Introduction:scikit-learn_Learn regression on Boston dataset_Understand train/test splits_Preprocess data_Compose pipelines_Save and load models

In [1]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [2]:
```
type(boston)
```

Out[2]:
```
sklearn.utils.Bunch
```

In [3]:
```
boston.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [4]:
```
type(boston['data'])
```

Out[4]:
```
numpy.ndarray
```

In [5]:
```
boston['feature_names']
```

Out[5]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [6]:
```
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 (a
ttribute 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/ (https://a
rchive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carneg
ie 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 tha
t 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, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [7]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [8]:
```
clf = RandomForestRegressor()
```

In [9]:
```
clf.fit(boston['data'], boston['target'])
```

Out[9]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [10]:
```
clf.score(boston['data'], boston['target'])
```

Out[10]:
```
0.9829396479746284
```

In [11]:
```
clf.score?
```

In [12]:
```
dir(clf)
```

Out[12]:
```
['__abstractmethods__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_impl',
 '_estimator_type',
 '_get_param_names',
 '_get_tags',
 '_make_estimator',
 '_more_tags',
 '_required_parameters',
 '_set_oob_score',
 '_validate_X_predict',
 '_validate_estimator',
 '_validate_y_class_weight',
 'apply',
 'base_estimator',
 'base_estimator_',
 'bootstrap',
 'ccp_alpha',
 'class_weight',
 'criterion',
 'decision_path',
 'estimator_params',
 'estimators_',
 'feature_importances_',
```

```
 'fit',
 'get_params',
 'max_depth',
 'max_features',
 'max_leaf_nodes',
 'max_samples',
 'min_impurity_decrease',
 'min_impurity_split',
 'min_samples_leaf',
 'min_samples_split',
 'min_weight_fraction_leaf',
 'n_estimators',
 'n_features_',
 'n_jobs',
 'n_outputs_',
 'oob_score',
 'predict',
 'random_state',
 'score',
 'set_params',
 'verbose',
 'warm_start']
```

In [13]:
```
clf.n_features_
```

Out[13]:
```
13
```

In [14]:
```
boston['data'].shape
```

Out[14]:
```
(506, 13)
```

In [15]:
```
row = boston['data'][17]
row.shape
```

Out[15]:
```
(13,)
```

In [16]:
```
row.reshape(-1,13)
```

Out[16]:
```
array([[  0.7842,   0.    ,   8.14  ,   0.    ,   0.538 ,   5.99  ,
         81.7   ,   4.2579,   4.    , 307.    ,  21.    , 386.75  ,
         14.67  ]])
```

In [17]:
```
clf.predict(row.reshape(-1,13))
```

Out[17]:
```
array([18.056])
```

In [18]:
```
boston['target'][17]
```

Out[18]:
```
17.5
```

In [19]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(boston['data'], boston['targe
```

In [21]:
```
clf = RandomForestRegressor()
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[21]:
```
0.819897754977189
```

In [22]:
```
import pandas as pd
```

In [23]:
```
df = pd.DataFrame(boston['data'], columns=boston['feature_names'])
df.max(axis=0)
```

Out[23]:
```
CRIM        88.9762
ZN         100.0000
INDUS       27.7400
CHAS         1.0000
NOX          0.8710
RM           8.7800
AGE        100.0000
DIS         12.1265
RAD         24.0000
TAX        711.0000
PTRATIO     22.0000
B          396.9000
LSTAT       37.9700
dtype: float64
```

In [25]:
```
from sklearn.svm import SVR
```

In [26]:
```
clf = SVR()
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[26]:
```
0.12996136281434778
```

In [27]:
```
from sklearn import preprocessing
Xs  = preprocessing.scale(boston['data'])
```

In [29]:
```
df = pd.DataFrame(Xs, columns=boston['feature_names'])
df.max(axis=0)
```

Out[29]:
```
CRIM       9.933931
ZN         3.804234
INDUS      2.422565
CHAS       3.668398
NOX        2.732346
RM         3.555044
AGE        1.117494
DIS        3.960518
RAD        1.661245
TAX        1.798194
PTRATIO    1.638828
B          0.441052
LSTAT      3.548771
dtype: float64
```

In [30]:
```
Xs_train, Xs_test, ys_train, ys_test = train_test_split(Xs, boston['target'], tes
```

In [31]:
```
clf = SVR()
clf.fit(Xs_train, ys_train)
clf.score(Xs_test, ys_test)
```

Out[31]:
```
0.5791336312320645
```

In [32]:
```
from sklearn.decomposition import PCA
```

In [34]:
```
pca = PCA(n_components=5)
pca.fit(boston['data'])
```

Out[34]:
```
PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [35]:
```
Xp = pca.transform(boston['data'])
Xp.shape
```

Out[35]:
```
(506, 5)
```

In [38]:
```
clf = RandomForestRegressor()
Xp_train, Xp_test, yp_train, yp_test = train_test_split(Xp, boston['target'], tes
clf.fit(Xp_train, yp_train)
clf.score(Xp_test, yp_test)
```

Out[38]:
```
0.5611805760435045
```

In [40]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [41]:
```
pipe = Pipeline([
    ('scale', StandardScaler()),
    ('pca', PCA(n_components=5)),
    ('svr', SVR()),
])
```

In [42]:
```
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Out[42]:
```
0.5470623790288851
```

In [43]:
```
pipe.steps
```

Out[43]:
```
[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('pca',
  PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
      svd_solver='auto', tol=0.0, whiten=False)),
 ('svr',
  SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
      kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False))]
```

In [44]:
```
pipe.get_params()
```

Out[44]:
```
{'memory': None,
 'steps': [('scale', StandardScaler(copy=True, with_mean=True, with_std=True)),
  ('pca',
   PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
       svd_solver='auto', tol=0.0, whiten=False)),
  ('svr',
   SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
       kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False))],
 'verbose': False,
 'scale': StandardScaler(copy=True, with_mean=True, with_std=True),
 'pca': PCA(copy=True, iterated_power='auto', n_components=5, random_state=Non
e,
    svd_solver='auto', tol=0.0, whiten=False),
 'svr': SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='sca
le',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
 'scale__copy': True,
 'scale__with_mean': True,
 'scale__with_std': True,
 'pca__copy': True,
 'pca__iterated_power': 'auto',
 'pca__n_components': 5,
 'pca__random_state': None,
 'pca__svd_solver': 'auto',
 'pca__tol': 0.0,
 'pca__whiten': False,
 'svr__C': 1.0,
 'svr__cache_size': 200,
 'svr__coef0': 0.0,
 'svr__degree': 3,
 'svr__epsilon': 0.1,
 'svr__gamma': 'scale',
 'svr__kernel': 'rbf',
 'svr__max_iter': -1,
 'svr__shrinking': True,
 'svr__tol': 0.001,
 'svr__verbose': False}
```

In [45]:
```
pipe.set_params(svr__C=0.9)
```

Out[45]:
```
Pipeline(memory=None,
         steps=[('scale',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('pca',
                 PCA(copy=True, iterated_power='auto', n_components=5,
                     random_state=None, svd_solver='auto', tol=0.0,
                     whiten=False)),
                ('svr',
                 SVR(C=0.9, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
                     gamma='scale', kernel='rbf', max_iter=-1, shrinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
```

In [46]:
```
import pickle
```

In [47]:
```
with open('model.pickle', 'wb') as out:
    pickle.dump(pipe, out)
```

In [48]:
```
with open('model.pickle', 'rb') as fp:
    pipel = pickle.load(fp)
```

In [49]:
```
pipel.steps
```

Out[49]:
```
[('scale', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('pca',
  PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
      svd_solver='auto', tol=0.0, whiten=False)),
 ('svr',
  SVR(C=0.9, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
      kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False))]
```

In [50]:
```
pipel.score(X_test, y_test)
```

Out[50]:
```
0.5470623790288851
```