritmo di gradient descent per ottenere la soluzione locale. Quindi si presume che il *consenso* venga raggiunto dopo un numero significativo di iterazioni
Dopo un numero di iterazioni di ADMM pari a 2000, si ottiene che le copie locali calcolate "virtualmente" su ogni agente sono molto vicine dall'essere uguali fra di loro.

	1	2	3	4	5	6	7	8	9	10
1	0.0125	0.0126	0.0126	0.0127	0.0126	-0.2910	0.0751	0.0127	0.0126	0.0126
2	1.9680	1.9680	1.9680	1.9680	1.9681	1.7648	2.3791	1.9681	1.9680	1.9680
3	-0.1368	-0.1369	-0.1369	-0.1369	-0.1369	-0.5123	0.0177	-0.1368	-0.1368	-0.1368
4	-0.0861	-0.0861	-0.0861	-0.0860	-0.0859	-0.5340	0.0824	-0.0860	-0.0860	-0.0860
5	-0.3343	-0.3342	-0.3342	-0.3342	-0.3341	-0.6230	-0.0040	-0.3342	-0.3341	-0.3342
6	0.2688	0.2689	0.2689	0.2689	0.2688	-0.1575	0.6006	0.2689	0.2688	0.2689
7	0.2842	0.2845	0.2845	0.2844	0.2844	-0.0496	0.6336	0.2845	0.2844	0.2844
8	-0.5604	-0.5604	-0.5605	-0.5605	-0.5604	-0.3955	-0.7167	-0.5604	-0.5605	-0.5604
9	-0.3425	-0.3422	-0.3423	-0.3423	-0.3423	-0.5899	-0.1528	-0.3422	-0.3423	-0.3423
10	-0.1126	-0.1125	-0.1125	-0.1125	-0.1125	-0.2503	0.0267	-0.1125	-0.1126	-0.1125

In figura sono mostrate le soluzioni locali ottenute su ogni agente.

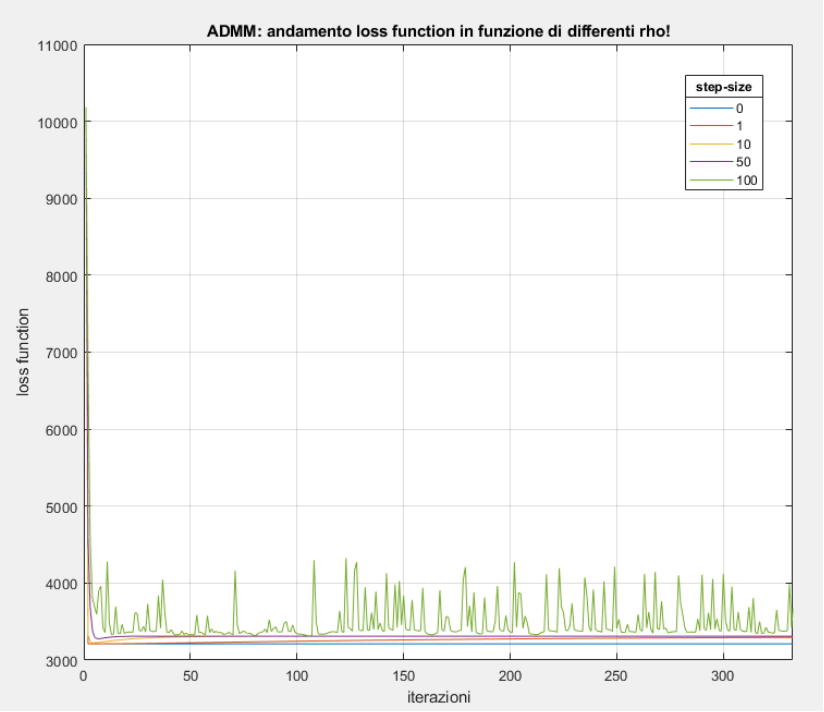
La differenza invece è più sensibile se si confrontano con il vettore di parametri ottenuti in precedenza con tecniche centralizzate.

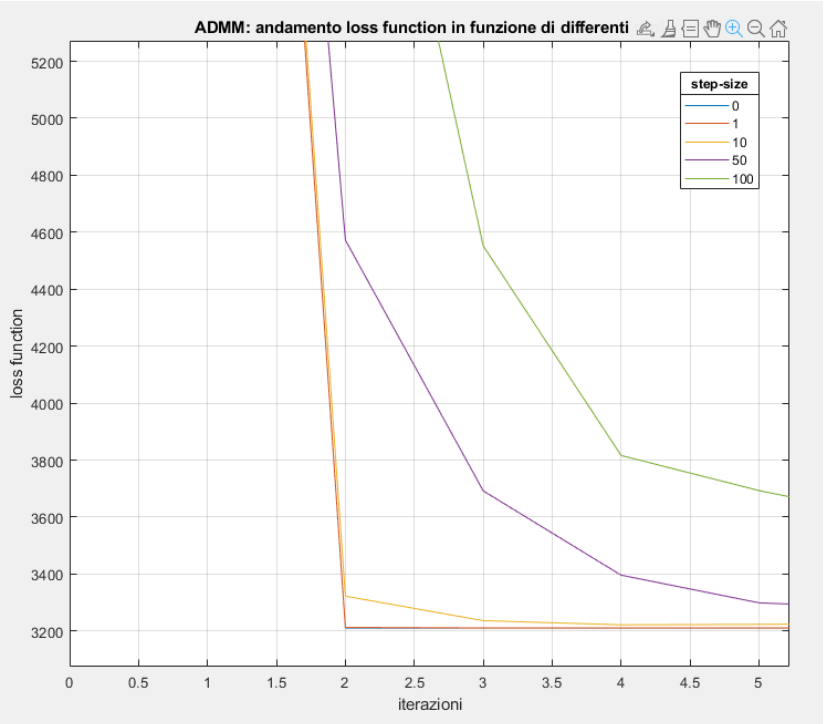
	1
1	-0.1255
2	1.8616
3	-0.2302
4	-0.2437
5	-0.2927
6	0.3142
7	0.1639
8	-0.5308
9	-0.2682
10	-0.2154

	1
1	-0.1268
2	1.8672
3	-0.2307
4	-0.2443
5	-0.2953
6	0.3130
7	0.1653
8	-0.5312
9	-0.2682
10	-0.2145

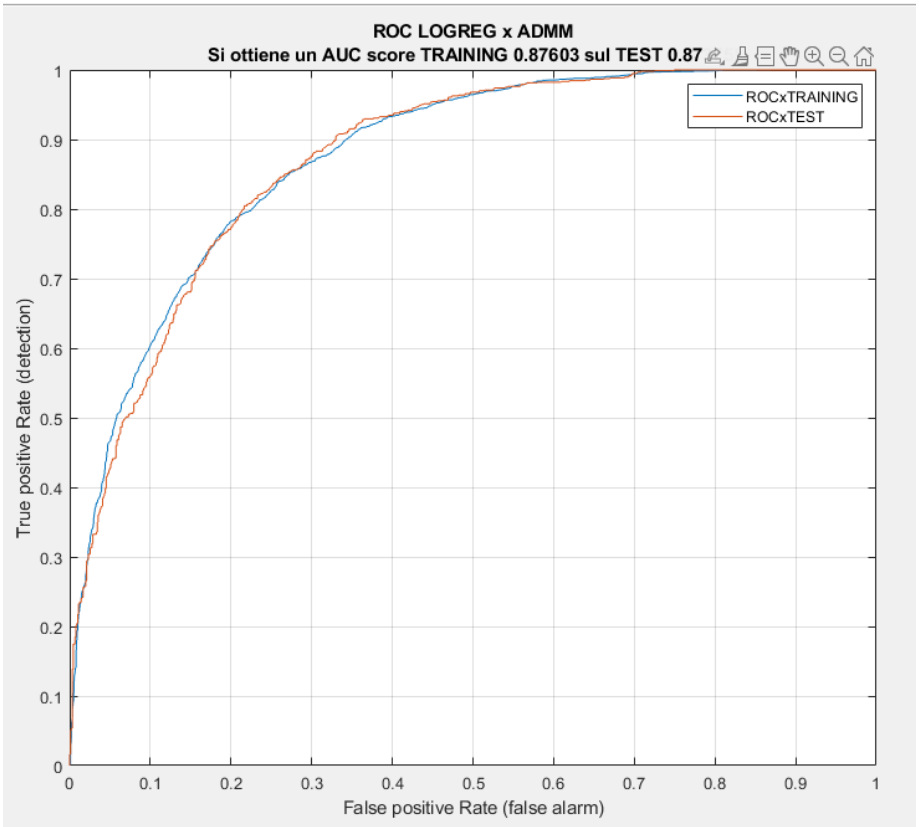
A sinistra il vettore di parametri ottenuto da gradient descent, a destra quello ottenuto dall'algoritmo di Newton

Per quanto riguarda l'andamento della funzione di costo **globale** (ovvero ottenuta sommando le singole funzioni distribuite) in funzione del parametro di convessificazione si evince che in realtà i migliori risultati si ottengono quando ρ è uguale a 0 (soprattutto dopo pochissimi step!). Questo non sorprende, in quanto in precedenza (nel capitolo **2.2.1**) si è dimostrato come la funzione costo soddisfa la convessità in senso stretto, e visto che il vincolo di uguaglianza *copie locali - variabile globale* è un vincolo affine, la funzione lagrangiana risulta essere strettamente convessa! Si può anche evidenziare come, al crescere del parametro ρ, l'andamento della funzione obiettivo peggiori significativamente.





Per quanto riguarda le prestazioni del classificatore ottenuto con questi parametri sono del tutto comparabili con quelli degli altri due addestramenti, infatti si ottiene un AUC score sia sul training che sul test di 0.87.



Extra: Trasformazione dati

In questa parte della tesina si trattano argomenti non del tutto inerenti al corso ma che nonostante ciò riguardano la preparazione del set di dati ad essere utilizzato per l'addestramento di un modello di machine learning.
Prima di tutto, va controllato se ci sono valori nulli; in questo caso erano presenti solo poche osservazioni con features nulle, pertanto si è provveduto a rimuovere le osservazioni incriminate.
Come in parte accennato nella parte di descrizione dei dati, ci sono alcune features delle nostre osservazioni che hanno bisogno di essere trasformate in variabili numeriche: queste erano **Gender**; **Smoking status**, **Residence** e infine **Work Type.** La feature "Gender" è stata trasformata in una variabile binaria che assume valore 1 se il paziente è una donna, 0 se invece il paziente è un uomo o oppure non è specificato (questo perchè nel dataset ci sono più donne che uomini). La feature "Smoking status" è una variabile categorica nominale, ovvero i valori che assume non hanno una connotazione "ordinabile", quindi fare un mapping nome = numero intero non è la cosa più corretta da fare, in quanto questo implicherebbe di dare più importanza ad un particolare valore di quella feature rispetto ad un altro. Anche in questo caso, si è deciso di applicare una trasformazione binaria della feature: assume valore 1 nel caso in cui il paziente sia un attuale fumatore oppure un ex fumatore, 0 nel caso questa informazione non sia disponibile oppure il paziente non ha mai fumato. La feature "Work Type" subisce un trattamento analogo, infatti dai valori assunti dalla feature inizialmente (ovvero il tipo di lavoro, o se non aveva lavorato), si ricava una feature binaria, che riassume con 1 se il paziente ha lavorato, 0 altrimenti. Infine la feature "Residence type" viene rimossa, in quanto risulta essere poco discriminativa in funzione della classificazione.
L'ultima parte di trasformazione dati riguarda un controllo sui range di valori assunti dalle variabili già numeriche. infatti può accadere di trovare dei valori che sono affetti da rumore e quindi diventino dei valori completamente assurdi. Questo è il caso che è capitato andando ad esplorare la feature **BMI;** alcune osservazioni riportano valori di BMI ben più grandi di 40, che risulta essere lo stadio di obesità più grave. toccando picchi di circa 90. Visto la considerazione fatta, si è scelto di rimuovere tutte le osservazioni con BMI maggiore di 60, giusto per fare una scelta conservativa.

Altra trasformazione effettuata ai dati riguarda un qualcosa intrinseco ai dati; infatti facendo un'analisi sulla distribuzione della variabile di uscita "stroke", è risultato che il dataset fosse molto sbilanciato rispetto al valore 0 (non predisposizione a ictus). Avere un dataset fortemente sbilanciato rispetto alla variabile di uscita può portare ad addestrare modelli ad alta varianza, quindi con scarse prestazioni.
Lavorando con un dataset non così grande, non si potevano rimuovere ulteriori osservazioni di quelle che già, per un motivo o per un altro, erano state già rimosse. Si è previsto quindi di utilizzare una tecnica di resampling, che permettesse di generare campioni sintetici per la classe meno rappresentata all'interno delle osservazioni. La tecnica utilizzata prende il nome di SMOTE (*Synthetic Minority Oversampling Technique*), che permette, utilizzando una variante del K-NN, di generare campioni sintetici in modo "automatico". I modelli di machine learning addestrati su dataset