2-Regressione

March 3, 2021

1 Regressione

1.1 Ipotesi e Definizione

La **regressione** è una tecnica attraverso cui data una variabile dipendente e una o più indipendenti si cerca di stabilire quale sia la relazione tra la variabile dipendenti e variabili indipendenti, per tradurre tutto ciò in matematico:

$x; variabile; indipendente, ; y; variabile; dipendente \rightarrow y = f(x)$

Ora che abbiamo capito che noi vogliamo trovare $f(x)$, ma in molti casi i dati non rappresentazione una situazione esatta, è pertanto necessario modificare quella formula al fine di tenere conto di un maggior numero di casi, ovvero la relazione ora diventerà: $y = f(x) + \epsilon$. Il termine ϵ è un termine che permette di tenere conto di molteplici fattori come rumore o altro ed in molti casi **non è eliminabile**.

1.2 Errore

Tenendo conto di ciò possiamo quindi procedere al calcolo della stima dell'errore attraverso diverse metriche nella regressione, ovvero come calcolare in maniera quantitativa $y - f(x)$, tra le metriche più utilizzate ci sono:

- Errore Quadratico Medio(Root Mean Square Error o RMSE) $\rightarrow \frac{1}{n} \sum_{i=1}^{n} \sqrt{y_i^2 - f(x_i)^2}$

- Errore Assoluto Medio (Mean Absolute Error o MAE) $\rightarrow \frac{1}{n} \sum_{i=1}^{n} |y_i - f(x_i)|$

- Errore Log(cosh) $\frac{1}{n} \sum_{i=1}^{n} |y_i - log(cosh(x_i))|$

- $R^2 \rightarrow 1 - \frac{RSS}{TSS}$ $\quad RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$ devianza residua, $TSS = \sum_{i=1}^{n} (y_i - \hat{y})^2$ devianza totale, calcolata con $\hat{y}$ valore medio dei dati.

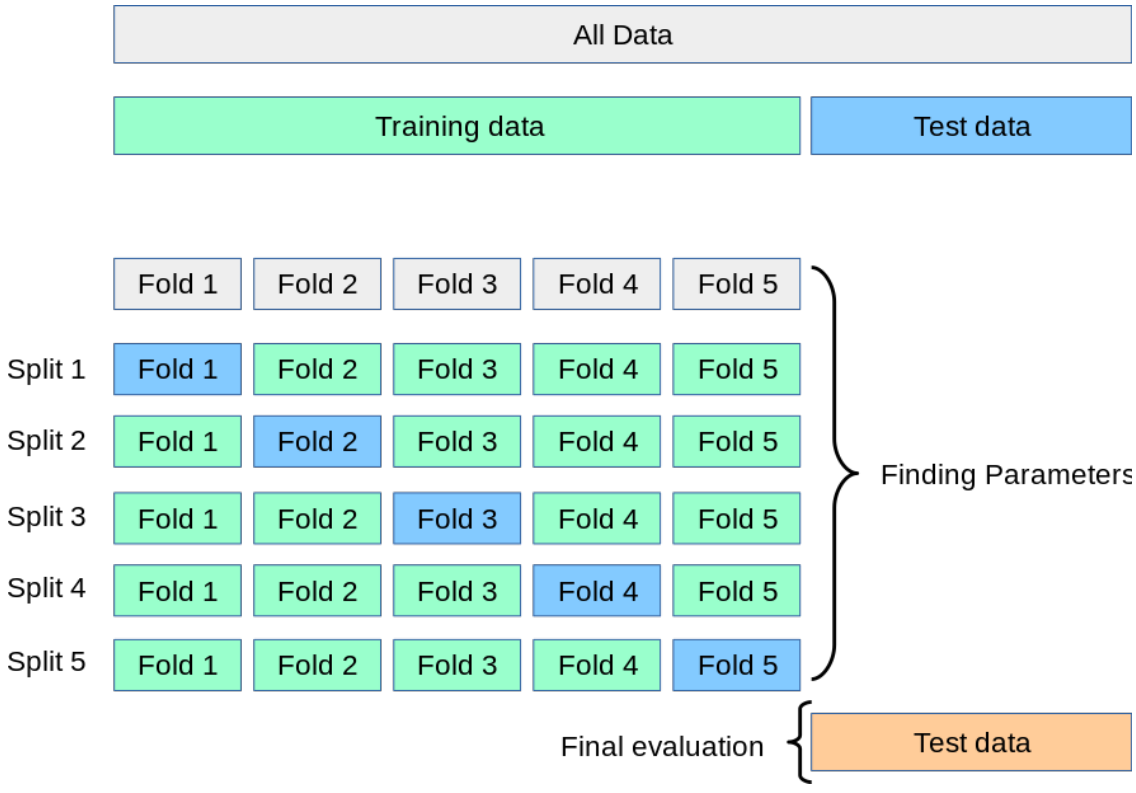
1.3 Come ridurre l'errore- Training

Ora che abbiamo capito come valutare il fit è necessario cercare di migliorarlo ora in questo caso è necessario ragionare in termini di coefficienti per la regressione, in genere per una qualsiasi regressione N-dimensionale è possibile ragionare in termini **matriciali** se infatti noi prendiamo il caso lineare possiamo tradurre tutto in matrici $y = mx + q \rightarrow Y = WX + B$ con W matrice dei pesi, mentre b è il bias ovvero l'intercetta nel caso lineare ed in alcune trattazione viene assarboito all'interno della W. Tornando a come ridurre l'errore, in alcuni casi possono essere utili

metodi di geometria lineare, ma qualora si vada in dimensioni troppo alte la complessità computazionale richiede altri metodi come ad esempio lo **Stocastic Gradient Descent(SGD)** che presenta la seguente forma: $W = W - \alpha \frac{\partial L}{\partial f} \frac{\partial f}{\partial W}$, dove il simbolo ∂ indica la derivata parziale, mentre α è un termine definibile da noi o adattativo come vedremo in seguito, questo procedimento richiede che all'aggiornamento della W sia ricalcolato l'errore per essere riusato in seguito per il nuovo aggiornamento, il numero di volte da fare determina una maggiore precisione, ma attenzione che in molti casi l'errore oltre un certo tot diventa estremamente piccolo determinando solo poche variazioni. Lo **SGD** verrà approfondito di più nelle lezioni successive.

1.4 Testare la regressione - Testing

Per testare ora la regressione è possibile usare una parte dei dati **non usata per il training** per calcolare su di essi l'errore del fit, l'errore sul **test set** è composto principalmente da due parti: $err = bias + varianza$ il *bias* rappresenta l'errore sistematico del modello ovvero la sua incapacità di riconoscere i pattern, mentre la *varianza* rappresenta la volatilità del modello a fornire previsioni. Qualora si sta parlando di ***overfitting*** si riferisce aal fatto che il modello presenta una basso **bias**, ma un alta **varianza** ovvero il modello è troppo complesso per il dataset, mentre ***underfitting*** nel caso contrario ovvero il modello è troppo semplica per la cattura dei pattern. In genere i dataset di **training** e **testing** sono usati in aggiunta ad un altro metodo che è la **cross-validation**, ovvero dal dataset si estraggono sempre due parti il **train set** e **test set**, ma appena si è finito di allenare si riprende il dataset ed la parte che in precedenza era il **train set ora diventerà il test set, mentre il test set diventerà il train set** ripetendo fino a che ogni sezione di grandezza a noi scelta sarà stata sia test set che train set, ovvero:



Il motivo di tale suddivisione sono due: - aumentare il dataset disponibile per train e test - offrire

maggiore flessibilità e generalizzazione

1.5 Proviamo a usare scikit!

Ora che abbiamo capito come funziona una regressione e come sia possibile valutare tutto usando scikit.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston

#otteniamo il dataset
data = load_boston()
#convertiamo il dataset in pandas
df = pd.DataFrame(data.data, columns = data.feature_names)
df['MEDV'] = data.target
display(df)
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       ...   ...    ...   ...    ...    ...   ...     ...  ...    ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

1.5.1 Usare la regressione lineare con il metrodo OLS(Ordinary Least Square)

Prima di vedere come applicare la regressione lineare, cerchiamo di capire su cosa si basa il **metodo OLS** o chiamato anche in italiano **metodo dei minimi quadrati**, la formula su cui si basa il metodo è : $\min_w \parallel wX - Y \parallel^2$. La formula precedente ci dice che poiché noi possiamo controllare solo i valori di w, noi cerchiamo quei valori per cui la funzione assume il valore minimo possibile, come vedremo in seguito questa funzione può portare a problemi di **overfitting** o **underfitting** in particolare quando utilizziamo il metodo del gradiente, vedremo poi in seguito delle tecniche per ridurre questa possibiltà. Ora che abbiamo capito che abbiamo un dataset con 506 dati e 13 colonne dette anche **feature** e come funziona il metodo di regressione lineare, proviamo a vedere il comportamento del Median value of owner-occupied homes ogni 1000 dollari(MEDV) in funzione della lower status of the population(LSTAT) e vediamo se facendo una regressione lineare cosa si ottiene.

[2]:
```
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

#dividiamo il dataset in train e test
X_train, X_test, y_train, y_test = train_test_split(df.LSTAT.values, df.MEDV.
 ↪values, random_state=0,
                                                    test_size = 0.2)

#creiamo il modello con scikit
model = LinearRegression()
#alleniamo il modello
#reshape (-1,1) significa convertire i dati prima in array e poi come una␣
 ↪matrice colonna
model.fit(X_train.reshape(-1, 1), y_train)

#creiamo i futuri fit usando dei range
x_range = np.linspace(df.LSTAT.min(), df.LSTAT.max(), 100)
y_range = model.predict(x_range.reshape(-1, 1))

#creiamo la figura
fig = go.Figure([
    go.Scatter(x=X_train, y=y_train, name='train', mode='markers'),
    go.Scatter(x=X_test,  y=y_test, name='test', mode='markers'),
    go.Scatter(x=x_range, y=y_range, name='prediction')
])
fig.update_xaxes(title_text = "LSTAT")
fig.update_yaxes(title_text = "MEDV")
fig.show()
```

Per sapere come sia fatto il modello possiamo richiedere più informazioni semplicemente richiamando le funzioni.

[3]:
```
print('Coefficienti modello', model.coef_)
print('Parametri', model.get_params())
```

```
Coefficienti modello [-0.95648761]
Parametri {'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize':
False, 'positive': False}
```

Per valutare come sia il uo score possiamo usare la funzione all'interno dell'oggetto oppure usare delle metriche fornite da scikit.

[4]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#create the predicted output
y_ptrain = model.predict(X_train.reshape(-1, 1))
y_ptest  = model.predict(X_test.reshape(-1, 1))

print('R2 score')
print('Training Set:', r2_score(X_train,y_ptrain), 'Test Set:', r2_score(X_test,␣
 ↪y_ptest))
print('RMSE')
print('Training Set:', mean_squared_error(X_train,y_ptrain), 'Test Set:',␣
 ↪mean_squared_error(X_test, y_ptest))
print('MAE')
print('Training Set:', mean_absolute_error(X_train,y_ptrain), 'Test Set:',␣
 ↪mean_absolute_error(X_test, y_ptest))
```

```
R2 score
Training Set: -4.6742007815061575 Test Set: -5.418161658507217
RMSE
```

```
Training Set: 301.5495989395749 Test Set: 269.0927779444686
MAE
Training Set: 15.019429187947798 Test Set: 13.961546682555158
```

Ora che abbiamo capito come funziona il modello di scikit e come possiamo testare e allenare su una serie di dati, proviamo ad applicare il concetto di **cross validation**, per farlo utilizziamo la funzione già presente in scikit e usiamo un numero di folds uguale a 5 applicando in ognuno il modello e vediamo cosa otteniamo.

[5]:
```
from sklearn.model_selection import cross_validate

#effettua la cross validation impostando il numero di folds e le metriche
cv_results = cross_validate(model, df.LSTAT.values.reshape(-1,1), df.MEDV.
 ↪values, cv =5,
                            scoring = ['r2', 'neg_mean_squared_error',␣
 ↪'neg_mean_absolute_error'],
                            return_train_score = True)
display(cv_results)
```

```
{'fit_time': array([0.00100374, 0.00099945, 0.00100064, 0.00100541, 0.00100136]),
 'score_time': array([0.0019958 , 0.00099945, 0.00100136, 0.00099874, 0.        ]),
 'test_r2': array([0.31784807, 0.5406078 , 0.07608699, 0.42423767, 0.1267687 ]),
 'train_r2': array([0.55953888, 0.53499276, 0.55004371, 0.5617624 , 0.5056059 ]),
 'test_neg_mean_squared_error': array([-23.55825   , -41.82157437, -73.99360893,␣
 ↪-50.50118016,
        -23.21775049]),
 'train_neg_mean_squared_error': array([-42.7285291 , -37.97136234, -31.25967296,␣
 ↪-35.78590001,
        -42.49238511]),
 'test_neg_mean_absolute_error': array([-4.18200738, -4.38425663, -6.00666715, -5.
 ↪43917773, -3.74895682]),
 'train_neg_mean_absolute_error': array([-4.81563608, -4.39256578, -3.88848603, -4.
 ↪44496347, -4.84370493])}
```

I dati che abbiamo ottenuto rappresentano i valori di score del modello usando le diverse metriche, come possiamo notare i valori in genere sono in simili sia in fase di training e di testing, questo significa che il modello è in grado di generalizzare indipendentemente dalla sezione del dataset che si usa per allenare. Qualora volessimo solo ottenere le previsioni del modello senza considerare lo scoring possiamo usare un altra funzione di scikit:

[6]:
```
from sklearn.model_selection import cross_val_predict

pred_results = cross_val_predict(model, df.LSTAT.values.reshape(-1,1), df.MEDV.
 ↪values, cv =5)
display(pred_results)
```

```
array([30.70217411, 26.55374947, 31.6495307 , 32.73649774, 30.353148  ,
       30.4728141 , 23.27290401, 16.57160266,  5.8215984 , 18.61589846,
       15.27521996, 22.43524134, 20.00203073, 27.43130083, 25.43686591,
```

27.22188517, 29.10662617, 21.03913689, 24.01084493, 24.41970409,
14.706806 , 21.87679956, 17.00040617, 15.84363391, 19.41367243,
19.20425676, 20.89952645, 18.43639931, 22.90393355, 23.72165187,
13.13120241, 22.66460136, 8.03542117, 17.36937663, 15.38491388,
26.01525204, 24.29006582, 26.92271993, 25.56650418, 31.36033764,
33.6938265 , 30.84178456, 29.87448362, 28.24901915, 26.14489031,
25.48672678, 21.55768997, 16.92062877, 4.94404704, 19.51339418,
22.2557422 , 26.2645564 , 30.40300887, 27.26177386, 20.90949862,
30.87170108, 29.91437232, 31.7293081 , 28.82740528, 26.47397207,
22.55490744, 21.26849691, 28.95704355, 26.19475118, 27.6407165 ,
31.01131153, 25.45681026, 27.59085563, 22.61474049, 26.90277558,
28.96701573, 25.81580854, 30.16367668, 28.14929741, 28.90718268,
26.75319296, 23.73162404, 25.42689373, 23.36265358, 26.59363816,
30.3930367 , 28.46840699, 28.96701573, 28.17921393, 26.07508508,
29.15648704, 22.8441005 , 27.25180169, 30.18362103, 29.98417754,
26.88283123, 27.49113388, 27.53102258, 29.47559663, 25.10778414,
29.03682095, 24.35987105, 31.47003156, 32.10825074, 29.49554098,
26.27452858, 28.01965914, 23.93156342, 21.33844389, 22.36277224,
18.54230433, 16.52133217, 20.73861197, 22.41814134, 19.3912972 ,
21.74448396, 24.36528804, 18.78223709, 17.97015696, 24.09767072,
19.19750535, 22.63038956, 24.23609347, 19.5574045 , 21.18156477,
20.48022284, 20.57250467, 17.19498955, 10.29230839, 17.51797597,
20.07418277, 8.58509446, 17.87787512, 19.53894813, 16.81663403,
22.11361129, 22.42736953, 23.47938243, 19.87116273, 17.76713692,
18.09012334, 18.14549244, 20.2772028 , 14.06663539, 16.70589583,
11.44583131, 1.98694335, 8.99113453, 9.36026187, 6.71177324,
 8.08677256, 18.37619703, 6.49029684, 7.60690702, 13.946669 ,
20.72938379, 21.48609482, 22.55656409, 19.1698208 , 19.78810908,
19.88039092, 18.84683438, 29.50538618, 27.80740044, 26.92149483,
28.66562149, 32.14464662, 31.96931114, 30.67736547, 22.9995169 ,
24.68827445, 30.3266945 , 22.53810773, 23.4978388 , 23.29481876,
20.42485373, 22.63961775, 20.18492097, 25.39884457, 24.84515357,
28.82250061, 24.41142895, 27.936595 , 27.35521945, 29.09011793,
26.76461571, 25.02048906, 29.29313796, 28.49951419, 20.84012199,
21.60606121, 29.63458074, 27.57669585, 29.53307073, 28.76713151,
29.03474883, 29.41310434, 31.09263372, 29.09934611, 29.69917803,
31.00035188, 29.97602353, 25.79565646, 27.63206495, 29.53307073,
29.63458074, 26.8845821 , 30.87115732, 29.17668809, 29.97268057,
23.13399246, 23.04840187, 16.98002907, 19.89010911, 12.67482241,
17.65619472, 11.91306616, 18.71751804, 24.40929225, 7.14567032,
24.33226072, 20.87440089, 24.14396142, 17.09985589, 23.45067764,
24.1268433 , 14.06994902, 23.938544 , 25.93280474, 28.89423914,
28.47484525, 29.7587041 , 26.99412805, 29.08253844, 29.21948338,
22.46638586, 27.9441836 , 30.32360199, 29.05686126, 25.54764709,
23.1254334 , 24.2723473 , 28.38925466, 26.99412805, 26.1296631 ,
22.69748045, 21.82445644, 22.8344254 , 27.99553795, 21.73886585,
16.63766671, 24.59759154, 23.75024471, 24.28946542, 26.82294687,
27.38784477, 29.36498739, 29.41634174, 29.40778268, 26.81438782,

```
24.52056001, 29.77582222, 28.05545137, 25.77018262, 26.53193887,
24.22955201, 26.22381275, 27.37928571, 22.80874822, 25.50485179,
23.49347294, 19.77884135, 26.06974969, 29.73302692, 20.75457407,
21.3109129 , 26.7972697 , 25.82153698, 26.80582876, 29.41634174,
29.88708998, 27.25945888, 28.87712103, 26.28372616, 28.28654596,
29.21948338, 28.50908149, 29.86141281, 29.73302692, 25.71882827,
25.39358403, 21.37082631, 26.32652145, 25.93280474, 24.29802448,
29.58752292, 29.39066456, 28.41493184, 25.09401696, 23.53626823,
27.07115958, 26.11254498, 18.88014016, 28.18383725, 28.38069561,
27.24234076, 24.30658354, 25.01698543, 28.2779869 , 28.59823713,
26.61343853, 29.05474081, 28.00279755, 30.97007147, 25.58134325,
22.93163711, 29.54101647, 23.84464447, 27.63560981, 26.26609877,
24.06297232, 17.28488508, 19.65671942, 25.19430753, 22.84232118,
28.33028932, 28.65778109, 27.83408967, 23.82479649, 29.40208057,
30.43417584, 29.37230859, 22.78277722, 25.58134325, 28.19135342,
26.45465464, 23.14004097, 27.70507776, 29.83873626, 28.77686901,
27.52644589, 25.75005114, 24.99582766, 27.03024624, 25.80959509,
26.25617478, 30.02729213, 26.89131033, 28.35013731, 30.90060352,
25.02559964, 22.90186514, 29.16390474, 29.53109248, 29.63033241,
29.54101647, 30.02729213, 27.74477373, 31.00976744, 27.48674992,
29.94790019, 18.00933657, 22.30642555, 24.08282031, 22.90186514,
27.74477373, 21.3934182 , 25.36301541, 20.94683851, 30.22577199,
28.40968127, 21.58197406, 22.2468816 , 32.24034258, 31.7739149 ,
32.53806237, 26.01799895, 26.66305849,  0.96984054, -2.20583723,
22.13771767, 12.4122045 , 14.39700311, 11.96562482, 13.86110749,
18.3963723 , 14.555787  , 12.05494076, 11.10223742,  5.07837366,
 4.89974178,  7.41051202,  3.7287106 ,  5.08829765, 14.78403884,
18.49561223, 16.85815338,  9.99075021, 20.42086688, 19.2498357 ,
18.48568824, 16.25278981, 15.70697019,  5.11806963,  5.7333572 ,
 8.90903497, 15.31001047, 15.31993446, 15.85583009,  8.30367139,
12.99231392, 12.64600977, 23.42953623,  9.70242449, 16.07057304,
25.3726873 , 14.68535643,  2.03563538, 15.78198624, -0.47506971,
 7.15324116, 10.28921763,  9.47155506, 15.26253002, 13.22318335,
20.64948347, 19.99535341, 21.53448296, 12.69410757, 18.59089769,
11.635956  , 20.00497297, 21.13046146, 14.39676964, 11.93416235,
18.12915882, 16.15714907, 23.52573183, 19.4951363 , 20.50519007,
12.71334669, 17.73475687,  9.65432669,  2.37231997, 13.08850952,
13.82921562, 16.32068159, 19.13921259, 16.96519209, 12.21312958,
12.03035795, 17.98486542, 19.28350599, 17.65780039, 16.52269234,
18.32155001, 18.04258278, 18.48508253, 18.99491919, 17.09986593,
17.65780039, 16.81127913, 18.802528  , 19.48551674, 20.95730938,
19.30274511, 21.00540718, 21.64029812, 25.19953522, 22.38100422,
21.50562429, 18.60051725, 14.58916084, 17.65780039, 20.89959202,
19.42779938, 22.71768881, 21.28437441, 23.88165554, 17.64818083,
11.91492323, 17.12872461, 11.13573889, 17.75399599, 22.48681938,
24.76665504, 27.65252296, 28.35475082, 25.07448095, 22.2655695 ,
24.92056799, 20.68796171, 24.08366629, 17.72513731, 12.03997751,
 6.5472089 , 17.71551775, 22.25594994, 23.54497095, 22.02508051,
```

```
       18.16763706, 14.76231291, 21.53448296, 22.66959101, 20.57252699,
       21.31323309, 25.79594793, 26.36350195, 29.6726305 , 28.86458748,
       27.51784912])
```

1.6 Multiple Linear Regression (MLR)

Nel caso in cui si abbia un dataset che contiene molte features è possibile usare una **MLR** ovvero una regressione lineare su più variabili indipendenti per la previsione di una variabile dipendente da tutte quelle impostate nel modello, vediamo in dettaglio come usarlo.

[7]:
```
#prendiamo ogni colonna tranne quella di MEDV
X = df.drop('MEDV', axis = 1)
Y = df[['MEDV']]
#aplichiamo il modello ora non più in un contesto 1D
#effettua la cross validation impostando il numero di folds e le metriche
cv_results = cross_validate(model, X, Y, cv =5,
                            scoring = ['r2', 'neg_mean_squared_error',␣
 ↪'neg_mean_absolute_error'],
                            return_train_score = True)
display(cv_results)
```

```
{'fit_time': array([0.00599957, 0.00400114, 0.00400162, 0.00500011, 0.00600934]),
 'score_time': array([0.00700021, 0.00699854, 0.00600004, 0.00599885, 0.00698924]),
 'test_r2': array([ 0.63919994,  0.71386698,  0.58702344,  0.07923081, -0.
 ↪25294154]),
 'train_r2': array([0.74652533, 0.72763185, 0.69498059, 0.84181027, 0.73545537]),
 'test_neg_mean_squared_error': array([-12.46030057, -26.04862111, -33.07413798,␣
 ↪-80.76237112,
        -33.31360656]),
 'train_neg_mean_squared_error': array([-24.5892302 , -22.24092194, -21.19051839,␣
 ↪-12.91756328,
        -22.73718934]),
 'test_neg_mean_absolute_error': array([-2.62190565, -3.90725478, -4.386606  , -5.
 ↪57073637, -4.76333993]),
 'train_neg_mean_absolute_error': array([-3.55584094, -3.29666359, -3.20481356, -2.
 ↪74360523, -3.30689921])}
```

A PRESTO, SPERIAMO CHE QUESTA LEZIONE SIA STATA PIACEVOLE!