Exercicio - Regressão Linear

November 23, 2020

1 Exercicio Regressão Linear

Considere o dataset a seguir. O dataset apresenta um conjunto de caracteristicas para o levantamento de preços de residências para a região de boston

[132]:
```
import numpy as np
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt

boston = load_boston()
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
```

```
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
…', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

1.1 Parte 1 - Analise dos dados

Observe os dados de forma grafica e veja a correlação entre as diferentes features do modelo. Observe quais são as melhores features do modelo. Considere o`pairplot`como um bom candidato para a tarefa

[133]:
```python
### Seu codigo aqui. Voce pode utilizar quantas celulas voce achar necessario.
import seaborn as sns

df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['target'] = pd.Series(boston.target)
df.head()
```

[133]:

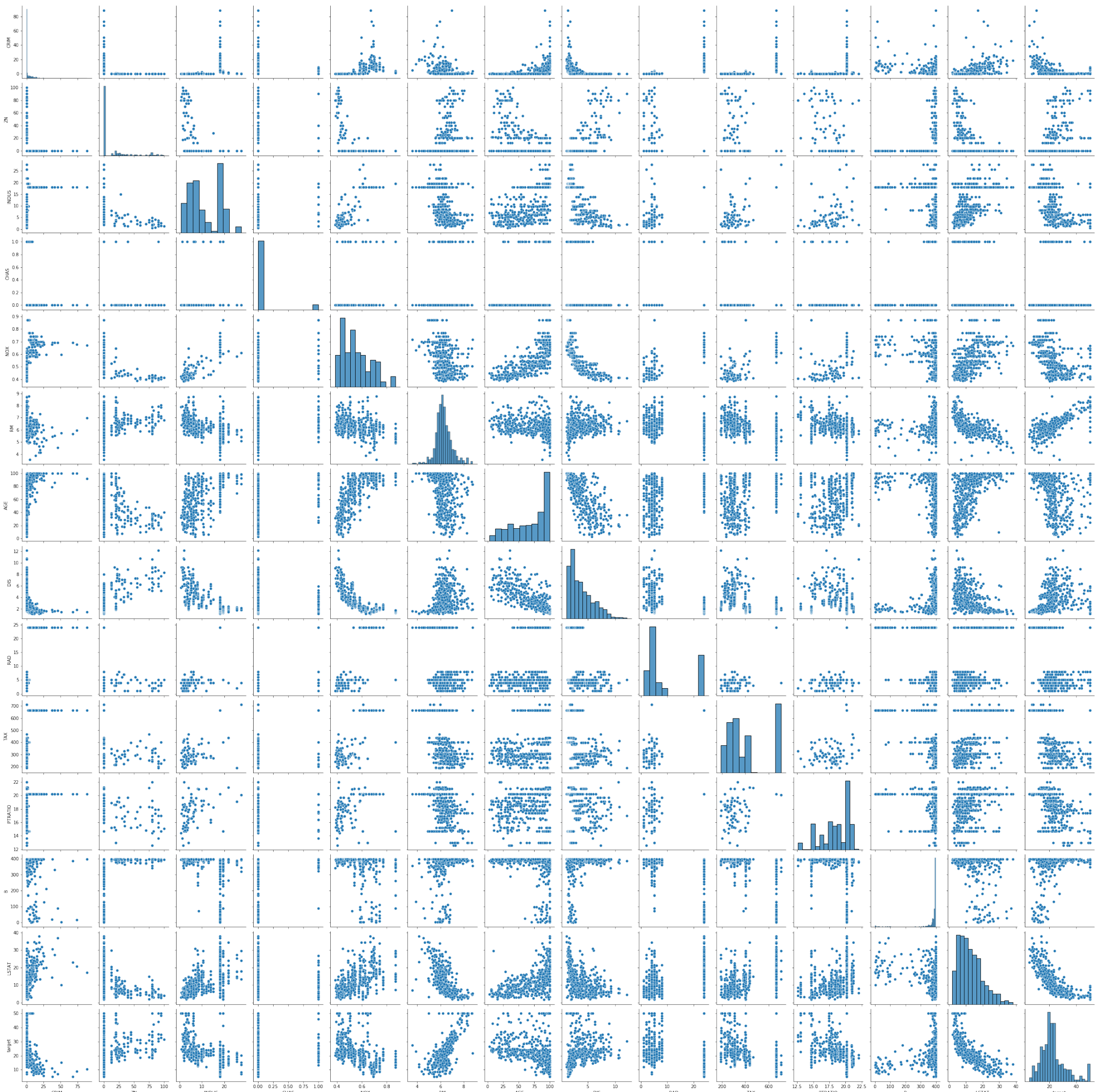
```
       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   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

   PTRATIO       B  LSTAT  target
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
```

[134]:

```
sns.pairplot(df)
```

[134]:

```
<seaborn.axisgrid.PairGrid at 0x7f97851a7640>
```



1.2 Parte 2 - Utilizando a biblioteca do Sklearn

1.2.1 Parte 2.1 - Separando os conjuntos de treinamento e teste

Separe o seu dataset com as features que você achou mais conveniente (podem ser todas as features inclusive) em um conjunto de treinamento e teste. Utilize a proporção 80%-20%

[135]:
```
# Exemplo
# Observe que os vetores X e y sao repartidos
# em dois vetores com 70% e 30% dos dados originais
from sklearn.model_selection import train_test_split

X, y = np.arange(10).reshape((5, 2)), range(5)
```

```
print(f"Vetor de features original \n{X}")
print(f"Vetor de previsoes original \n{y}")


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3)

print(f"Vetor de features de treinamento\n{X_train}")
print(f"Vetor de previsoes de treinamento \n{y_train}")

print(f"Vetor de features de teste\n{X_test}")
print(f"Vetor de previsoes de teste\n{y_test}")
```

```
Vetor de features original
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
Vetor de previsoes original
range(0, 5)
Vetor de features de treinamento
[[6 7]
 [2 3]
 [4 5]]
Vetor de previsoes de treinamento
[3, 1, 2]
Vetor de features de teste
[[0 1]
 [8 9]]
Vetor de previsoes de teste
[0, 4]
```

[136]:
```
X = df.drop(columns =['target'])
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2)
```

1.2.2 Parte 2.2 - Criando um regressor com a biblioteca sklearn

Crie um regressor utilizando a biblioteca padrão do sklearn. Utilize o conjunto de treinamento para fitar o modelo.

[137]:
```
# Seu código aqui
from sklearn import linear_model
linearRegression = linear_model.LinearRegression()
```

[138]:
```
linearRegression.fit(X_train, y_train)
```

[138]:
```
LinearRegression()
```

[139]:
```
y_pred = linearRegression.predict(X_test)
```

1.2.3 Parte 2.3 - Verificando qualidade do modelo

Observe a qualidade do seu regressor com o uso do RMSE. O RMSE é uma função que calcula o erro médio quadrado do regressor e dado pela seguinte equação.

$$\mathrm{RMSE} = \left[\sum_{i=1}^{N} \left(y_i - \bar{y}\right)^2 / N\right]^{1/2}$$

Onde: * y_i corresponde a cada saida do regressor * $\bar{y}$ média das saidas do regressor * N número de elementos

[165]:
```
# Seu codigo aqui
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared=False)
```

[165]:
```
4.5432041287323415
```

1.3 Parte 3 - Criando um modelo de regressão linear do zero

Utilze o modelo apresentado na aula para criar um regressor linear utilizando apenas a biblioteca numpy como base. Para tal, implemente o método dos mínimos quadrados apresentado em aula e verifique a qualidade do modelo assim como mostrado na **Parte 2.3**

[166]:
```
print(df.corr())
```

```
             CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
ZN      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
target  -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955
```

```
              DIS       RAD       TAX   PTRATIO         B     LSTAT    target
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
target   0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

[167]:
```
# Seu codigo aqui
features_train = X_train[['RM', 'LSTAT', 'PTRATIO']].to_numpy() # Variáveis com␣
 ↪maior
                                                                # correlação
target_train = y_train.to_numpy()
print(features_train)
```

```
[[ 6.854  2.98  17.6  ]
 [ 7.831  4.45  17.8  ]
 [ 6.782  6.68  15.2  ]
 …
 [ 5.036 25.68  20.2  ]
 [ 5.868  9.97  16.9  ]
 [ 5.874  9.1   18.7  ]]
```

[168]:
```
# Criando os pesos baseado na quantidade de variáveis dependentes
weights = np.random.rand(len(features_train[0]))
weights
```

[168]:
```
array([0.56983317, 0.03481006, 0.17661372])
```

[169]:
```
# Definindo o bias
b = np.random.rand(1)
bias = np.array([b[0] for i in range(len(features_train))])
```

[170]:
```
# Função de Regressão Linear
def linearReg(features, weights, bias):
    y_hat = weights.dot(features.transpose()) + np.array([bias[0] \
                                    for i in range(len(features))])
    return y_hat
```

[171]:
```
y_hat = linearReg(features_train, weights, b)
print(y_hat[:5])
```

```
[7.86433545 8.50755599 7.52823176 8.12401739 7.16082401]
```

[172]:
```
# Função para calcular o erro
def rootMeanSquaredError(y, y_hat):
    mse = np.sum((y - y_hat)**2)/len(y)
    return np.sqrt(mse)
```

[178]:
```
# Antes da otimização o erro está alto
print('Total Error: {}'.format(rootMeanSquaredError(target_train, y_hat)))
```

```
Total Error: 17.253574068379628
```

[174]:
```
# Função para calcular os gradientes
def gradient(target, features, weights, bias):
    # Retorna o gradiente para weigths and biases
    m = len(features)
    target_pred = linearReg(features, weights, bias)
    loss = target -target_pred # y - y_hat
    # Calculo do gradiente para o bias
    grad_bias = np.array([-2/m * np.sum(loss)])

    grad_weights = np.ones(len(features[0]))
    # Calculo dos gradientes
    for i in range(len(features[0])):
        grad_weights[0] = -2/m * np.sum(loss * \
                                        (np.array([feature[0] for feature in␣
 ↪features])))
    return grad_bias, grad_weights
```

[175]:
```
# Função de otimização
def stochGradDes(learning_rate, epochs, target, features, weights, bias):
    MSE_list = []
    for i in range(epochs):
        grad_bias, grad_weights = gradient(target, features, weights, bias)
        weights -= grad_weights * learning_rate
        bias -= grad_bias * learning_rate
        new_pred = linearReg(features, weights, bias)
        total_MSE_new = rootMeanSquaredError(target, new_pred)
        MSE_list.append(total_MSE_new)
    return_dict = {'weights':weights, 'bias':bias[0], 'RMSE':total_MSE_new,␣
 ↪'MSE_list':MSE_list}
    return return_dict
```

[176]:
```
model_val = stochGradDes(0.001, 2000, target_train, features_train, weights,␣
↪bias)
```

[177]:
```
# Aqui o erro já reduz, ,após passar pela função de otimização
print("Weights: {}\nBias: {}\nRMSE: {}".format(model_val['weights'],␣
↪model_val['bias'], model_val['RMSE']))
```

```
Weights: [12.03518065 -1.96518994 -1.82338628]
Bias: 4.2918386187266995
RMSE: 15.936304417944147
```