

Attività progettuale di Intelligent Systems

Report Finale

Autore: Lorenzo Piazza
Matricola: 0000908824

Repository del progetto

<https://github.com/LorenzoPiazza/MachineLearning-case-study>

Sommario

Introduzione: contesto e obiettivi	3
1. Esplorazione, pulizia e visualizzazione del Dataset	4
1.1 Descrizione del dataset	4
1.2 Pulizia del dataset.....	5
1.3 Visualizzazione del dataset.....	5
2. Modelli predittivi	8
2.1 Linear Regression e Regression Tree per la predizione di <i>nTraces</i>	9
2.1.1 Tentativo di rafforzare i modelli con l'aggiunta delle feature Load e PV	12
2.2 Linear Regression e Regression Tree per la predizione di <i>sol(keuro)</i>	13
2.3 Multi-Layer Perceptron Regression con 3 input feature e target a rotazione	15
2.3.1 Feature Scaling	15
2.3.2 Simple Holdout con parametri di default: metodo e risultati	16
2.3.3 GridSearch Cross Validation: metodo e risultati	16
2.4 Multi-Layer Perceptron Regression con feature <i>nTraces</i> e target a rotazione	18
2.5 Multi-Layer Perceptron Regression con feature <i>nTraces-PV-Load</i> e target a rotazione.....	19
2.6 Embedding delle NN in Empirical Model Learning	20

Introduzione: contesto e obiettivi

Questa attività progettuale si contestualizza all'interno del topic *Algorithm runtime prediction*. Con questo nome ci si riferisce all'utilizzo dell'Intelligenza Artificiale, più precisamente del Machine Learning, per sviluppare modelli il cui scopo sia quello di predire dei parametri relativi all'esecuzione di un algoritmo su un'istanza di problema.

Detto in altri termini, l'*Algorithm runtime prediction* permette di fare predizioni riguardo a come potrà comportarsi un certo algoritmo nel risolvere un problema mai visto in precedenza. Il tutto senza che sia necessario eseguire l'algoritmo. Ampliando la visione, l'*Algorithm runtime prediction* può essere utile a diversi scopi. Ad esempio, può servire a scegliere tra una famiglia di algoritmi quello più performante e che meglio si adatta all'istanza di problema che abbiamo a mano (*algorithm selection*). Ancora, partendo dalle predizioni è possibile fare il *tuning* di un algoritmo, cioè trovare il setting dei suoi parametri ottimali prima dell'esecuzione e, solo dopo, eseguirlo con quel setting in modo da ottenere le migliori performance.

Nello specifico di questa attività progettuale, l'algoritmo indagato si chiama (*nome dell'algoritmo??*) e trova applicazione nella gestione energetica di una centrale elettrica. Si occupa di problemi in cui le variabili di incertezza sono il carico di energia richiesto dagli utilizzatori (in seguito indicato con il termine Load) e la quantità di energia rinnovabile disponibile nella centrale (in seguito indicata con PV). Prese in input queste quantità e fatte altre considerazioni, che non riporto in quanto esulano dagli scopi della mia attività, l'algoritmo è in grado di calcolare la quantità di energia strettamente necessaria che dovrà essere prodotta dalla centrale per sopperire al carico richiesto.

L'attività progettuale parte dall'esplorazione di un dataset in cui sono stati raccolti i dati relativi all'esecuzione dell'algoritmo su diverse istanze di problema. Prosegue poi nello studio di alcune tecniche di Machine Learning, precisamente modelli di Linear Regression, Regression Tree e Neural Networks, per fare predizioni riguardo a un certo *target* considerando diverse *feature*.

Come strumenti di lavoro ho sfruttato il linguaggio Python, in quanto mette a disposizione diverse librerie di Data Mining e Machine Learning, e l'ausilio di un Notebook Jupyter per raccogliere il codice e i commenti.

Questo documento ha lo scopo di riassumere l'intera attività progettuale, scomponendola nelle sue fasi salienti. Il documento, quindi, si presenta diviso in sezioni, una per ogni fase, che sono ordinate cronologicamente e che permettono al lettore di ripercorrere il mio lavoro. Per ogni step progettuale descritto, il focus vuole essere duplice: da un lato sulle ragioni che lo motivano, dall'altro sui risultati che sono stati ottenuti.

1. Esplorazione, pulizia e visualizzazione del Dataset

L'obiettivo di questa fase è quello di prendere confidenza con i dati che mi sono stati forniti, requisito fondamentale per agire con maggior consapevolezza negli step successivi. Ho esplorato il dataset per capire quanti record avessi a disposizione, da quanti e quali campi fossero composti e di che tipo fossero. È seguita poi una procedura di pulizia del dataset, spesso additata come necessaria nei testi di Data Mining per la possibile presenza di errori o dati mancanti. Infine, ho visualizzato i dati servendomi dell'aiuto visivo di diversi tipi di grafici.

1.1 Descrizione del dataset

Per creare il dataset sono state considerate cento diverse istanze di problema, ottenute dall'osservazione di un *Virtual Power Plant* (VPP). Su ciascuna istanza è stato fatto girare l'algoritmo di fixing per 100 volte, considerando ogni volta un numero diverso di tracce¹ (da 1 a 100).

Quindi il dataset risulta composto da **10000 record** (100run x 100istanze), dove nelle prime 100 righe avremo i dati relativi all'algoritmo di fixing fatto girare sulle 100 diverse istanze ma considerando 1 sola traccia. Poi nelle seconde 100 righe i risultati dell'algoritmo di fixing applicato sempre a ciascuna di quelle 100 istanze ma stavolta considerando 2 tracce, ecc.

Gli attributi più rilevanti al progetto sono i seguenti:

- **Load(kW):** vettore di 96 valori relativi alle osservazioni di carico campionate in 96 stage (ogni 15 minuti nell'arco di una giornata).
- **PV(kW):** vettore di 96 valori relativi alle osservazioni di energia fotovoltaica disponibile. Anche per PV le osservazioni sono state campionate in 96 stage (ogni 15 minuti nell'arco di una giornata).
- **Sol(keuro):** valore della soluzione relativa alla quantità di energia che la centrale deve produrre. Viene calcolata dall'algoritmo e ottenuta come somma su tutto l'orizzonte temporale (sui 96 stage) di tutte le 96 soluzioni ottenute dai running parziali.
- **Time(sec):** tempo necessario al runtime online dell'algoritmo. Anch'esso è ottenuto come somma su tutto l'orizzonte temporale (sui 96 stage) di tutte i 96 tempi di running parziali.
- **memAvg(MB):** rappresenta la memoria utilizzata dall'algoritmo sulla macchina su cui è stato eseguito. È una media della memoria usata sui 96 stage.

¹ Una traccia può essere definita come uno scenario, cioè un insieme dei valori assegnati alle variabili incerte -nel nostro caso *Load* e *PV*- che, una volta sottoposto ad un algoritmo anticipatorio, è stato arricchito con la sequenza di stati visitati e con le decisioni prese. Le tracce vengono quindi create in una fase *offline* e poi possono essere sfruttate *online* dall'algoritmo di fixing. Vengono usate come una collezione di soluzioni pre-calcolate utili a calcolare il valore di *sol(keuro)* in modo più efficiente.

1.2 Pulizia del dataset

Osservando le dimensioni del dataset fornito mi sono accorto che qualcosa non tornava rispetto a quanto mi aspettavo. Secondo quanto descritto in precedenza i record dovevano essere 10000 mentre il dataset fornito ne presentava 10099. Erano ripetuti infatti, ogni cento righe, gli header di intestazione delle colonne, giustificando così la presenza di 99 record aggiuntivi. È stato quindi necessario pulire il dataset rimuovendo le suddette righe.

1.3 Visualizzazione del dataset

Per poter impostare lo studio dei modelli predittivi sviluppato nelle fasi di progetto successive è stato fondamentale capire come fossero correlati tra loro i vari dati di cui disponevo. A questo scopo ho realizzato due *Correlation Matrix* che riporto in Figura 1.1 e 1.2:

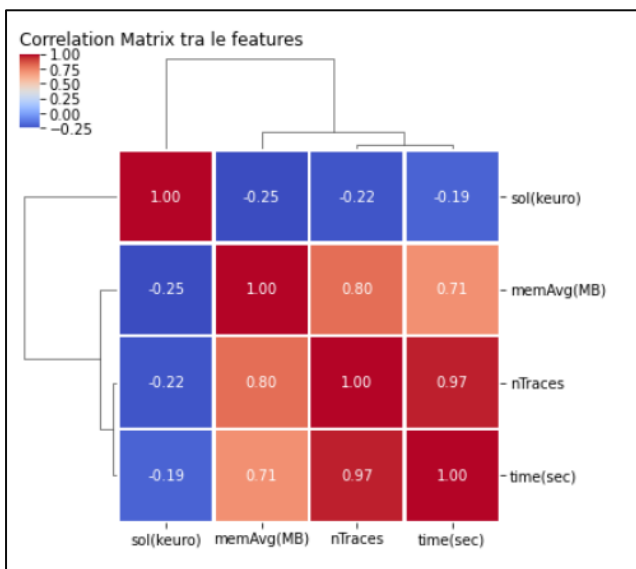


Figura 1.1 – Correlation Matrix tra le features

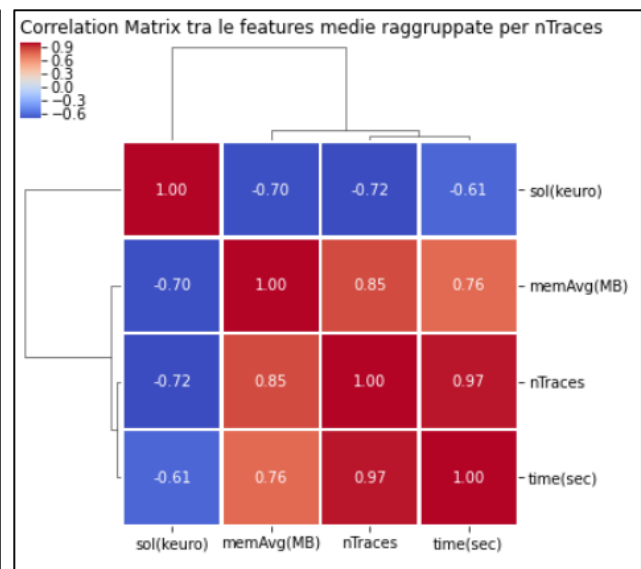


Figura 1.2 – Correlation Matrix tra le features medie raggruppate per nTraces

Dalla Figura 1.1 possiamo notare che:

- *nTraces* è fortemente correlato in primo luogo con *time* e a seguire con *memAvg*.
- *sol(keuro)* è scarsamente correlata con tutte le features considerate.

Diverso è il discorso se consideriamo i valori medi delle features raggruppate per numero di tracce. In figura 1.2 infatti notiamo che:

- *nTraces* è ancora fortemente correlato in primo luogo con *time* e poi con *memAvg*, ma ora ha una discreta correlazione inversa anche con *sol(keuro)*.
- aumenta la correlazione di *sol(keuro)* con le altre features considerate.

In seguito, ho provveduto ad analizzare più nello specifico le singole relazioni presenti tra i vari attributi. Per farlo ho realizzato due *pairplot* considerando, da un lato gli attributi con i valori per ogni singola istanza (Figura 1.3), dall'altro gli attributi con i valori medi ottenuti raggruppando le istanze per numero di tracce (Figura 1.4). Il *pairplot* con i valori medi è più indicativo di un andamento generale in quanto considera i dati aggregati e mediati di cento diverse istanze di problema. Tuttavia, anche il primo *pairplot* è utile poiché i modelli predittivi che ho sviluppato vengono allenati considerando le singole istanze e non i valori medi.

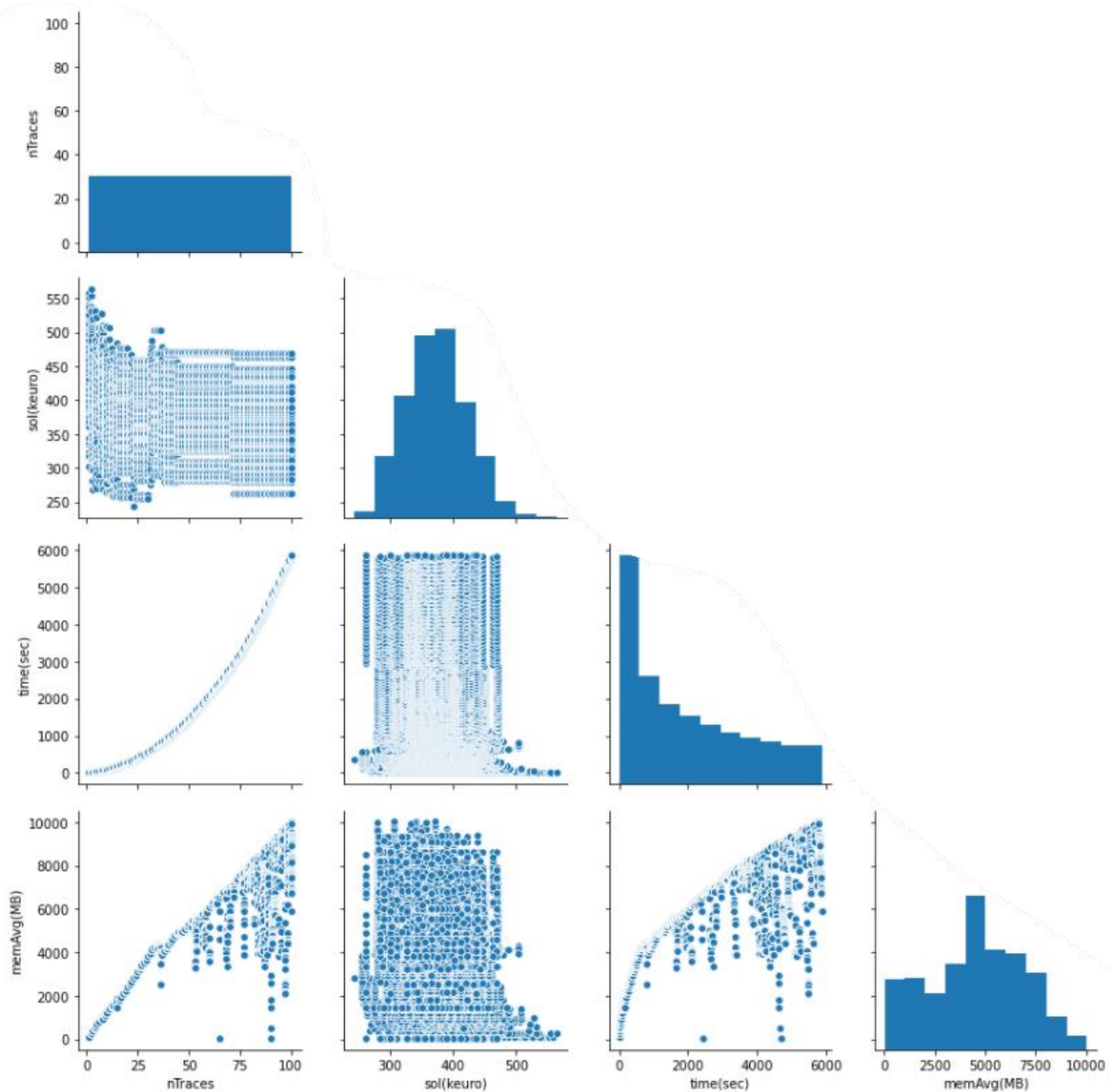


Figura 1.3 – Pairplot dei vari attributi con i valori per singola istanza.

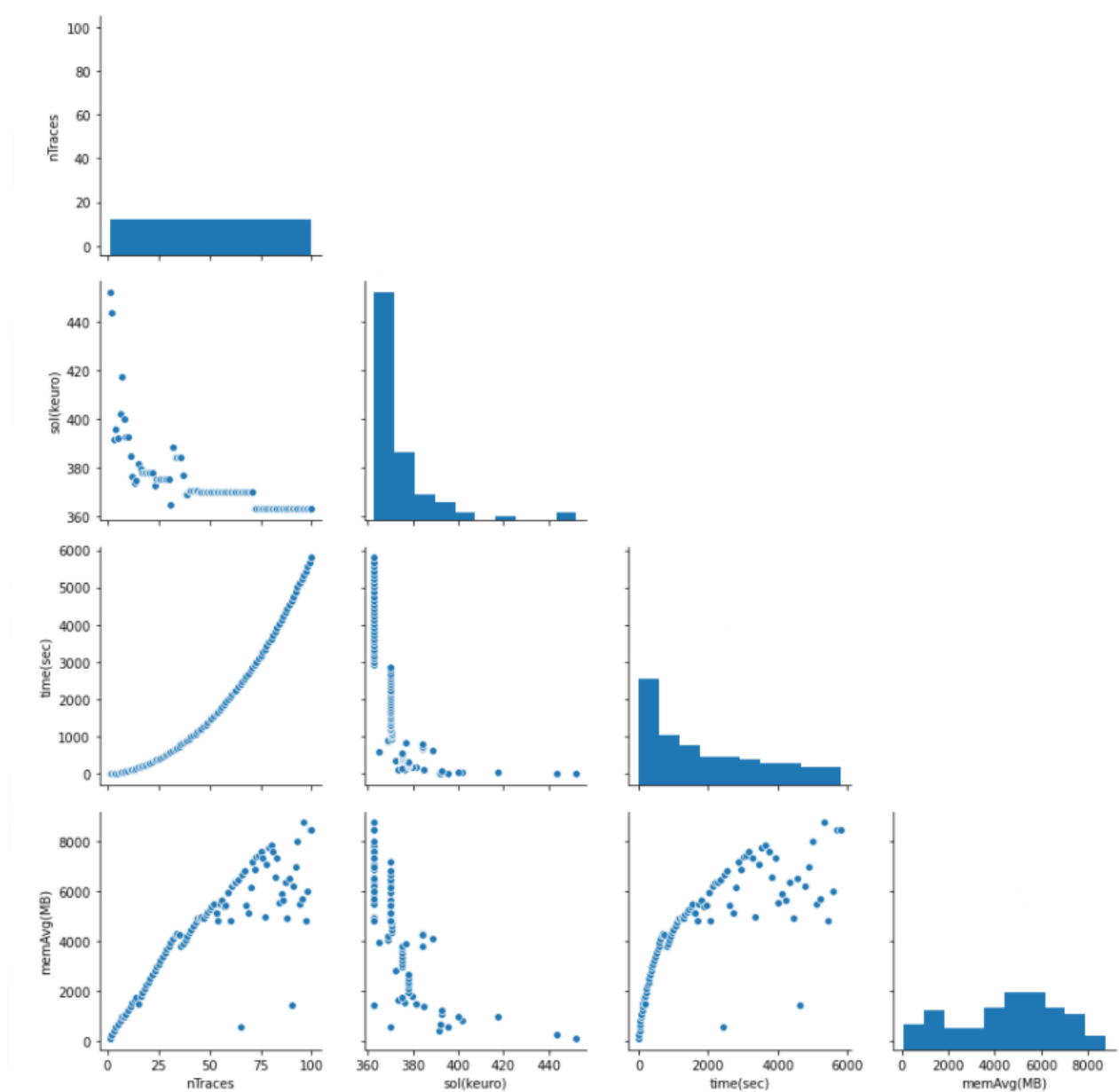


Figura 1.4 – Pairplot degli attributi con i valori medi delle istanze aggregati per numero di traccia.

Infine, per completare la fase di visualizzazione del dataset ho realizzato dei *boxplot*, utili a esplicitare la distribuzione dei valori vari attributi. Tuttavia, quest'ultime informazioni non si sono rivelate di grande importanza ai fini delle fasi successive. I *boxplot* sono riportati in Figura 1.5.

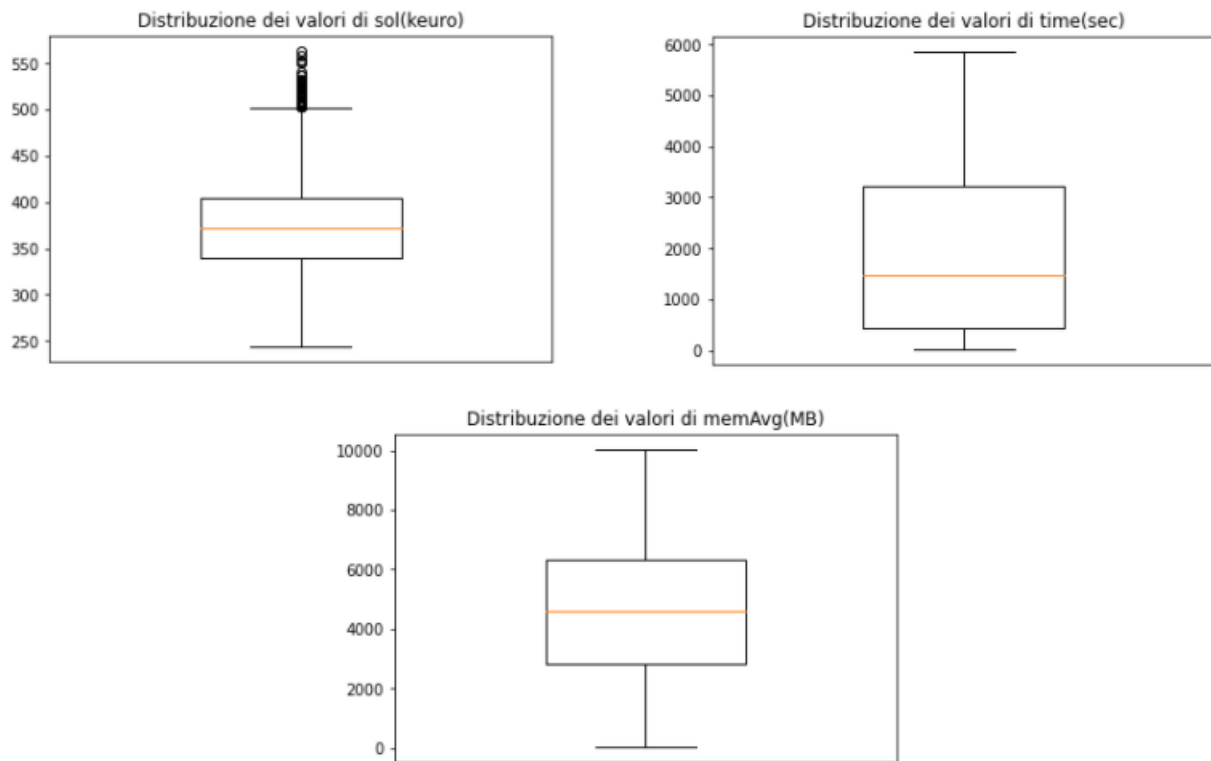


Figura 1.5 – Boxplot con le distribuzioni dei valori degli attributi *sol(keuro)*, *time(sec)* e *memAvg(MB)*.

2. Modelli predittivi

Come anticipato nella sezione di Introduzione, in questa seconda fase di progetto mi sono dedicato allo sviluppo di modelli di Machine Learning che fossero in grado di predire un certo attributo, detto *target*, fornendo in input i valori di altri attributi, detti *feature*. Dal momento che il target da predire era sempre un valore di tipo numerico ho utilizzato dei modelli di Regression. In particolare, le tecniche che ho utilizzato sono la Linear Regression, i Regression Tree e la Multi-Layer Perceptron Regression.

Le motivazioni che hanno portato allo studio di modelli predittivi sono di carattere pratico: supponiamo infatti di avere a disposizione un'istanza di problema sulla quale fare eseguire l'algoritmo; supponiamo inoltre di avere dei vincoli sull'esecuzione (ad esempio un tempo massimo entro cui l'algoritmo deve terminare, un limite di risorse che possiamo dedicarvi, ecc.). Sarebbe utile in queste situazioni poter predire il valore di una determinata variabile target prima che l'algoritmo esegua concretamente. Ad esempio, poter prevedere il valore della soluzione che verrà ottenuta sotto determinati vincoli imposti. Si possono fare tanti altri esempi di applicazioni dei modelli predittivi semplicemente elencando tutte le possibili combinazioni in cui, come target fissiamo l'attributo da predire, mentre come *feature* scegliamo uno (o più) attributi rappresentanti i vincoli di esecuzione. Nelle prossime sezioni vengono indagati alcuni di questi casi applicativi.

2.1 Linear Regression e Regression Tree per la predizione di *nTraces*

Come si evince dal titolo di questa sezione, lo scopo di questa fase è quello di sviluppare modelli predittivi con target *nTraces* e una o più feature a scelta tra *sol*, *time* e *memAvg*. Seguendo il ragionamento appena esposto, questi modelli sono utili a dirci in anticipo il numero di tracce con cui dover eseguire l'algoritmo laddove i vincoli, e quindi le feature da considerare per allenare il modello, potrebbero essere uno o più tra i seguenti:

- “desidero ottenere un certo valore di soluzione” → feature *sol(keuro)*.
- “ho un limite di tempo entro il quale l'algoritmo deve terminare” → feature *time(sec)*.
- “voglio dedicare all'algoritmo un certo quantitativo di memoria media” → feature *memAvg(MB)*.

In questa fase di progetto ho utilizzato le tecniche di Linear Regression e Regression Tree sfruttando la libreria Sklearn. Per ciascuna delle due tecniche ho sviluppato sette modelli predittivi, uno per ogni possibile combinazione di feature in input.

Il dataset è sempre stato diviso rispettivamente in Train Set, su cui allenare il modello, e Test Set, su cui invece misurarne le performance. Per evitare *overfitting* del modello e considerando che disponiamo di un cospicuo numero di record, la dimensione percentuale del Train Set non è stata scelta eccessivamente alta, bensì abbastanza equilibrata rispetto alla dimensione del Test Set (Train Set: 55%, Test Set: 45%).

Le performance dei vari modelli sviluppati sono state misurate sfruttando le metriche R^2 (“R squared”)² e MSE (“Mean Squared Error”)³.

In Figura 2.1 e 2.2 vengono mostrate le performance ottenute rispettivamente dai Linear Regressor e dai Regression Tree.

² R^2 è una metrica che rappresenta quanto il nostro modello si comporta meglio, in termini percentuali, rispetto ad un modello regressore dalle scarse prestazioni, che predice il target semplicemente come media dei valori su cui è stato allenato. Il valore R^2 appartiene all'intervallo $[0,1]$. Detto \hat{y}_i l'i-esimo valore predetto dal modello, y_i il corrispondente true value e $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ il valor medio, R^2 si calcola nel seguente modo:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

³ MSE è una metrica che rappresenta la media degli errori di predizione, elevati al quadrato per evitare che errori negativi bilancino quelli positivi. Detto \hat{y}_i l'i-esimo valore predetto dal modello e y_i il corrispondente true value, MSE si calcola con la seguente formula:

$$MSE(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Modelli di Linear Regression - target nTraces

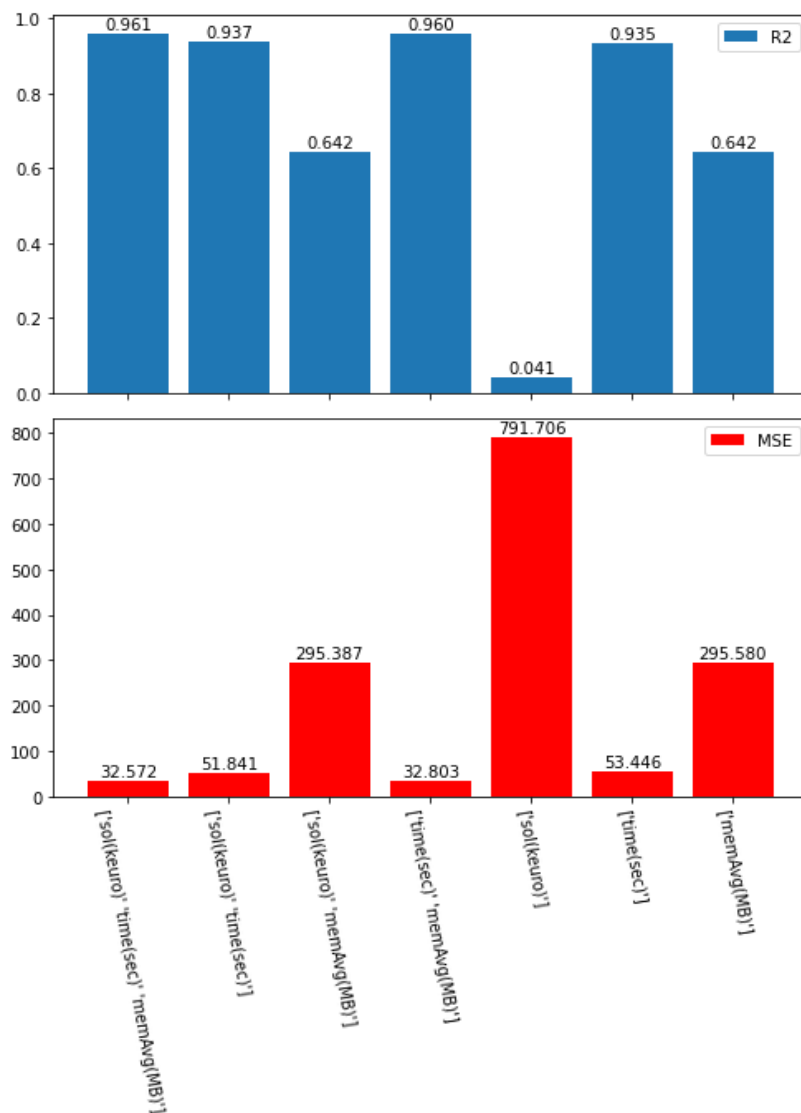


Figura 2.1 – Performance ottenute dai modelli sviluppati con la Linear Regression.

I risultati ottenuti rispecchiano le considerazioni fatte sulle *correlation Matrix* mostrate in Figura 1.1, dove risultava che le feature maggiormente correlate con nTraces fossero *time(sec)* e *memAvg(MB)*. Non ci sorprende dunque il fatto che i risultati di predizione migliori sono stati ottenuti quando il modello viene allenato considerandole entrambe. Al secondo posto nella classifica delle performance troviamo i risultati di predizione ottenuti considerando la feature *time(sec)* ed al terzo quelli ottenuti considerando *memAvg(MB)*.

In questa classifica non è stata volutamente citata *sol(keuro)* in quanto possiamo notare che la sua presenza tra le feature allenanti il modello è pressoché irrilevante in termini di performance.

Questo comportamento è giustificato dal fatto che *sol(keuro)* è scarsamente correlata con nTraces e quindi non è molto utile alla predizione del target. La R2 sul modello di linear regression in cui considero solo *sol(keuro)* infatti è parecchio scarsa (circa 4%).

Modelli di Regression Tree - target nTraces

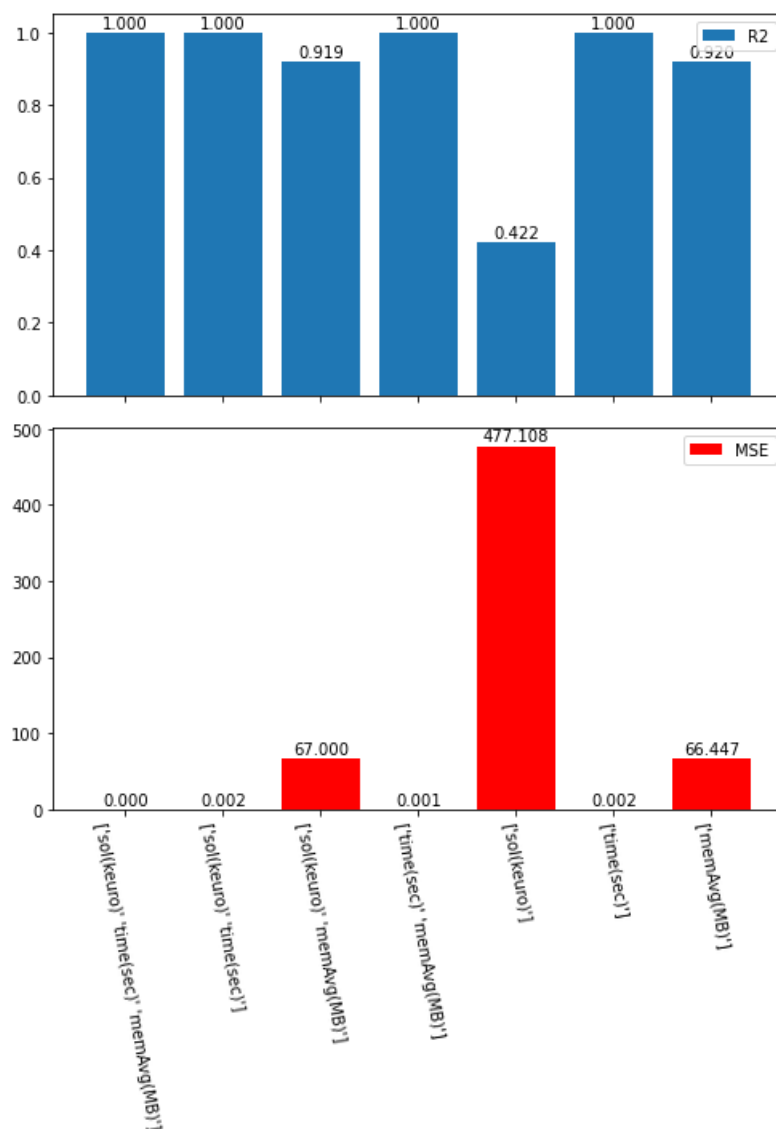


Figura 2.2 – Performance ottenute dai modelli sviluppati con la tecnica di Regression Tree.

Dalla Figura 2.2 possiamo invece notare che:

- I risultati ottenuti con i modelli di Regression Tree sono migliori di quelli ottenuti dalla Linear Regression. Questo è giustificabile in quanto, come visualizzabile dai pairplot in Figura 1.3, le feature non presentano tra loro delle relazioni lineari ed i Regression Tree sono più abili ad approssimare relazioni non lineari rispetto ai modelli di Linear Regression.
- Parecchi modelli qui allenati hanno un R2 prossimo a 1 e un MSE prossimo a 0.
- Sebbene la R2 del modello allenato con la feature sol(keuro) sia migliorata (ora è al 42%) vale ancora la stessa osservazione fatta per la linear Regression: la sua presenza/assenza non inficia significativamente sulle performance predittive ottenute con le altre feature.

2.1.1 Tentativo di rafforzare i modelli con l'aggiunta delle feature Load e PV

Visti gli scarsi risultati di predizione del target nTraces ottenuti dai modelli che utilizzavano la sola feature sol è stato fatto un tentativo di migliorarli con l'aggiunta delle feature Load(kW) e PV(kW), trattate come medie aritmetiche dei 96 valori.

Tuttavia, dai nuovi modelli non solo non sono state ottenute metriche migliori, ma nel caso della Regression Tree sono stati ottenuti risultati addirittura peggiori di quelli di partenza.

Ho concluso quindi che le feature Load(kW) e PV(kW) non possono essere utilizzate per predire nTraces in quanto quest'ultimo è totalmente indipendente da esse.

Probabilmente non era necessario sperimentarlo per trarre questa conclusione. Sarebbe stato sufficiente fare un'osservazione di carattere teorico notando che gli stessi valori di Load e PV si ripetono periodicamente per ogni valore di nTraces, di conseguenza non possono rappresentare un carattere utile a predire il target.

Esempio: Dati x, valor medio di 96 valori di PV, e y, valor medio di 96 valori di Load, sappiamo predire nTraces? No, in quanto ritrovando lo stesso x e la stessa y in tutti i valori di nTraces, l'entropia è massima (i valori assumibili da nTraces sono tutti equiprobabili).

Tuttavia, a questa conclusione vi sono arrivato solo una volta eseguito l'esperimento, ragionando sui risultati ottenuti.

Vista la poca rilevanza di questa fase di progetto, non ho ritenuto necessario riportare in questo documento i suoi risultati tabellari. Per un eventuale consulto si può fare riferimento alla sezione 3.1.3 del notebook con il codice del progetto, il quale è accessibile pubblicamente su Github (link in copertina).

2.2 Linear Regression e Regression Tree per la predizione di *sol(keuro)*

Analogamente a quanto compiuto in sezione 2.1, in questa fase di progetto è stato ripetuto l'iter procedurale fissando però come target l'attributo *sol(keuro)*. Come feature allenanti i modelli si sono considerate tutte le possibili combinazioni derivanti dalla scelta tra *nTraces*, *time* e *memAvg*.

L'applicazione pratica dei modelli sviluppati in questa fase diventa quindi quella di fornire in anticipo il valore di soluzione che ci aspettiamo a seconda della presenza di uno o più tra i seguenti vincoli:

- “l'algoritmo ha a disposizione un certo numero di tracce” → feature *nTraces*.
- “ho un limite di tempo entro il quale l'algoritmo deve terminare” → feature *time(sec)*.
- “voglio dedicare all'algoritmo un certo quantitativo di memoria media” → feature *memAvg(MB)*.

In Figura 2.3 e 2.4 vengono mostrate le performance ottenute rispettivamente dai modelli di Linear Regression e dai Regression Tree.

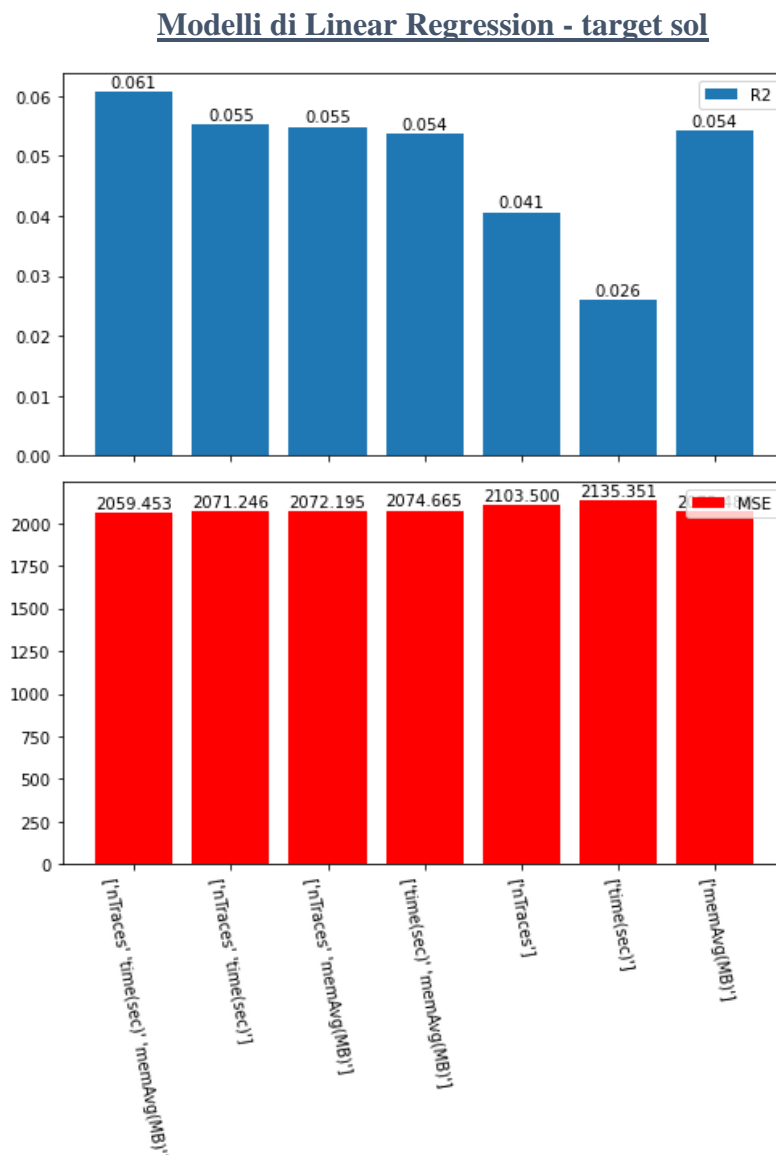


Figura 2.3 – Performance ottenute dai modelli sviluppati con la Linear Regression.

Modelli di Regression Tree - target sol

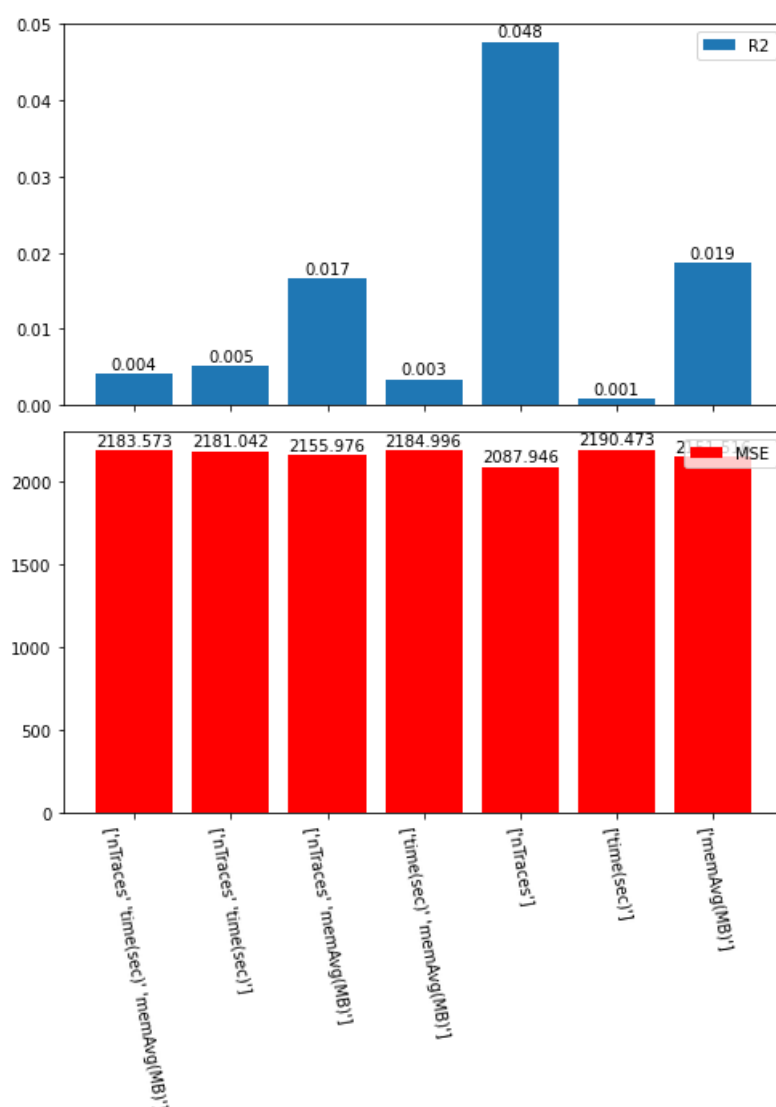


Figura 2.4 – Performance ottenute dai modelli sviluppati con la tecnica di Regression Tree.

Dalle metriche mostrate nelle ultime due Figure si nota come le performance di predizione del target soluzione siano parecchio scarse, sia per i modelli di Linear Regression che per i modelli di Regression Tree. Ricordando quanto evidenziato dalla correlation matrix in Figura 1.1, i risultati sono giustificabili dalla scarsa correlazione che l'attributo *soluzione* ha con le altre feature prese in esame (*nTraces*, *time* e *memAvg*).

2.3 Multi-Layer Perceptron Regression con 3 input feature e target a rotazione

In questa fase di progetto il focus del mio studio si è spostato sullo sviluppo di Neural Networks (NN) per la predizione di un certo attributo target, fornendo in input i valori di altre tre feature.

A questo fine sono stati presi in considerazione gli attributi ['nTraces', 'sol(keuro)', 'time(sec)', 'memAvg(MB)'] dei quali uno, a rotazione, viene considerato come target.

Rispetto ai modelli indagati nelle sezioni precedenti quindi, in questa sezione vengono considerati come target anche time e memAvg, che prima erano limitati ad essere feature in input.

Il modello di Neural Network che utilizzo è il Multi-Layer Perceptron (MLP) Regressor, messo a disposizione dalla libreria Sklearn.

Prima di iniziare lo sviluppo delle reti neurali è stato applicato un processo di feature scaling che descrivo nel paragrafo 2.3.1.

Le reti neurali poi, sono state allenate adottando due diversi metodi che per il momento mi limito a citare:

- Simple Holdout con iper-parametri di default.
- Tuning degli iper-parametri con la GridSearchCV.

I paragrafi 2.3.2 e 2.3.3 sono dedicati a descrivere questi due metodi e i risultati ottenuti.

2.3.1 Feature Scaling

Ciascuna NN sviluppata è stata allenata considerando tre differenti versioni dello stesso dataset:

- dataset con i valori originali.
- dataset con i valori standardizzati (normalizzazione Z-score).
- dataset con i valori normalizzati (normalizzazione Min-Max scaling).

La [normalizzazione Z-Score](#) viene calcolata come $z = (x - u) / s$,

dove u è la media dei campioni, e s è la deviazione standard.

Essa non produce dati in un range unitario ma li riconduce ad una distribuzione di media 0 e deviazione standard 1.

La [normalizzazione min-max scaling](#) viene calcolata come $z = (x - \min(x)) / (\max(x) - \min(x))$

Produce dati in un range unitario $[0,1]$.

Queste tecniche di feature scaling (ridimensionamento dei dati) sono molto utili per uniformare i valori delle varie feature evitando in questo modo che una feature possa dominare sulle altre ed incidere maggiormente sul training della rete solo perché presenta valori più grandi.

2.3.2 Simple Holdout con parametri di default: metodo e risultati

Simple Holdout è il classico metodo di allenamento e validazione del modello con divisione del dataset in Train Set + Test Set. I modelli di MLP regressor qui sviluppati sono stati allenati sul Train Set e poi valutati sul Test Set.

In questa fase ho usato i parametri di default del MLP regressor, consultabili sulla documentazione ufficiale⁴ di Sklearn.

Seguono, riportati in Tabella 2.1, i risultati delle metriche R2 ottenute dalle varie NN.

Tipo di Dataset Target	Original	Standardized	Normalized
nTraces	0.993	0.999	0.997
sol(keuro)	-1.025	0.085	0.054
time(sec)	0.979	0.999	0.995
memAvg(MB)	0.674	0.741	0.706

Tabella 2.1 – Valori di R2 dei MLP regressor allenati con il metodo Simple Holdout..

Possiamo notare come i risultati migliori siano stati ottenuti dal MLP allenato sul dataset con i valori standardizzati. A seguire MLP allenato sul dataset con i valori normalizzati ed infine MLP allenato sul dataset con i valori originali.

Come già osservato nei paragrafi precedenti, i target che possono essere predetti con maggior precisione sono nTraces e time.

La predizione del target sol(keuro) restituisce ancora una volta scarse prestazioni.

2.3.3 GridSearch Cross Validation: metodo e risultati

In questa sezione ho fatto il tuning di alcuni parametri del modello MLP con l'obiettivo di trovare quelli che garantiscono le migliori performance predittive (secondo la scoring function R2).

La GridSearchCV è una classe Python fornita dalla libreria Sklearn che permette di eseguire, per un modello predittore, una ricerca esaustiva tra tutte le possibili combinazioni dei parametri forniti (una cosiddetta ParameterGrid). Quando viene fatta la *fit* sul dataset, infatti, la GridSearch valuta le performance dei modelli ottenuti con tutte le possibili combinazioni di parametri. La combinazione migliore viene poi scelta per creare il modello finale su cui poter invocare la funzione *predict*.

⁴ Scikit learn doc, *MLP Regressor*,

URL: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor)

[learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor).

URL visitato il 02/01/2021.

I parametri che ho deciso di testare sono:

- *activation*: è l'activation function utilizzata negli hidden layer.
- *hidden_layer_sizes*: Il numero di hidden layer e, per ciascuno, la sua dimensione in numero di neuroni. A default si ha 1 hidden layer di 100 neuroni.
- *max_iteration*: numero massimo di iterazioni a cui, anche se non si è raggiunta un'ottimizzazione, il training della rete si stoppa.
- *tol*: fattore di tolleranza per l'ottimizzazione. Se per più di *n_iter_no_change* (default = 10) iterazioni consecutive la scoring function non migliora di almeno *tol*, l'ottimizzazione si considera raggiunta e il training della rete si stoppa.

La *fit* sul dataset viene effettuata sfruttando una KFold Cross-Validation (con K=5). Il dataset cioè viene diviso in 5 parti (Fold) e viene eseguito il Train su K-1 di questi e il test sul restante. La procedura viene eseguita K volte cambiando ogni volta il Fold designato come Test Set. Infine, viene fatta una media della metrica calcolata ad ogni iterazione.

A fronte di un maggior costo computazionale, la Cross Validation porta i seguenti vantaggi:

- permette di allenare il modello utilizzando tutti i dati del dataset.
- permette di ottenere una stima della performance che, essendo basata su una media di K iterazioni, è più affidabile.
- tutte le istanze del dataset vengono usate una volta per il testing.

Seguono, riportati in Tabella 2.2, i risultati delle metriche R2 ottenute dalle varie NN.

Tipo di Dataset Target	Original	Standardized	Normalized
nTraces	0.99326	0.99991	0.99988
sol(keuro)	0.04689	0.09834	0.09058
time(sec)	0.98325	0.99991	0.99990
memAvg(MB)	0.74574	0.91531	0.76334

Tabella 2.2 – Valori di R2 dei MLP regressor allenati con la miglior combinazione di parametri ottenuta dalla GridSearchCV. Per visualizzare le migliori combinazioni si faccia riferimento al notebook con il codice del progetto.

I risultati ottenuti dai modelli MLP con la miglior combinazione di parametri trovata dalla GridSearchCV evidenziano che le performance generali sono molto buone (sempre eccezion fatta per i modelli con target sol). Tuttavia, sono solo leggermente migliori delle performance ottenute dai MLP regressor allenati con i parametri di default in sezione 2.3.2.

L'unica prestazione che è migliorata significativamente è la predizione di memAvg(MB) nel dataset con i valori standardizzati. Lo score R2 migliora di circa 17 punti percentuali arrivando a segnare un punteggio di 91.5%.

2.4 Multi-Layer Perceptron Regression con feature *nTraces* e target a rotazione

Lo scopo di questa fase di progetto è quello di realizzare tre modelli di Multi Layer Perceptron Regressor rispettivamente con *nTraces* feature fissa e target a rotazione tra *time*, *memAvg* e *sol*. Rispetto ai paragrafi precedenti dove si consideravano tre input feature alla volta, l'obiettivo qui è isolare la feature *nTraces* e vedere come si comporta ai fini predittivi.

Riassumendo, le NN che sono state sviluppate sono le seguenti:

1. feature: *nTraces*, target: *time*.
2. feature: *nTraces*, target: *memAvg*.
3. feature: *nTraces*, target: *sol*.

Dal momento che eseguire la GridSearch non aveva portato miglioramenti significativi, in questa e nella successiva fase sono state sviluppate NN con parametri scelti arbitrariamente: *activation='relu'*, *hidden_layer_sizes=(100, 100, 100, 100)*, *max_iter=500*, *tol=0.0001*. È stata mantenuta invece la KFold Cross-Validation (con *K=5*).

Inoltre, gli stessi modelli sono stati allenati e testati sia sul dataset con valori originali che sul dataset con valori standardizzati. Da quanto sottolineato negli scorsi due paragrafi, infatti, le NN allenate con i valori standardizzati garantivano performance migliori (soprattutto per il target *memAvg*).

Per misurare le performance dei nostri modelli sono state prese in considerazione tre metriche. Oltre a *R2* e *MSE*, già esaminate nelle fasi precedenti, è stata valutata anche la metrica *MAE* ("Mean Absolute Error")⁵.

I risultati ottenuti sono riportati qui di seguito in tabella 2.3:

NN con feature *nTraces* (4 Hidden layer da 100 neuroni)

Data Type Target to predict	R2		MSE		MAE	
	original	standardized	original	standardized	original	standardized
memAvg(MB)	0.710879	0.826751	1.567541e+06	0.173278	699.044479	0.224230
sol(keuro)	0.071948	0.073336	2.070719e+03	0.925208	36.923603	0.781482
time(sec)	0.999613	0.999832	1.133966e+03	0.000168	20.981788	0.009215

Tabella 2.3 – Metriche ottenute dai MLP regressor.

⁵ Similmente alla metrica *MSE* (vedi nota 3) *MAE* rappresenta la media degli errori di predizione, che questa volta non sono elevati al quadrato ma solo presi in valore assoluto. Viene calcolato con la seguente formula:

$$MAE(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|.$$

2.5 Multi-Layer Perceptron Regression con feature *nTraces-PV-Load* e target a rotazione

In questa fase di progetto ho ripetuto interamente l'iter procedurale descritto nel paragrafo 2.4, arricchendo però la feature *nTraces* con i valori di input Load e PV.

Le motivazioni alla base di questa scelta risiedono nelle scarse performance che fin qui avevo ottenuto per la predizione della soluzione. L'obiettivo di questa fase è quindi migliorarle sfruttando i valori di Load e PV da cui essa dipende fortemente.

Lo scenario pratico che si vuole perseguire è *"se l'algoritmo dovesse considerare un certo numero di tracce e, considerati i valori di PV e Load osservati, quanta energia elettrica si prevede che dovrà essere prodotta dalla centrale?"*

Come già detto in precedenza, PV e Load sono vettori di 96 elementi. Dal momento che per il training della rete neurale dobbiamo trattare un valore singolo, ho utilizzato le medie aritmetiche dei 96 valori.

I risultati ottenuti sono riportati qui di seguito in tabella 2.4:

NN con feature *nTraces-Load-PV* (4 Hidden layer da 100 neuroni)

Data Type Target to predict	R2		MSE		MAE	
	original	standardized	original	standardized	original	standardized
memAvg(MB)	0.701870	0.888496	1.616579e+06	0.111649	734.217220	0.173034
sol(keuro)	0.116707	0.907168	1.971699e+03	0.092823	36.066087	0.202641
time(sec)	0.999184	0.999934	2.402534e+03	0.000066	31.900137	0.006222

Tabella 2.4 – Metriche ottenute dai MLP regressor.

Rispetto alla tabella 2.3, avendo ora arricchito le feature con i dati di Load e PV medi, possiamo notare che R2 migliora o rimane all'incirca stabile per tutti i modelli considerati.

Come ci aspettavamo, il miglioramento più significativo è osservabile nel valore di R2 ottenuto per il target sol(keuro) con i dati standardizzati, che raggiunge finalmente valori notevoli: 90.7%. Si noti come il feature scaling (in questo caso la standardizzazione) si sia rivelato fondamentale per raggiungere questo traguardo.

Possiamo quindi concludere che Load e PV sono feature indispensabili per la predizione del target soluzione.

2.6 Embedding delle NN in Empirical Model Learning

In questa ultima fase di progetto mi sono limitato ad eseguire alcune richieste finalizzate a valutare un possibile inserimento delle NN che ho sviluppato all'interno di un EML⁶ sviluppato in un differente progetto. (???)

Dal momento che EML supporta l'embedding di NN con un massimo 4 hidden layer da 10 neuroni ciascuno, mi è stato chiesto di rieseguire le fasi di progetto 2.4 e 2.5 con questa modifica alle NN e valutare come cambiassero le performance.

Se nelle precedenti fasi le NN sviluppate presentavano 4 hidden layer da 100 neuroni ciascuna, ecco i risultati ottenuti dalle nuove NN con 4 hidden layer da 10 neuroni ciascuna.

NN con feature nTraces-Load-PV (4 Hidden layer da 10 neuroni)

Data Type	R2		MSE		MAE	
	original	standardized	original	standardized	original	standardized
Target to predict						
memAvg(MB)	0.672747	0.746857	1.777273e+06	0.253085	895.917936	0.272000
sol(keuro)	0.066964	0.075869	2.081971e+03	0.922676	37.068502	0.780562
time(sec)	0.999803	0.999834	5.795610e+02	0.000166	18.248133	0.009472

Tabella 2.5 – Metriche ottenute dai MLP regressor.

NN con feature nTraces-Load-PV (4 Hidden layer da 10 neuroni)

Data Type	R2		MSE		MAE	
	original	standardized	original	standardized	original	standardized
Target to predict						
memAvg(MB)	0.663705	0.766250	1.821030e+06	0.233639	895.332041	0.262009
sol(keuro)	0.111026	0.596505	1.983133e+03	0.402826	36.095508	0.482324
time(sec)	0.999090	0.998963	2.676656e+03	0.001037	36.047142	0.024849

Tabella 2.6 – Metriche ottenute dai MLP regressor.

Confrontando le metriche R2 di Tabella 2.5 e 2.6 rispettivamente con quelle di Tabella 2.3 e 2.4, notiamo come le performance siano ora peggiorate per la predizione di memAvg e di sol.

Non variano invece le performance per la predizione del target time che rimangono assestate su valori R2 del 99%.

⁶ <https://emlopt.github.io/>