06_RegressioneEsempio

January 16, 2020

1 Cosa facciamo oggi?

Ripasso Regressione usando il California Housing Dataset!

- Regression Lineare
- Regressione polinomiale
- Decision Tree

2 Dove troviamo i dataset?

L'obbiettivo di oggi è quello di applicare gli algoritmi studiati sia di Regressione che di Classificazione a dei dataset un po' più corposi. L'idea è quella di cominciare ad affrontare problemi il più possibile vicini alla realtà. Per face ciò ci servono due dataset uno di regressione e uno di classificazione.

Dove troviamo i datasets?

- Scikit-Learn Toy datasets
- Scikit-Learn Real World Dataset
- Altri dataset importabili usando Scikit-Learn
- OpenML Dataset
- LIBSVM Regression Dataset

Datasets Scikit Learn Main Page

- Classificazione: The Labeled Faces in the Wild face recognition dataset (esempio)
- Regressione: California Housing dataset (esempio)

2.1 Regressione: California Housing dataset

2.1.1 0) Raccolta Informazioni e Link utili

- Esempio Utile Regressione
- Stesso dataset leggermente aggiornato Kaggle
- Spiegazione dataset

2.1.2 1) Comprensione del dataset

[62]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
import pandas as pd

# Download dataset
cal_housing = fetch_california_housing()

print(cal_housing.keys())
print(cal_housing.DESCR)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.
```

```
.. topic:: References

   - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
     Statistics and Probability Letters, 33 (1997) 291-297
```

- Obbiettivo: Stimare qual'è il valore medio delle case in California.
- Dati: Input o Attribute Information:
  - MedInc: media degli stipendi (annuale) delle persone che vivono nel quartiere in dollari per casa (unità di misura: decine di migliaia di dollari of US Dollars esempio: 8.3252 = 83252 dollari). Date N case in un quartiere, per ogni casa mi calcolo lo stipendio medio annuale e faccio la media.
  - HouseAge: media dell'età delle persone che vivono nella casa. Un numero basso significa che l'edificio è nuovo.
  - AveRooms: numero medio camere per casa
  - AveBedrms: numero medio di camere da letto per casa
  - Population: numero di persone medio che vivono nel quartiere (block)
  - AveOccup: numero mddio di persone che vivono in ogni casa
  - Latitude: latitudine della casa.Una misura che mi dice quanto una casa è a nord
  - Longitude: longitudine della casa. Una misura che mi dice quanto una casa è ad ovest
- Target:
  - HousePrice: valore della casa in centinaia di migliaia di dollari (esempio: 4.526 => 4.526*100*1000 = 452600.0)

2.1.3 2) Estrazione dati dal dataset e conversione a dataframe (pandas)

[0]:
```
X = cal_housing['data']                    # or cal_housing.data
X_names = cal_housing['feature_names']     # or cal_housing.feature_names
Y = cal_housing['target']                  # or cal_houseing.target
cal_housing.update([ ('target_names', ['HousePrice'])] )  # add missing names
Y_names =  cal_housing['target_names']     # cal_houseing.target_names
```

[119]:
```
df_X = pd.DataFrame(data=X, columns =X_names)
df_Y = pd.DataFrame(data=Y, columns =Y_names)
display(df_X)
display(df_Y)
```

```
       MedInc  HouseAge  AveRooms  ...  AveOccup  Latitude  Longitude
0      8.3252      41.0  6.984127  ...  2.555556     37.88    -122.23
1      8.3014      21.0  6.238137  ...  2.109842     37.86    -122.22
2      7.2574      52.0  8.288136  ...  2.802260     37.85    -122.24
3      5.6431      52.0  5.817352  ...  2.547945     37.85    -122.25
4      3.8462      52.0  6.281853  ...  2.181467     37.85    -122.25
...       ...       ...       ...  ...       ...       ...         ...
20635  1.5603      25.0  5.045455  ...  2.560606     39.48    -121.09
```

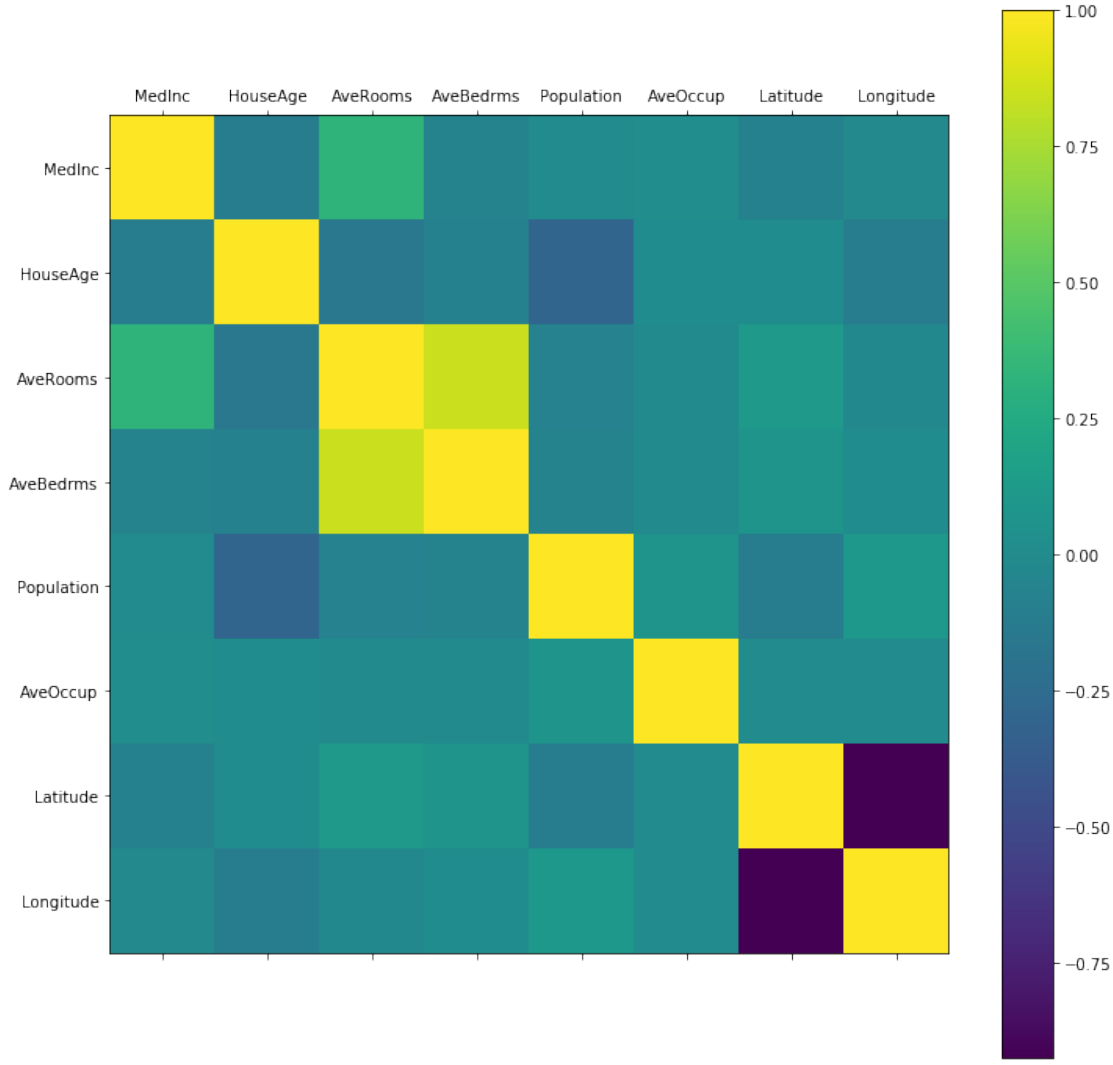
```
20636  2.5568      18.0  6.114035  ...  3.122807     39.49    -121.21
20637  1.7000      17.0  5.205543  ...  2.325635     39.43    -121.22
20638  1.8672      18.0  5.329513  ...  2.123209     39.43    -121.32
20639  2.3886      16.0  5.254717  ...  2.616981     39.37    -121.24

[20640 rows x 8 columns]
```

```
       HousePrice
0           4.526
1           3.585
2           3.521
3           3.413
4           3.422
...           ...
20635       0.781
20636       0.771
20637       0.923
20638       0.847
20639       0.894

[20640 rows x 1 columns]
```

2.1.4 3) Considerazioni e analisi dei dati

Correlation Matrix: Quanto le variabili di input sono correlate? È utile applicare la PCA cioè ridurre il numero di input? Il contenuto di informazioni delle variabili di input è omogeneo oppure vi sono variabili inutili?

[129]:
```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 12]

# Correlation Matrix
plt.matshow(df_X.corr())
plt.xticks(range(len(df_X.columns)), df_X.columns)
plt.yticks(range(len(df_X.columns)), df_X.columns)
plt.colorbar()
plt.show()
display(df_X.corr())
```



	MedInc	HouseAge	AveRooms	...	AveOccup	Latitude	Longitude
MedInc	1.000000	-0.119034	0.326895	...	0.018766	-0.079809	-0.015176
HouseAge	-0.119034	1.000000	-0.153277	...	0.013191	0.011173	-0.108197
AveRooms	0.326895	-0.153277	1.000000	...	-0.004852	0.106389	-0.027540
AveBedrms	-0.062040	-0.077747	0.847621	...	-0.006181	0.069721	0.013344
Population	0.004834	-0.296244	-0.072213	...	0.069863	-0.108785	0.099773
AveOccup	0.018766	0.013191	-0.004852	...	1.000000	0.002366	0.002476
Latitude	-0.079809	0.011173	0.106389	...	0.002366	1.000000	-0.924664
Longitude	-0.015176	-0.108197	-0.027540	...	0.002476	-0.924664	1.000000

[8 rows x 8 columns]

2.1.5 Grafici utili

- Libreria Bamboolib Demo

Non vi è ancora un integrazione con google-colab, in particolare vi è un errore di compatibilità, poichè Bamboolib richiede notebook>=5.3 mentre google-colab usa la versione 5.2.2.

Al fine di utilizzare la libreria è necessario visitare la pagina [Ufficiale](https://bamboolib.8080labs.com/) e premere su "Try the Live Demo".

Copiare nel notebook il seguente codice: "'python from sklearn.datasets import fetch_california_housing from sklearn.model_selection import train_test_split import pandas as pd

Download dataset cal_housing = fetch_california_housing() cal_housing "' Questo Video Demo spiega come utilizzare graficamente la libreria. Inoltre c'è un un interessante tutorial

2.1.6 Preprocessamento

Non è sempre un buona idea normalizzare i dati perchè si perde l'informazione del massimo e del minimo. Tuttavia liner regression algoritmi come Support Vector Machine e Linear Regression convergono più velocemente su data normalizzati.

- StandarScaler: input data con mean value (valore medio) uguale a zero e deviazione standard uguale ad 1.
- MinMaxScaler: scaliamo i dati in un range minimo massimo. Di solito da 0 a 1

La normalizzazione permetterà al nostro algoritmo di raggiungere la stima ottima in un tempo minore di quello che occorre se non normalizzaziamo.

[65]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# --------- Standard Scaler Mean=0, Variance=1
data_X = StandardScaler().fit_transform(df_X.values)
df_X_preprocessed_SC = pd.DataFrame(data = data_X , columns = X_names)

# --------- Min Max Scaler Min=0, Max=1
data_X = MinMaxScaler().fit_transform(df_X.values)
df_X_preprocessed_MMS = pd.DataFrame(data = data_X , columns = X_names)

display(df_X.describe())
display(df_X_preprocessed_SC.describe())
display(df_X_preprocessed_MMS.describe())
```

	MedInc	HouseAge	...	Latitude	Longitude
count	20640.000000	20640.000000	...	20640.000000	20640.000000
mean	3.870671	28.639486	...	35.631861	-119.569704
std	1.899822	12.585558	...	2.135952	2.003532
min	0.499900	1.000000	...	32.540000	-124.350000
25%	2.563400	18.000000	...	33.930000	-121.800000
50%	3.534800	29.000000	...	34.260000	-118.490000

```
75%        4.743250     37.000000  ...     37.710000   -118.010000
max       15.000100     52.000000  ...     41.950000   -114.310000

[8 rows x 8 columns]
```

```
              MedInc      HouseAge  ...      Latitude     Longitude
count  2.064000e+04  2.064000e+04  ...  2.064000e+04  2.064000e+04
mean   3.734255e-16  8.557001e-16  ...  1.256263e-15 -6.527810e-15
std    1.000024e+00  1.000024e+00  ...  1.000024e+00  1.000024e+00
min   -1.774299e+00 -2.196180e+00  ... -1.447568e+00 -2.385992e+00
25%   -6.881186e-01 -8.453931e-01  ... -7.967887e-01 -1.113209e+00
50%   -1.767951e-01  2.864572e-02  ... -6.422871e-01  5.389137e-01
75%    4.593063e-01  6.643103e-01  ...  9.729566e-01  7.784964e-01
max    5.858286e+00  1.856182e+00  ...  2.958068e+00  2.625280e+00

[8 rows x 8 columns]
```

```
              MedInc      HouseAge  ...      Latitude     Longitude
count  20640.000000  20640.000000  ...  20640.000000  20640.000000
mean       0.232464      0.541951  ...      0.328572      0.476125
std        0.131020      0.246776  ...      0.226988      0.199555
min        0.000000      0.000000  ...      0.000000      0.000000
25%        0.142308      0.333333  ...      0.147715      0.253984
50%        0.209301      0.549020  ...      0.182784      0.583665
75%        0.292641      0.705882  ...      0.549416      0.631474
max        1.000000      1.000000  ...      1.000000      1.000000

[8 rows x 8 columns]
```

2.1.7 4) Divisione del dataset in training e test

[66]:
```
X_train, X_test, Y_train, Y_test = train_test_split(df_X.values, df_Y.values,␣
 ↪test_size=0.1, random_state=0) # data not normalized
X_train_SC, X_test_SC, Y_train, Y_test = train_test_split(df_X_preprocessed_SC.
 ↪values, df_Y.values, test_size=0.1, random_state=0) # normalize data␣
 ↪standard scaler
X_train_MMS, X_test_MMS, Y_train, Y_test =␣
 ↪train_test_split(df_X_preprocessed_MMS.values, df_Y.values, test_size=0.1,␣
 ↪random_state=0) # normalize data minmax scaler

print("Training Input Dimensions: ", X_train.shape)
print("Training Output Dimensions: ", Y_train.shape)
print("Test Input Dimensions: ", X_test.shape)
print("Test Output Dimensions: ", Y_test.shape)
```