



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

Corso di Laurea Magistrale in Informatica

Diego Rando (1000027105)

**Regressione mediante algoritmi di Machine Learning
utilizzando python e pyTorch**

Anno Accademico 2020 – 2021

Indice

Indice	1
Introduzione	2
Capitolo 1 – Descrizione del problema e del dataset	3
1.1 – Quale problema vogliamo risolvere?	3
1.2 – Descrizione del dataset e composizione dei dati	3
1.2.1 – Modifica e miglioramento delle features	4
1.2.2 – Divisione del dataset e visualizzazione dei dati	6
Capitolo 2 – Tecniche utilizzate ed esperimenti effettuati	7
2.1 – Metodi e soluzioni proposte con pytorch	8
2.1.1 – Algoritmi di Valutazione	9
2.1.2 – Regressione Lineare	11
2.1.3 – MLP – Multi-layer Perceptron	15
2.1.4 – SVM – Support Vector Machine	17
2.1.5 – Altri esperimenti effettuati	20
Capitolo 3 – Composizione del main	22
Capitolo 4 – Conclusioni	23
4.1 – Valutazioni finali	23
4.2 – Miglioramenti effettuati e possibili miglioramenti	24

INTRODUZIONE

L'obiettivo di questo elaborato è quello di effettuare regressione e quindi previsione, mediante algoritmi di machine learning, tramite l'utilizzo del linguaggio di programmazione python ed in particolare usufruendo della libreria pytorch molto utilizzata nel campo del deep learning.

I dati in campo medico-sanitario appartenenti al dataset, fornito dall'azienda Digitizers S.R.L, sono stati fondamentali per lo sviluppo di tale progetto.

Sono state utilizzate diverse reti neurali che per definizione sono modelli computazionali composti da neuroni artificiali che cercano di emulare quello che è il comportamento delle reti neurali umane. Per alcuni casi sono state adottate reti neurali già solide con qualche modifica per adattarle all'obiettivo prefissato, in altri casi sono state costruite da zero.

CAPITOLO 1 - Descrizione del problema e del dataset

1.1 Quale problema vogliamo risolvere?

La domanda che ci si è posti in questo elaborato è la seguente: è possibile prevedere il numero di dimissioni all'interno di un ospedale o di una clinica, tramite l'utilizzo di algoritmi di previsione?

Ottenere una risposta positiva a questa domanda potrebbe essere d'aiuto in termini di risparmio delle risorse, per tutte le aziende del settore medico-sanitario, in quanto per esempio, potrebbero essere occupate un numero di stanze minore e quindi un vantaggio sia economico in quanto si hanno costi di mantenimento delle stanze minore e sia in termini di spazi in quanto verrebbero occupate meno stanze.

1.2 Descrizione del dataset e composizione dei dati

I dati che costituiscono il dataset vengono estratti da un database locale che contiene dati sanitari per un totale di 10 anni, non continui, forniti dall'azienda Digitizers S.R.L. Vengono estratti tramite una query con la seguente forma:

```
sql="SELECT dataDimissioneMorte FROM ricoveri"
```

in cui vengono prelevati tutti i dati appartenenti alla colonna “dataDimissioneMorte” dalla tabella ricoveri.

Questi dati sono ancora grezzi e devono essere trattati per essere utilizzati in quanto hanno la seguente forma:

```
[“01-01-2010”, “01-01-2010”, “01-01-2010”, “01-01-2010”, “01-01-2010” ]
```

Mentre il risultato per una facile gestione dei dati che si vuole ottenere è del tipo:

```
[“01-01-2010”, 5]
```

quindi coppie in cui il primo elemento rappresenta la data (tramite un tipo una stringa) ed il secondo elemento il numero di volte che quella data si ripete (tramite un tipo intero), che sta ad indicare il numero di dimissioni per quella determinata data.

Per fare ciò vengono utilizzati due array:

- array_el_ripetuti, in cui sono stati inseriti tutti gli elementi estratti tramite la query. Data l'estrazione abbiamo la ripetizione di alcune date e l'array presenta la seguente struttura:

[“01-01-2010”, “01-01-2010”, “01-01-2010”, “02-01-2010”, “03-01-2010”]

- array_el_non_ripetuti, in cui sono stati inseriti tutti gli elementi senza ripetizioni per un risultato finale del tipo:

[“01-01-2010”, “02-01-2010”, “03-01-2010”]

Gli elementi appartenenti al primo array sono stati ordinati tramite un algoritmo di ordinamento apposito.

È stato utilizzato un array_coppie in cui sono stati inseriti i dati con la seguente struttura:

[“Data”, numero_intero]

In cui “Data” rappresenta la data, mentre numero_intero rappresenta il numero di volte che quella data viene ripetuta.

Infine, l'array_coppie è stato scisso in due array:

- l'array denominato “x”, in cui sono state inserite tutte le date, come tipo stringa
- l'array denominato “y”, in cui nell'i-esima posizione, è stato inserito un intero rappresentante il numero di dimissioni per la data in posizione i-esima dell'array x

Tramite la funzione get_from_db() che ritorna questi due array, x ed y, abbiamo i dati a disposizione e possiamo utilizzarli.

1.2.1 Modifica e miglioramento delle features

Tramite la funzione create_date_features() andiamo a modificare l'array x. Vogliamo ottenere al suo interno degli elementi che siano in grado di descrivere al meglio la data e darci più informazioni possibili. Mentre prima la data era descritta solamente dal giorno, dal mese e dall'anno, aggiungiamo delle informazioni come, ad esempio, quale giorno della settimana (es. lunedì) al fine di aumentare la semantica dei dati.

Aggiungiamo quindi le seguenti features:

- 1) Il giorno, un numero compreso tra 1 e 31
- 2) Il mese, un numero compreso tra 1 e 12
- 3) L'anno, un numero che dato il nostro dataset è compreso tra 2012 e 2020
- 4) Giorno della settimana, un numero che va da 0 (lunedì) a 6 (domenica)
- 5) N-esimo giorno dell'anno, un numero che va da 1 (1° gennaio) a 365 (31 dicembre)
o 366 se l'anno è bisestile
- 6) Working day, un numero booleano che sarà uguale a 0 nel caso in cui quel giorno
è un giorno festivo, uguale a 1 altrimenti.

Per determinare se un giorno sia un giorno lavorativo, è stato utilizzata la seguente libreria:

```
from workalendar.europe import Italy
```

e tramite la funzione `is_working_day()` otteniamo il valore booleano che ci serve.

Ogni elemento dell'array `x` avrà quindi la seguente forma:

[1,1,2012,0,1,0]

Che ci indica che si tratta del primo gennaio del 2012, corrispondente ad un lunedì, corrispondente al primo giorno dell'anno, che è un giorno in cui non si lavora.

I vettori `x` e `y` avranno una lunghezza di 2613 elementi ed ogni elemento del vettore `x` sarà composto da un totale di 6 features.

I dati vengono normalizzati tramite media e deviazione standard per evitare “overfitting” ovvero un buon risultato del nostro modello per i dati di training ma un cattivo risultato su dati che non ha mai visto e quindi di test

```
division=int((15/100) *len(x_orig)) +1
division=len(x_orig)-division
x_tr = torch.Tensor(x_orig[:division])
y_tr = torch.Tensor(y_orig[:division])
x_te = torch.Tensor(x_orig[division:])
y_te = torch.Tensor(y_orig[division:])
```

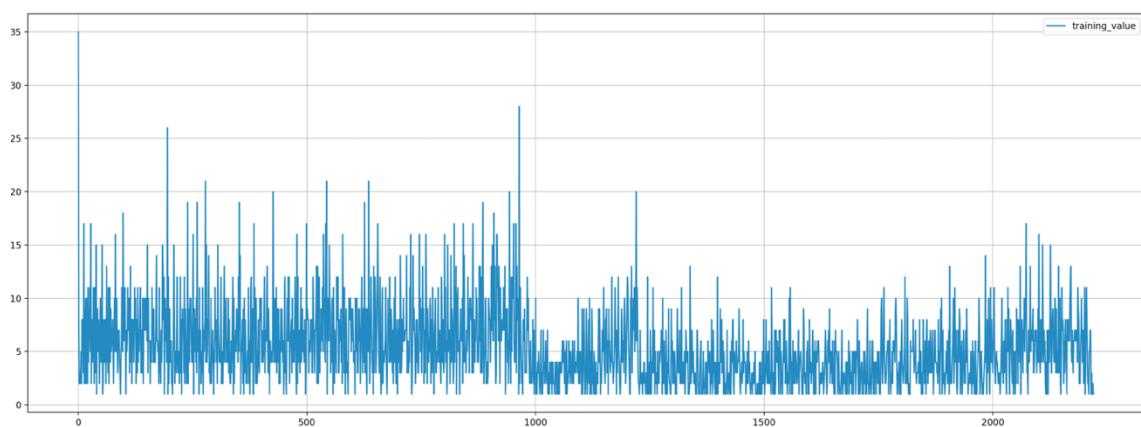
1.2.2 Divisione del dataset e visualizzazione dei dati

Dividiamo il nostro dataset in Training set e Test set. Il training set costituirà l'85% del dataset, mentre il test set il 15%.

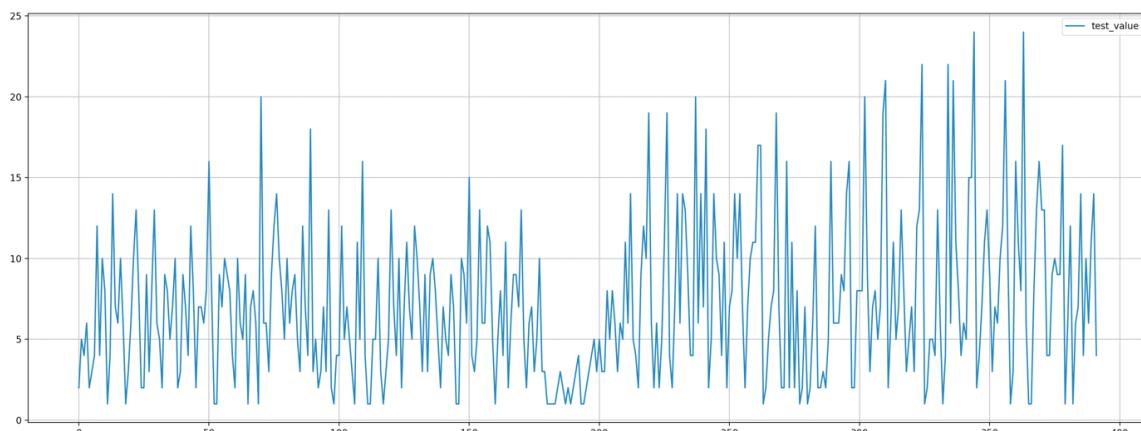
Inseriamo i nostri dati all'interno di tensori forniti da pytorch tramite la funzione `torch.Tensor()`.

Per avere più chiaro l'andamento dei dati possiamo visualizzarli:

- Training set, composto da 2221 elementi:



- Test set, composto da 392 elementi:



CAPITOLO 2 – Tecniche utilizzate ed esperimenti effettuati

Tutti i modelli allenati con pyTorch hanno utilizzato la seguente pipeline per il training:

```
lr = 0.01
epochs = 10000
means = x_tr.mean(0)
stds = x_tr.std(0)
X_training_norm = (x_tr-means)/stds
X_testing_norm = (x_te-means)/stds
x_tot_norm = (x_tot-means)/stds

if reg=="linearRegression":
    reg = LinearRegression(6,1)
    print("linear model")

if reg=="model":
    reg = Model(6,1)
    print("model")

if reg=="mlp":
    reg = MLP(6)
    print("mlp")

optimizer = torch.optim.SGD(reg.parameters(), lr=lr, weight_decay = 0.0001, momentum=0.95)

losses_train = []
losses_test = []
output_te = []
output_tr = []

for e in range(epochs):
    reg.train()

    output_tr = reg(X_training_norm)

    l= RMSE(output_tr.view(-1), y_tr)
    losses_train.append(l.item())

    l.backward()

    optimizer.step()
    optimizer.zero_grad()

    reg.eval()

    with torch.set_grad_enabled(False):
        output_te = reg(X_testing_norm)
        l= RMSE(output_te.view(-1), y_te)
    losses_test.append(l.item())
```

È stato utilizzato un learning rate pari a 0.01, un numero di epoche per il training pari a 10.000.

Come parametro di ottimizzazione è stato utilizzata la SGD, ovvero la discesa del gradiente in cui abbiamo utilizzato come weight_decay il valore 0.0001 e come momentum, parametro utile alla convergenza dell'algoritmo, un valore pari a 0.95.

È possibile osservare la procedura di training, all'interno del ciclo for che itera è nel range del numero di epoche.

2.1 Metodi e soluzioni proposte con pytorch e python

Tramite l'utilizzo di pytorch è possibile utilizzare la struttura di alcune reti neurali, già implementate e funzionanti.

Pytorch è una libreria machine learning open source, utilizzata per applicazioni nel campo della computer vision, machine learning e NLP.

Nel seguente progetto sono state utilizzate le seguenti reti neurali:

- Regressione Lineare:

```
class LinearRegression(nn.Module):
    def __init__(self,in_size, out_size):
        super(LinearRegression,self).__init__()
        self.linear = nn.Linear(in_size, out_size)

    def forward(self, x):
        result = self.linear(x)
        return result
```

- Multi-Layer Perceptron:

```

class MLP(nn.Module):
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(n_inputs, 10)
        nn.init.xavier_uniform_(self.hidden1.weight)
        self.act1=nn.LeakyReLU()

        self.hidden2 = nn.Linear(10, 8)
        nn.init.xavier_uniform_(self.hidden2.weight)
        self.act2=nn.LeakyReLU()

        self.hidden3 = nn.Linear(8, 1)
        nn.init.xavier_uniform_(self.hidden3.weight)
        self.dropout = nn.Dropout(p=0.5,inplace=True)

    def forward(self, X):
        X = self.hidden1(X)
        X = self.act1(X)
        X = self.hidden2(X)
        X = self.act2(X)
        X = self.hidden3(X)

    return X

```

- Support Vector Machine:

```

def __init__(self, *, epsilon=0.0, tol=1e-4, C=1.0,
            loss='epsilon_insensitive', fit_intercept=True,
            intercept_scaling=1., dual=True, verbose=0,
            random_state=None, max_iter=1000):
    self.tol = tol
    self.C = C
    self.epsilon = epsilon
    self.fit_intercept = fit_intercept
    self.intercept_scaling = intercept_scaling
    self.verbose = verbose
    self.random_state = random_state
    self.max_iter = max_iter
    self.dual = dual
    self.loss = loss

```

2.1.1 Algoritmi di valutazione

Dopo aver sviluppato i nostri modelli, è necessario, oltre la visualizzazione grafica, l'utilizzo di apposite misure di errore. Ogni misura ha una semantica diversa, quindi è

necessario utilizzarla sceglierla in maniera corretta in base al risultato che si vuole ottenere. In questo elaborato utilizzeremo le seguenti misure di valutazione:

- 1) **MAE**, mean absolute error, esso viene misurato in valore assoluto e rappresenta una media totale in cui viene calcolata la differenza, per ogni valore, tra il valore reale ed il valore predetto:

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

- 2) **MSE**, mean squared error, rappresenta una media totale in cui viene calcolata la differenza, per ogni valore, tra il valore reale ed il valore predetto. Il risultato di questa differenza viene elevato al quadrato:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

- 3) **RMSE**, root mean squared error, rappresenta una media totale in cui viene calcolata la differenza, per ogni valore, tra il valore reale ed il valore predetto. Il risultato di questa differenza viene elevato al quadrato. Il risultato finale ottenuto viene inserito sotto radice quadrata:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$$

- 4) **R2_score**, rappresenta la porzione di varianza nelle variabili dipendenti che è prevedibile dalle variabili indipendenti. Nella frazione troviamo al numeratore la somma dei quadrati residui e al denominatore il totale della somma dei quadrati:

$$R^2 = 1 - \frac{RSS}{TSS}$$

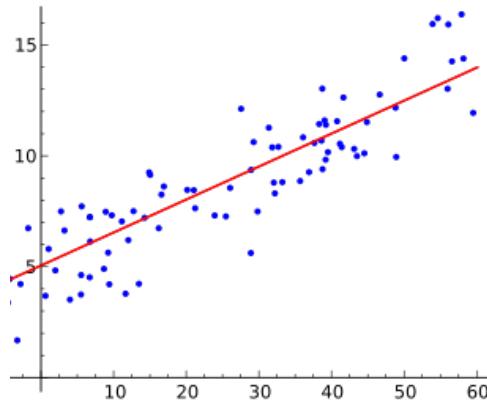
2.1.2 Regressione Lineare

La regressione lineare è un processo statistico che cerca di stabilire una relazione tra due o più variabili. Fornito al modello un valore x , questo restituirà il corrispondente valore y generato dall'elaborazione di x .

La regressione lineare ha come ipotesi la seguente equazione:

$$y = xA^T + b$$

che essendo una funzione lineare, risulterà molto efficiente nel momento in cui i nostri dati fittano una funzione lineare, come possiamo vedere dal seguente esempio



Per calcolare l'errore andiamo a fare la proiezione di ogni dato sulla retta in modo da ottenere un termine errore che viene addizionato alla formula.

La funzione costo della regressione lineare è la seguente:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

minimizzata tramite discesa del gradiente tramite il seguente aggiornamento:

```

repeat until convergence {
     $\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$ 
     $\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$ 
}

```

Al fine di migliorare in maniera corretta il risultato, dobbiamo trovare i valori migliori e fare aggiornamento dei parametri in maniera simultanea

Correct: Simultaneous update

```

temp0 :=  $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ 
temp1 :=  $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ 
 $\theta_0 := \text{temp0}$ 
 $\theta_1 := \text{temp1}$ 

```

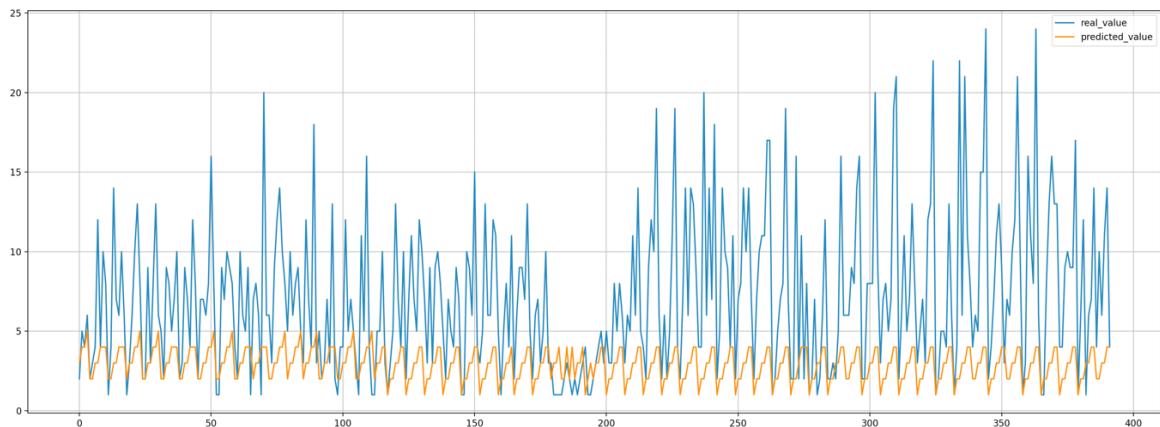
Incorrect:

```

temp0 :=  $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ 
 $\theta_0 := \text{temp0}$ 
temp1 :=  $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ 
 $\theta_1 := \text{temp1}$ 

```

Il risultato ottenuto tramite la regressione lineare sul nostro set di dati risulta essere il seguente:



in cui i valori della linea blu rappresentano i valori reali del test set, mentre la linea arancione rappresenta i valori predetti dal modello secondo il training effettuato.

Come già possibile vedere dal grafico, il risultato ottenuto non è per niente attendibile.

Scarso risultato confermato dai metodi di evaluation:

- 1) R2 Score = -0.63
- 2) MAE = 4.49
- 3) RMSE = 37.62
- 4) MSE = 6.13

Tramite l'utilizzo di una nuova rete neurale che utilizza sempre la regressione lineare, ma con le seguenti caratteristiche

```
reg = torch.nn.Sequential(  
    torch.nn.Linear(6, 12),  
    torch.nn.LeakyReLU(),  
    torch.nn.Linear(12, 12),  
    torch.nn.LeakyReLU(),  
    torch.nn.Linear(12, 1),  
)
```

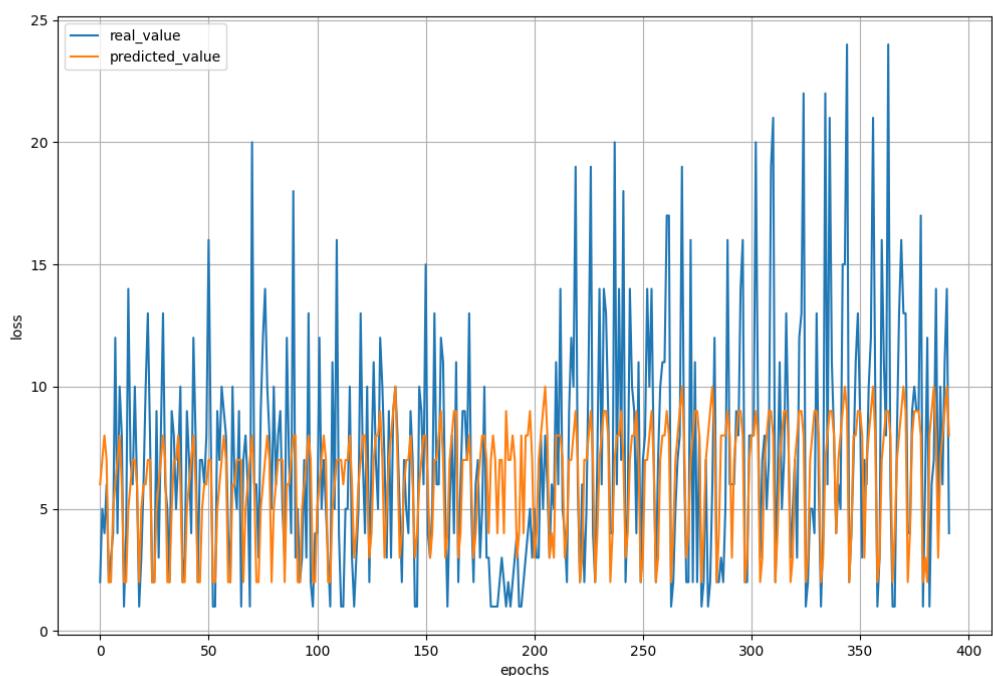
Utilizziamo come funzione di attivazione la ReLU, definita come la parte positiva del suo argomento secondo l'equazione

$$f(x) = x^+ = \max(0, x)$$

Risulta essere una tra le più utilizzate in quanto diventa costante durante la backpropagation e quindi non pesa nella computazione ed in più è una tra le più veloci ed in grado di annullare problemi di saturazione. La ReLU non porta indietro il gradiente.

La LeakyReLU è una particolare versione della ReLU che riesce a portare un minimo di gradiente dietro.

Siamo riusciti a migliorare notevolmente i risultati, come possiamo vedere dal seguente grafico



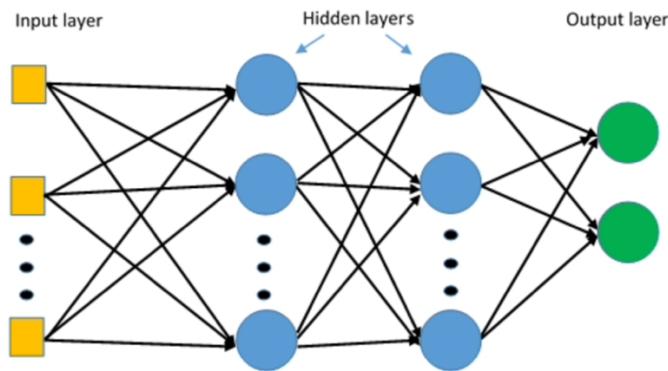
E dal risultato dei metodi di evaluation:

- 1) R2 Score = 0.25
- 2) MAE = 3.06
- 3) RMSE = 17.3
- 4) MSE = 4.16

2.1.3 MLP – Multi-layer Perceptron

Le multi-layer perceptron sono delle reti neurali artificiali che compiono due passaggi:

- Trasformare l'input x in $a(x) = (\theta f x + b f)$, detta rappresentazione di x , serve a rappresentare il dato grezzo in un nuovo spazio per essere utile al task che vogliamo risolvere
- Trasformare il risultato intermedio $a(x)$ in un output $h(x) = \theta h f(x) + \theta h$



Questo tipo di reti presentano la seguente struttura e sono utilizzate per far fittare i dati in una funzione non lineare. Sono per questo molto utilizzati con dati 1d e poco con dati 2d o 3d come immagini o audio che hanno bisogno di una rappresentazione più complessa e creano un elevato numero di elementi all'interno della rete che la rende poco utilizzabile per la descrizione di dati multidimensionali.

Nell'implementazione possiamo scegliere il numero di hidden layers ed il numero di output da generare

```

class MLP(nn.Module):
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(n_inputs, 10)
        nn.init.xavier_uniform_(self.hidden1.weight)
        self.act1=nn.LeakyReLU()

        self.hidden2 = nn.Linear(10, 8)
        nn.init.xavier_uniform_(self.hidden2.weight)
        self.act2=nn.LeakyReLU()

        self.hidden3 = nn.Linear(8, 1)
        nn.init.xavier_uniform_(self.hidden3.weight)
        self.dropout = nn.Dropout(p=0.5,inplace=True)

    def forward(self, X):
        X = self.hidden1(X)
        X = self.act1(X)
        X = self.hidden2(X)
        X = self.act2(X)
        X = self.hidden3(X)

    return X

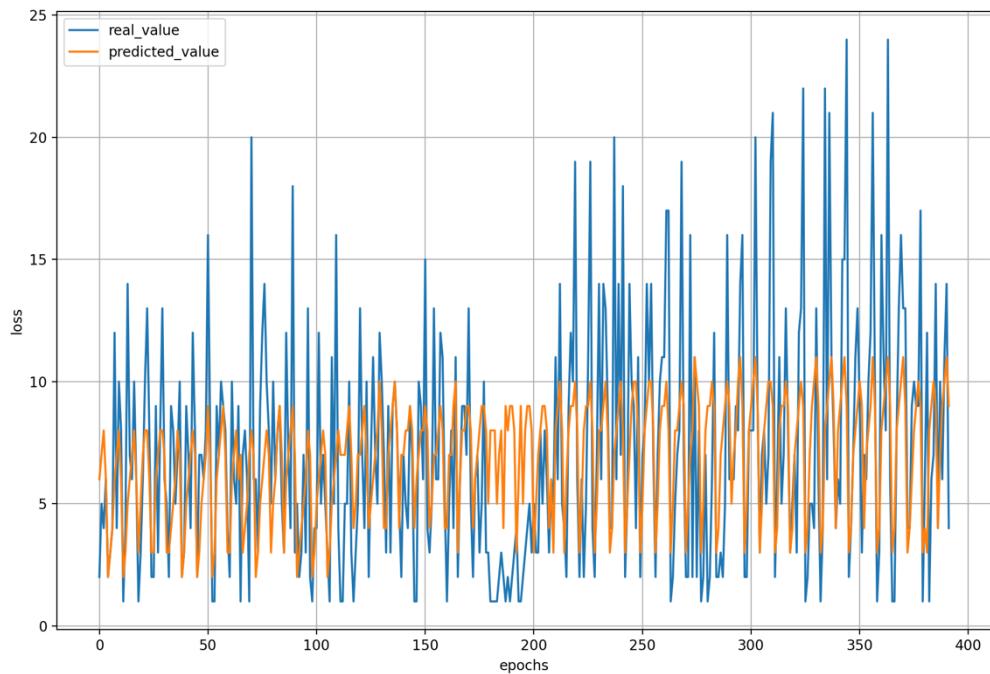
```

Come possiamo vedere sono stati utilizzati 3 hidden layers all'interno del nostro elaborato:

- Il primo layer prende i dati di input (`n_inputs=6`, cioè il nostro numero di features) e li inserisce all'interno di 10 nodi. Viene utilizzata la LeakyReLU per tutti gli hidden layers.
- Il secondo layer prende in input i 10 pesi uscenti dal layer1 e li inserisce in 8 nodi.
- Il terzo layer prende in input gli 8 pesi uscenti dal layer2 e li inserisce in 1 nodo che sarà quindi l'output finale.

Viene utilizzato il dropout: il dropout è una tecnica che permette l'interruzione di alcuni nodi casuali all'interno della rete in maniera del tutto probabilistica, tramite una probabilità che viene specificata come parametro della funzione. È stato utilizzato un valore di 0.5 nel seguente elaborato.

Il grafico ottenuto è il seguente:



E dal risultato dei metodi di evaluation:

- 1) R2 Score = 0.29
- 2) MAE = 3.01
- 3) RMSE = 16.35
- 4) MSE = 4.04

2.1.4 SVM - Support Vector Machine

L'SVM è un metodo di apprendimento supervisionato associato ad algoritmi di apprendimento per la regressione o la classificazione. Ciò che differenzia l'SVM dagli altri metodi è la capacità di descrivere le features non più su uno spazio 2d, ma su uno spazio 3d, trovando l'equazione di un piano di divisione per le features.

La funzione di loss dell'SVM presenta la seguente formula

$$L = \underbrace{\frac{1}{N} \sum_i \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)}_{\text{data loss}} + \lambda \underbrace{\sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}}$$

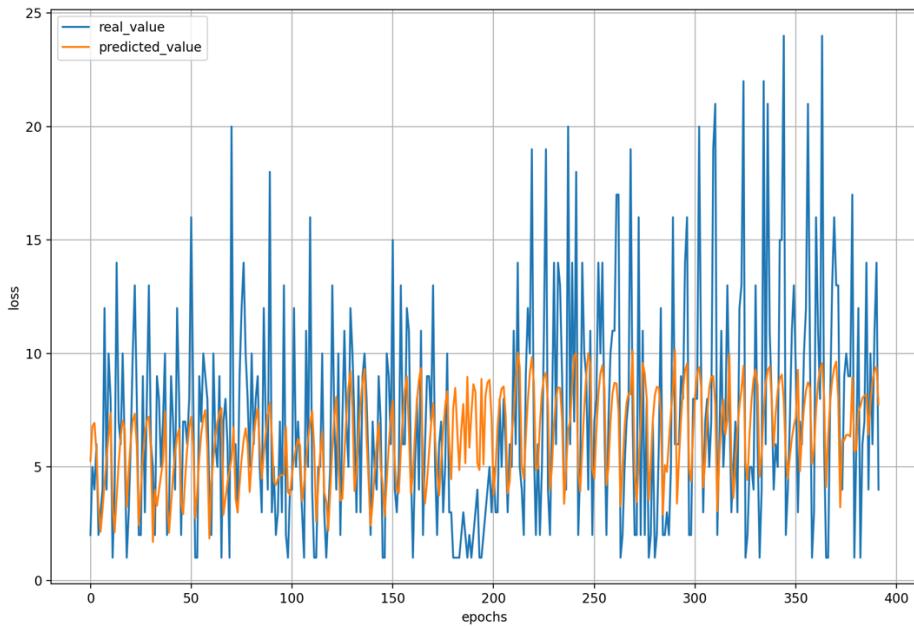
Utilizziamo la libreria sklearn per utilizzare l'SVM per quello che è l'SVR (Support Vector Regressor). Utilizziamo il seguente modello

```
model = SVR(kernel='rbf', C=20, epsilon=0.0001, gamma='auto')
```

in cui utilizziamo i seguenti parametri:

- Kernel='RBF' (Radial Basis Function Network), una neural network artificiale che utilizza la funzione radial basis come funzione di attivazione
- C=20, è un iperparametro per il controllo dell'errore. Non c'è una regola per scegliere il seguente parametro, ma devono essere 'provati' dei determini valori che spesso risultano essere i migliori. Vediamo successivamente come trovare i valori migliori per un determinato modello
- Epsilon=0.001, specifica il margine di tolleranza in cui non viene dato valore agli errori. Più grande è il valore, meno peso viene dato agli errori.
- Gamma='auto', viene trovato tramite la formula 1/numero_di_features

Facciamo previsione dei nostri nuovi risultati ottenendo il seguente risultato



Con i seguenti risultati di score:

- 1) R2 Score = 0.21
- 2) MAE = 3.29
- 3) RMSE = 18.46
- 4) MSE = 4.3

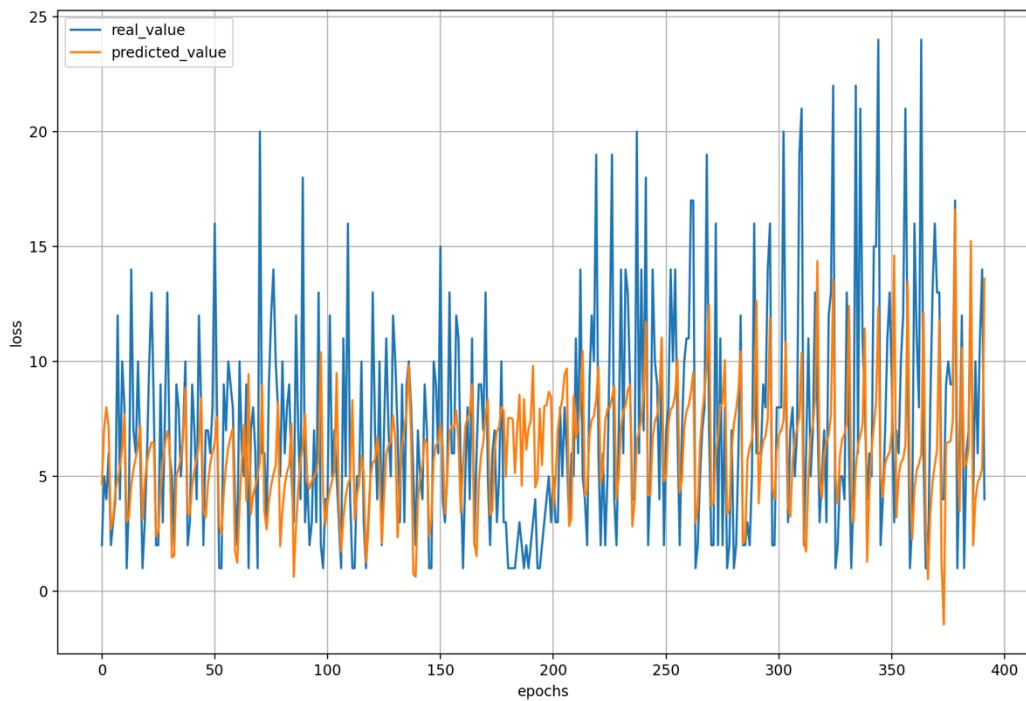
Possiamo utilizzare pure un altro kernel per il seguente elaborato. Il kernel='poly' riesce a creare un'equazione polinomiale su quale far fissare i nostri dati.

```
model = SVR(kernel='poly', C=20, gamma='auto', degree=4, epsilon=0.01, coef0=6)
```

Vediamo che tra i parametri vengono aggiunti:

- Degree, che rappresenta il grado del polinomio
- Coef0, che rappresenta i termini indipendenti nella funzione kernel

Otteniamo il seguente grafico



Con i seguenti risultati di score:

- 1) R2 Score = 0.15
- 2) MAE = 3.37
- 3) RMSE = 19.62
- 4) MSE = 4.43

2.1.5 Altri esperimenti effettuati

Sono stati effettuati pure altri esperimenti tramite il quale però sono stati ottenuti risultati bassi:

- **Decision Tree**, alberi decisionali che tramite delle scelte decisionali sono in grado di effettuare regressione. Abbiamo ottenuto i seguenti valori di score:
 - 1) R2 Score = -1.03
 - 2) MAE = 5.16
 - 3) RMSE = 46.86
 - 4) MSE = 6.85

- **K-Neighbors**, algoritmo in grado di effettuare classificazione che conserva tutti i possibili valori di output data una features e per una nuova features in input, associa l'output relativo alla feature di training più simile. I risultati ottenuti sono stati i seguenti:
 - 1) R2 Score = -0.38
 - 2) MAE = 4.14
 - 3) RMSE = 31.82
 - 4) MSE = 5.64
- **Ridge**, è una tecnica per l'analisi di regressioni multiple di dati multicollineari. Per multicollineari si intende l'esistenza di relazioni lineari tra le variabili indipendenti. I risultati ottenuti sono stati i seguenti
 - 1) R2 Score = -0.51
 - 2) MAE = 4.28
 - 3) RMSE = 34.87
 - 4) MSE = 5.9

Come possibile intuire dai risultati di score ottenuti dai seguenti modelli, non sono utilizzabili per scopi pratici reali.

CAPITOLO 3 – Composizione del main

È possibile dividere la funzione main del seguente elaborato in due parti:

- Training dei modelli, in cui tramite alcune funzioni e alcune specifiche andiamo a modificare la funzione o il modello utilizzato per fare regressione. Le possibili funzioni sono le seguenti:

```
# la variabile reg puo prendere i seguenti valori = linearRegression1, mlp
train_and_creation_model(iteration_number=1,reg="mlp",save=True)

# tramite il seguente modello otteniamo il risultato relativo al linearRegression2
train_net_reglm(iteration_number=100,save=True)

# la variabile model puo prendere i seguenti valori = rbf, poly, decisionTree, kNeighbors, ridge
training_sklearn(model="ridge",show=False,show_numbers=False,save=True)
```

- Testing dei modelli, in cui possiamo sfruttare i modelli già allenati tramite due funzioni:

```
# la variabile reg puo prendere i seguenti valori = linearRegression1, linearRegression2, mlp, poly, rbf
load_model(reg= "linearRegression1")

# la variabile reg puo prendere i seguenti valori = decisionTree, KNeighbors, Ridge
load_model_sklearn(reg="decisionTree")
```

CAPITOLO 4 – Conclusioni

4.1 Valutazioni finali

Il risultato finale non è dei migliori nonostante i diversi algoritmi di machine learning ed i diversi modelli utilizzati all'interno del seguente progetto.

La quantità dei dati non è tanto elevata da fornire un dataset completo e data l'autenticità e la veridicità dei dati non è possibile effettuare quelli che sono dei comuni algoritmi di data augmentation.

Il modello SVM tramite il kernel poly potrebbe sicuramente fornire il migliore risultato per il seguente laboratorio, essendo forniti di un calcolatore più potente, data la complessità dell'algoritmo e dell'elaborazione dei dati.

	R2 score	MAE	RMSE	MSE
Regressione Lineare 1	-0.63	4.49	37.62	6.13
Regressione Lineare 2	0.25	3.06	17.3	4.16
MLP	0.29	3.01	16.35	4.04
SVM – RBF	0.21	3.29	18.46	4.3
SVM - POLY	0.15	3.37	19.62	4.43
Decision Tree	-1.03	5.16	46.86	6.85
K-Neighbors	-0.38	4.14	31.82	5.64
Ridge	-0.51	4.28	34.87	5.9

Il risultato migliore è stato ottenuto con il MLP, ma nonostante questo il risultato non è ancora utilizzabile per la realizzazione di una previsione nel mondo reale, tale da risultare buono.

4.2 Miglioramenti effettuati e possibili miglioramenti

I miglioramenti effettuati durante lo sviluppo del progetto sono stati i seguenti:

- 1) Inizialmente i dati del dataset facevano riferimento ad un solo anno in campo medico-sanitario e si trovavano all'interno di un file con estensione ".xml". Siamo passati da un totale di 400 dati a un totale di 2613.
- 2) Sono stati migliorati i parametri del modello SVM e ottenuti risultati migliori. Per trovare i migliori parametri, abbiamo utilizzato la funzione GridSearchcv() che ci permette di trovare i migliori hyperparametri dato un modello. Vengono effettuati diversi tentativi tramite i valori che vengono forniti degli hyperparametri

I possibili miglioramenti effettuabili sono invece i seguenti:

- 1) Sarebbe possibile migliorare tutti gli score aggiungendo dati all'interno del nostro dataset
- 2) Per i modelli dipendenti dai parametri è possibile provare e trovare dei nuovi parametri tali da migliorare lo score di essi
- 3) Potrebbero essere utilizzati nuovi modelli come, per esempio, le Recurrent Neural Network.