Week 3

March 14, 2022

0.1 Linear Regression - for continuous values

[65]:
```
#load the boston dataset
from sklearn.datasets import load_boston
```

[66]:
```
boston = load_boston()
```

```
/home/andropov/anaconda3/envs/imp/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is
deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::
```

```
        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

[3]:
```python
#see description of dataset
boston.DESCR
```

[3]:
```
".. _boston_dataset:\n\nBoston house prices
dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n
:Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical
predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute
Information (in order):\n        - CRIM     per capita crime rate by town\n
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n
- INDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        -
NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000\n
- PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2
where Bk is the proportion of black people by town\n        - LSTAT    % lower
status of the population\n        - MEDV     Median value of owner-occupied
homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator:
Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing
dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-
databases/housing/\n\n\nThis dataset was taken from the StatLib library which is
maintained at Carnegie Mellon University.\n\nThe Boston house-price data of
Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air',
J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh
& Welsch, 'Regression diagnostics\n…', Wiley, 1980.   N.B. Various
transformations are used in the table on\npages 244-261 of the latter.\n\nThe
Boston house-price data has been used in many machine learning papers that
address regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsley,
Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
of Collinearity', Wiley, 1980. 244-261.\n   - 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.\n"
```

[67]:
```python
import pandas as pd
```

[68]:
```python
data=pd.DataFrame(boston.data, columns=boston.feature_names)
```

[69]:
```
#see data in dataframe
data
```

[69]:
```
        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
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       …       …     …
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
```

[70]:
```
#column of Median Value is usually the target - to be predicted by regression␣
↪model
data['MEDV']=pd.DataFrame(boston.target)
```

[71]:
```
data
```

[71]:
```
        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]
```

[20]:
```python
#find all correlation values, RM has highest with target MEDV, so select RM for training
pd.DataFrame(data.corr().round(2))
```

[20]:
```
         CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX  PTRATIO  \
CRIM     1.00 -0.20   0.41 -0.06  0.42 -0.22  0.35 -0.38  0.63  0.58     0.29
ZN      -0.20  1.00  -0.53 -0.04 -0.52  0.31 -0.57  0.66 -0.31 -0.31    -0.39
INDUS    0.41 -0.53   1.00  0.06  0.76 -0.39  0.64 -0.71  0.60  0.72     0.38
CHAS    -0.06 -0.04   0.06  1.00  0.09  0.09  0.09 -0.10 -0.01 -0.04    -0.12
NOX      0.42 -0.52   0.76  0.09  1.00 -0.30  0.73 -0.77  0.61  0.67     0.19
RM      -0.22  0.31  -0.39  0.09 -0.30  1.00 -0.24  0.21 -0.21 -0.29    -0.36
AGE      0.35 -0.57   0.64  0.09  0.73 -0.24  1.00 -0.75  0.46  0.51     0.26
DIS     -0.38  0.66  -0.71 -0.10 -0.77  0.21 -0.75  1.00 -0.49 -0.53    -0.23
RAD      0.63 -0.31   0.60 -0.01  0.61 -0.21  0.46 -0.49  1.00  0.91     0.46
TAX      0.58 -0.31   0.72 -0.04  0.67 -0.29  0.51 -0.53  0.91  1.00     0.46
PTRATIO  0.29 -0.39   0.38 -0.12  0.19 -0.36  0.26 -0.23  0.46  0.46     1.00
B       -0.39  0.18  -0.36  0.05 -0.38  0.13 -0.27  0.29 -0.44 -0.44    -0.18
LSTAT    0.46 -0.41   0.60 -0.05  0.59 -0.61  0.60 -0.50  0.49  0.54     0.37
MEDV    -0.39  0.36  -0.48  0.18 -0.43  0.70 -0.38  0.25 -0.38 -0.47    -0.51

            B  LSTAT  MEDV
CRIM    -0.39   0.46 -0.39
ZN       0.18  -0.41  0.36
INDUS   -0.36   0.60 -0.48
CHAS     0.05  -0.05  0.18
NOX     -0.38   0.59 -0.43
RM       0.13  -0.61  0.70
AGE     -0.27   0.60 -0.38
DIS      0.29  -0.50  0.25
```

```
RAD     -0.44   0.49 -0.38
TAX     -0.44   0.54 -0.47
PTRATIO -0.18   0.37 -0.51
B        1.00  -0.37  0.33
LSTAT   -0.37   1.00 -0.74
MEDV     0.33  -0.74  1.00
```

[21]:
```
# x corresponds to train data
x=data['RM']
```

[22]:
```
#y corresponds to labels
y=data['MEDV']
```

[73]:
```
#import module for linear regression
from sklearn.linear_model import LinearRegression
```

[75]:
```
#import train_test_split module
from sklearn.model_selection import train_test_split
```

[74]:
```
linearRegressionClassifier = LinearRegression()
```

[28]:
```
#convert x and y to pandas Dataframes
x=pd.DataFrame(x)
y=pd.DataFrame(y)
```

[76]:
```
#split the dataset using train_test_split function, pass train data, labels,␣
 ↪and test data ratio
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

[77]:
```
#check number of rows and columns
print(x_train.shape)
```

```
(404, 1)
```

[78]:
```
#train the classifier by fitting to train data and corresponding labels
linearRegressionClassifier.fit(x_train, y_train)
```

[78]:
```
LinearRegression()
```

[34]:
```
import numpy as np
```

[79]:
```
#import the mean squared error module
from sklearn.metrics import mean_squared_error
```

[80]:
```
#determine the predicted values
y_pred=linearRegressionClassifier.predict(x_test)
```

[82]:
```
y_pred.shape
```

[82]: (102, 1)

[83]:
```
#calculate root mean square error
np.sqrt(mean_squared_error(y_test, y_pred))
```

[83]: 6.342458820150728

[84]:
```
#determine R^2 scores (because regression)
linearRegressionClassifier.score(x_test,y_test)
```

[84]: 0.47919078205973686

0.2 working with Ridge

[ ]:
```
#L2 Regularization: It adds an L2 penalty which is equal to the square of the␣
↪magnitude of coefficients.
```

[41]:
```
#importing module to work with ridge regression model
from sklearn.linear_model import Ridge
```

[85]:
```
#setting parameter alpha for new ridge model
ridge1=Ridge(alpha=1)
```

[86]:
```
ridge1.fit(x_train,y_train)
```

[86]: Ridge(alpha=1)

[88]:
```
y_pred1=ridge1.predict(x_test)
```

[89]:
```
np.sqrt(mean_squared_error(y_test, y_pred1))
```

[89]: 6.341851705830557

[46]:
```
ridge2=Ridge(alpha=100)
```

[47]:
```
ridge2.fit(x_train,y_train)
```

[47]: Ridge(alpha=100)

[48]:
```
y_pred2=ridge2.predict(x_test)
```

[97]:
```
np.sqrt(mean_squared_error(y_test, y_pred2))
```

[97]: 9.28858567055522

[90]:
```
ridge2.score(x_test,y_test)
```

[90]: 0.4264608073819258

1 # Working with Lasso

[ ]:
```
#L1 regularization: It adds an L1 penalty that is equal to the
#absolute value of the magnitude of coefficient, or simply restricting the size␣
↪of coefficients.
```

[91]:
```
from sklearn.linear_model import Lasso
```

[92]:
```
Lasso1=Lasso(alpha=0.01)
```

[94]:
```
Lasso1.fit(x_train,y_train)
```

[94]: Lasso(alpha=0.01)

[95]:
```
y_predL1=Lasso1.predict(x_test)
```

[96]:
```
np.sqrt(mean_squared_error(y_test, y_predL1))
```

[96]: 6.342170750650113

[98]:
```
Lasso1.score(x_test,y_test)
```

[98]: 0.4792380904770168

[ ]:
```
#working with ElasticNet - combines feature elimination from lasso and feature␣
↪coefficient reduction from Ridge
```

[60]:
```
from sklearn.linear_model import ElasticNet
```

[99]:
```
en1=ElasticNet(alpha=0.1, l1_ratio=0.5)
```

[100]:
```
en1.fit(x_train,y_train)
```

[100]: ElasticNet(alpha=0.1)

[101]:
```
y_pred_en1=en1.predict(x_test)
```

[102]:
```
np.sqrt(mean_squared_error(y_test, y_pred_en1))
```

[102]: 6.359665176837157

[103]:
```
en1.score(x_test,y_test)
```

[103]: 0.4763611587818619

```
[ ]:
```