Implementation of Linear Regression in Python

Deep into MLF

Weiguan Wang
Shanghai University

Overview

1. Sklearn
2. Statsmodels

Two popular packages

Various Python packages have implemented linear regression model. The two most commonly used are

- Scikit-Learn (sklearn): sklearn.linear_model.LinearRegression
- statsmodel: statsmodels.api.OLS

The former, sklearn, is more machine learning oriented, while the latter is more statistics oriented. This is shown by the fact that the former only provides R^2 for evaluating the performance of fitting, while the latter gives more statistical testing measures.

Sklearn

In this course, we will carry out experiments based on Scikit-learn package. Although statsmodel seems more powerful in terms of linear model, it is very limited in machine learning models and related methodology.

Recall that a linear model $y = f_\theta(x) + \varepsilon = x\theta + \varepsilon$ is fitted by minimizing loss function $L(\theta|X, Y) = (Y - X\theta)^T(Y - X\theta)$, i.e. the sum of squared error between target y and predicted value $\hat{y}$.

As an example, we wish to fit a linear model $y = \theta_1 x_1 + \theta_2 x_2 + b$ on the dataset (X, Y) where

$$X = \begin{bmatrix} 1.5 & 2. \\ 0.5 & 1. \\ -1. & 0.3 \\ 2. & -1. \end{bmatrix}, \qquad Y = \begin{bmatrix} -0.11 \\ 0.38 \\ 0.56 \\ -1.52 \end{bmatrix}$$

Here the target Y is generated with the true model $y = -0.5x_1 + 0.5x_2 + \varepsilon$, where $\varepsilon \in \mathbb{N}(0, 0.1)$.

The following Python code generates the dataset (X, Y) with random seed 12345.

```python
import numpy as np
X = np.array([[1.5, 2.0], [0.5, 1.0], [-1, 0.3], [2, -1]])
coef = np.array([-0.5, 0.5])
rng = np.random.default_rng(12345)
Y = X@coef + rng.standard_normal([4]) * 0.1
```

To fit a linear model, we first import and instantiate `LinearRegression` object. The fitting is simply calling the object's `fit` method.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, Y)
```

The model coefficients and intercept from the estimation are stored as the object's attribute `coef_` and `intercept_`. To check the model performance, we may print the coefficient of determination R^2 which measures the goodness of fitting, by calling `score()` method.

```
print(f'The coefficients of the model are: {model.coef_.round(2)}')
print(f'The intercept of the model is: {model.intercept_.round(2)}')
print(f'The coefficient of determination is {model.score(X, Y).round(2)}')
print(f'The predicted values are {model.predict(X).round(2)}')
```

The output is

```
The coefficients of the model are: [-0.51  0.48]
The intercept of the model is: -0.02
The coefficient of determination is 0.99
The predicted values are [ 0.19  0.21  0.63 -1.51]
```

Statsmodels

As an alternative, one may also use `statsmodels` to fit a linear model. The sytax is similar, however it offers a lot more diagnostics after fitting.

```
import statsmodels.api as sm
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
```

The output is

```
                             OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.986
Model:                            OLS   Adj. R-squared (uncentered):              0.972
Method:                 Least Squares   F-statistic:                              70.04
Date:                Sat, 04 Jun 2022   Prob (F-statistic):                      0.0141
Time:                        10:20:11   Log-Likelihood:                          3.5650
No. Observations:                   4   AIC:                                     -3.130
Df Residuals:                       2   BIC:                                     -4.357
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.5116      0.052     -9.823      0.010      -0.736      -0.287
x2             0.4762      0.058      8.241      0.014       0.228       0.725
==============================================================================
Omnibus:                          nan   Durbin-Watson:                   3.044
Prob(Omnibus):                    nan   Jarque-Bera (JB):                0.734
Skew:                           0.962   Prob(JB):                        0.693
Kurtosis:                       2.160   Cond. No.                         1.23
==============================================================================
```

Thanks for your attention!

References I