GradientBoost_Boston_Real_Estate

September 19, 2022

Link a repositorio en GitHub

1 Importar bibliotecas

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from scipy.stats import uniform as sp_randFloat
     from scipy.stats import randint as sp_randInt
     %matplotlib inline
     %pip install mlxtend --upgrade
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.7/dist-packages
(0.14.0)
Collecting mlxtend
  Downloading mlxtend-0.21.0-py2.py3-none-any.whl (1.3 MB)
     || 1.3 MB 23.7 MB/s
Requirement already satisfied: scipy>=1.2.1 in
/usr/local/lib/python3.7/dist-packages (from mlxtend) (1.7.3)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.7/dist-
packages (from mlxtend) (1.1.0)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.7/dist-
packages (from mlxtend) (1.3.5)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.7/dist-packages (from mlxtend) (1.0.2)
Requirement already satisfied: matplotlib>=3.0.0 in
/usr/local/lib/python3.7/dist-packages (from mlxtend) (3.2.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from mlxtend) (57.4.0)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.7/dist-
packages (from mlxtend) (1.21.6)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in
```

```
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from
kiwisolver>=1.0.1->matplotlib>=3.0.0->mlxtend) (4.1.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.24.2->mlxtend) (2022.2.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.1->matplotlib>=3.0.0->mlxtend) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->mlxtend)
(3.1.0)
Installing collected packages: mlxtend
  Attempting uninstall: mlxtend
    Found existing installation: mlxtend 0.14.0
    Uninstalling mlxtend-0.14.0:
      Successfully uninstalled mlxtend-0.14.0
Successfully installed mlxtend-0.21.0
```

1.1 Importar módulos de Scikit-learn

[ ]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
 ↪GradientBoostingRegressor, HistGradientBoostingRegressor
from sklearn.svm import SVR, NuSVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import tree
from sklearn.model_selection import GridSearchCV
import numpy as np
from scipy.stats import loguniform
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV
from mlxtend.evaluate import bias_variance_decomp
```

2 Importar Dataset

- **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)ˆ2 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

```
[ ]: url = "https://raw.githubusercontent.com/crisb-7/BostonRealEstate/main/
      ↪bostonRealEstate.csv"
```

```
[ ]: df = pd.read_csv(url)
```

```
[ ]: df.head()
```

```
[ ]:        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
     0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
     1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
     2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
     3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
     4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

             B  LSTAT  MEDV
     0  396.90   4.98  24.0
     1  396.90   9.14  21.6
     2  392.83   4.03  34.7
     3  394.63   2.94  33.4
     4  396.90   5.33  36.2
```

3 Exploración del dataset

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 511 entries, 0 to 510
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     511 non-null    float64
 1   ZN       511 non-null    float64
 2   INDUS    511 non-null    float64
 3   CHAS     511 non-null    int64
 4   NOX      511 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      511 non-null    float64
 7   DIS      511 non-null    float64
 8   RAD      511 non-null    int64
 9   TAX      511 non-null    int64
 10  PTRATIO  511 non-null    float64
 11  B        511 non-null    float64
 12  LSTAT    511 non-null    float64
 13  MEDV     511 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 56.0 KB
```

Al ver que se tienen solo 5 registros con valores nulos para RM, se quitan estas filas para tener un conjunto de datos homogéneo.

[ ]:
```
df = df.dropna(axis = 0)
df.shape
```

[ ]:
```
(506, 14)
```

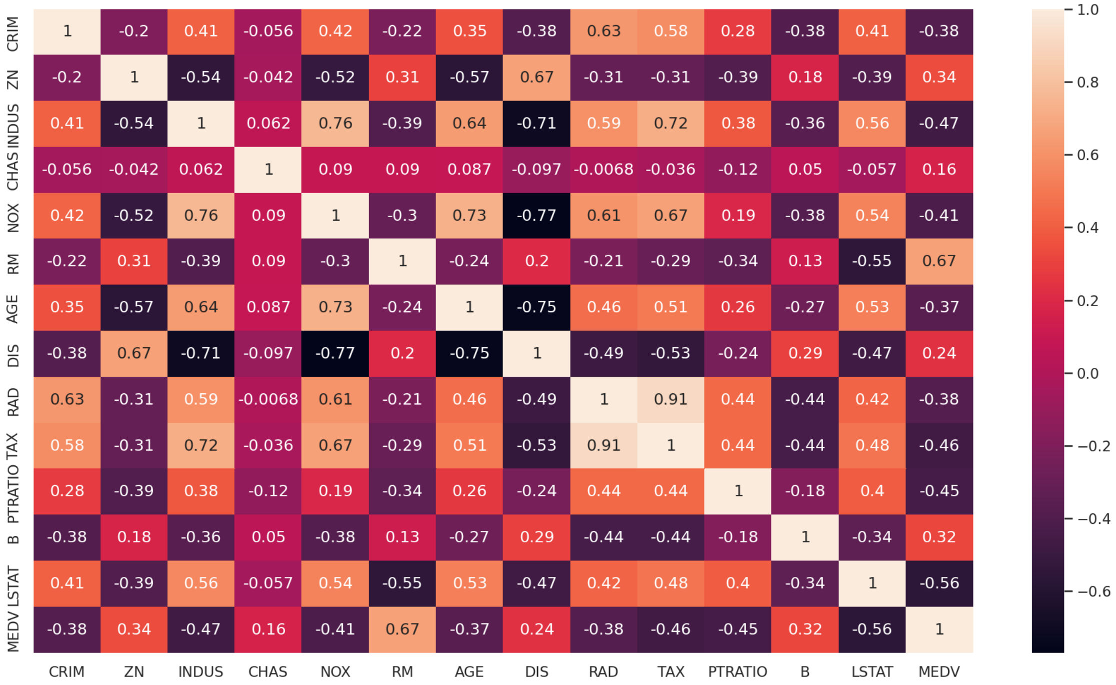
[ ]:
```
df.describe()
```

[ ]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.617404   11.289526   11.174842    0.069170    0.555209    6.287589
std      8.600123   23.325350    6.824592    0.253994    0.115611    0.703802
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082268    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.266005    0.000000    9.690000    0.000000    0.538000    6.209000
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.629750
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.555731    3.775231    9.531621  408.330040   18.498419  356.228379
std     28.161573    2.096147    8.716661  168.382685    2.202078   91.253462
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.098500    4.000000  280.250000   17.400000  374.687500
50%     77.500000    3.122200    5.000000  330.000000   19.100000  391.260000
75%     93.975000    5.117675   24.000000  666.000000   20.200000  396.210000
max    100.000000   12.126500   24.000000  711.000000   23.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.872569   22.711858
std      7.823528    9.520520
min      1.730000    5.000000
25%      6.950000   17.025000
```

```
50%      11.465000    21.200000
75%      17.107500    25.075000
max      76.000000    67.000000
```

[ ]:

```python
sns.set(rc={'figure.figsize':(16, 9)})
plt.rcParams["figure.dpi"] = 150
sns.heatmap(df.corr(), annot = True)
plt.show()
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	-0.2	0.41	-0.056	0.42	-0.22	0.35	-0.38	0.63	0.58	0.28	-0.38	0.41	-0.38
ZN	-0.2	1	-0.54	-0.042	-0.52	0.31	-0.57	0.67	-0.31	-0.31	-0.39	0.18	-0.39	0.34
INDUS	0.41	-0.54	1	0.062	0.76	-0.39	0.64	-0.71	0.59	0.72	0.38	-0.36	0.56	-0.47
CHAS	-0.056	-0.042	0.062	1	0.09	0.09	0.087	-0.097	-0.0068	-0.036	-0.12	0.05	-0.057	0.16
NOX	0.42	-0.52	0.76	0.09	1	-0.3	0.73	-0.77	0.61	0.67	0.19	-0.38	0.54	-0.41
RM	-0.22	0.31	-0.39	0.09	-0.3	1	-0.24	0.2	-0.21	-0.29	-0.34	0.13	-0.55	0.67
AGE	0.35	-0.57	0.64	0.087	0.73	-0.24	1	-0.75	0.46	0.51	0.26	-0.27	0.53	-0.37
DIS	-0.38	0.67	-0.71	-0.097	-0.77	0.2	-0.75	1	-0.49	-0.53	-0.24	0.29	-0.47	0.24
RAD	0.63	-0.31	0.59	-0.0068	0.61	-0.21	0.46	-0.49	1	0.91	0.44	-0.44	0.42	-0.38
TAX	0.58	-0.31	0.72	-0.036	0.67	-0.29	0.51	-0.53	0.91	1	0.44	-0.44	0.48	-0.46
PTRATIO	0.28	-0.39	0.38	-0.12	0.19	-0.34	0.26	-0.24	0.44	0.44	1	-0.18	0.4	-0.45
B	-0.38	0.18	-0.36	0.05	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1	-0.34	0.32
LSTAT	0.41	-0.39	0.56	-0.057	0.54	-0.55	0.53	-0.47	0.42	0.48	0.4	-0.34	1	-0.56
MEDV	-0.38	0.34	-0.47	0.16	-0.41	0.67	-0.37	0.24	-0.38	-0.46	-0.45	0.32	-0.56	1

[ ]:

```python
# sns.pairplot(df)
# plt.show()
```

4 Preprocesamiento de los datos

[ ]:

```python
scaler = StandardScaler()
```

[ ]:

```python
x = scaler.fit_transform(df.drop(columns = "MEDV"))
y = df.MEDV
y= y.values
```

5 División train-test

```
[ ]: x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.90,␣
      ↪random_state = 0)
```

6 Model Cross-validation

```
[ ]: randomState = 0

     Regressors = []

     # Regressors.append(MLPRegressor(random_state = randomState, activation =␣
      ↪"relu", solver = "adam",
     #                         hidden_layer_sizes = (100,), alpha = 0.0001,␣
      ↪learning_rate = "constant",
     #                         learning_rate_init = 0.0005, max_iter = 5000))

     Regressors.append(MLPRegressor(random_state = randomState, activation = "relu",␣
      ↪solver = "adam",
                              hidden_layer_sizes = (100,), alpha = 0.0101, learning_rate␣
      ↪= "adaptive",
                              learning_rate_init = 0.1, max_iter = 1000))

     Regressors.append(KNeighborsRegressor(n_neighbors = 2, weights = "uniform", p =␣
      ↪1))

     Regressors.append(DecisionTreeRegressor(random_state=randomState))

     Regressors.append(RandomForestRegressor(n_estimators = 250, max_depth = 7,␣
      ↪random_state=randomState))

     Regressors.append(SVR(C = 40.7, epsilon=0.56))

     Regressors.append(NuSVR(C = 31.2, nu=0.5))

     Regressors.append(AdaBoostRegressor(random_state = randomState))

     Regressors.append(GradientBoostingRegressor(random_state = randomState))

     Regressors.append(GaussianProcessRegressor(random_state = randomState))

     Regressors.append(HistGradientBoostingRegressor(random_state = randomState))

     cv_results = []
     cv_train_score = []
     for regressor in Regressors:
```

```
    cv_train=regressor.fit(x_train, y = y_train)
    cv_train_score.append(regressor.score(x_train, y_train))
    cv_results.append(regressor.score(x_test, y_test))

cv_res = pd.DataFrame({"Algorithm":["NN", "KN","Decision Tree","Random␣
 ↪Forest","SVR","NuSVR","AdaBoost","GradientBoost", "GaussianProcess",␣
 ↪"HistGradientBoosting"],
                       "TrainScore":cv_train_score, "TestScore":cv_results})
cv_res.sort_values(by = "TestScore", ascending = False)
```

```
[ ]:            Algorithm  TrainScore  TestScore
     7       GradientBoost    0.971574   0.909959
     3       Random Forest    0.952958   0.854074
     0                  NN    0.897917   0.823250
     9  HistGradientBoosting  0.958757   0.808177
     1                  KN    0.929634   0.794065
     2       Decision Tree    1.000000   0.778517
     4                 SVR    0.913308   0.746149
     5               NuSVR    0.904299   0.719656
     6            AdaBoost    0.862871   0.644497
     8     GaussianProcess    1.000000  -0.304075
```

7 Mejora del modelo

Mejoramos el modelo que nos dio más precisión en el test set (GradientBoost) aplicando un tuning de hiperparámetros con ayuda de RandomizedSearchCV

```
[ ]: model = GradientBoostingRegressor()
     parameters = {'learning_rate': sp_randFloat(),
                   'subsample'    : sp_randFloat(),
                   'n_estimators' : sp_randInt(50,1000),
                   'max_depth'    : sp_randInt(2,10)
                   }

     random = RandomizedSearchCV(estimator=model, param_distributions=parameters, cv␣
      ↪= 2, n_jobs=-1)
     random.fit(x_train, y_train)
```

```
[ ]: RandomizedSearchCV(cv=2, estimator=GradientBoostingRegressor(), n_jobs=-1,
                        param_distributions={'learning_rate':
     <scipy.stats._distn_infrastructure.rv_frozen object at 0x7faf5dc578d0>,
                                             'max_depth':
     <scipy.stats._distn_infrastructure.rv_frozen object at 0x7faf5dc76b50>,
                                             'n_estimators':
     <scipy.stats._distn_infrastructure.rv_frozen object at 0x7faf5dc76e90>,
                                             'subsample':
     <scipy.stats._distn_infrastructure.rv_frozen object at 0x7faf5dc57e10>})
```

[ ]:

```
random.best_estimator_
```

[ ]:

```
GradientBoostingRegressor(learning_rate=0.03039604699664189, n_estimators=420,
                          subsample=0.8699095805674916)
```

[ ]:

```
random.score(x_test,y_test)
```

[ ]:

```
0.911738833344591
```

[ ]:

```
random.score(x_train,y_train)
```

[ ]:

```
0.9801051973743056
```

8 Visualización

[ ]:

```
pred = {'Real Value':y_test, 'Prediction': random.predict(x_test)}
predictions_df = pd.DataFrame(pred)

plt.plot(predictions_df['Real Value'])
plt.plot(predictions_df.Prediction)

plt.legend(["Real Value", "Prediction"], loc=0)
plt.show()
```