

# ML5-mLR-LectureML5-Example

August 30, 2020

## 1 multiple Linear Regression (mLR) with scikit-learn (Example for lesson ML05)

Powered by: Dr. Hermann Völlinger, DHBW Stuttgart(Germany); July 2020

Following ideas from: “Linear Regression in Python” by Mirko Stojiljkovic, 28.4.2020 (see details: <https://realpython.com/linear-regression-in-python/#what-is-regression>)

You can obtain the properties of the model the same way as in the case of simple linear regression: The example is from Lecture: “ML\_Concept&Algorithm” (WS2020); Chapter ML5, Slide with title: “Definition of the “Adjusted R-Squared” Measure: Adj.R<sup>2</sup>”

So let’s start with the next level of Linear Regression, which is multiple Linear Regression (mLR). There are five basic steps when you’re implementing linear regression:

1. Import the packages and classes you need.
2. Provide data to work with and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions. These steps are more or less general for most of the regression approaches and implementations.

### 1.1 Steps 1 and 2: Import packages and classes, and provide data

First, you import numpy and sklearn.linear\_model.LinearRegression and provide known inputs and output:

```
[1]: # First, you import numpy and sklearn.linear_model.LinearRegression and  
# provide known inputs and output.  
  
import numpy as np  
from sklearn.linear_model import LinearRegression  
  
# That's a simple way to define the input x and output y.  
  
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]  
y = [4, 5, 20, 14, 32, 22, 38, 43]  
x, y = np.array(x), np.array(y)
```

```
[2]: # You can print x and y to see how they look now:
```

```
print('x looks like: ', x)
print('y looks like: ', y)
```

```
x looks like:  [[ 0  1]
 [ 5  1]
 [15  2]
 [25  5]
 [35 11]
 [45 15]
 [55 34]
 [60 35]]
y looks like:  [ 4  5 20 14 32 22 38 43]
```

## 1.2 Step 3: Create a model and fit it

The next step is to create the regression model as an instance of `LinearRegression` and fit it with `.fit()`:

```
[3]: # The result of this statement is the variable model referring to the object of LinearRegression.
      ↪ It represents the regression model fitted with existing data.

model = LinearRegression().fit(x, y)
```

## 1.3 Step 4: Get results

You can obtain the properties of the model the same way as in the case of simple linear regression:

```
[4]: r_sq = model.score(x, y)
      print('coefficient of determination:', r_sq)
      print('intercept:', model.intercept_)
      print('coefficients:', model.coef_)
```

```
coefficient of determination: 0.8615939258756776
intercept: 5.52257927519819
coefficients: [0.44706965 0.25502548]
```

You obtain the value of  $r^2$  using `.score()` and the values of the estimators of regression coefficients with `.intercept_` and `.coef_`. Again, `.intercept_` holds the bias term, while now `.coef_` is an array containing  $\beta_0$  and  $\beta_1$  respectively.

In this example, the intercept is approximately 5.52, and this is the value of the predicted response when  $x_1 = x_2 = 0$ . The increase of  $x_1$  by 1 yields the rise of the predicted response by 0.45. Similarly, when  $x_2$  grows by 1, the response rises by 0.26.

## 1.4 Step 5: Predict response

Predictions also work the same way as in the case of simple linear regression: The predicted response is obtained with `.predict()`

```
[5]: y_pred = model.predict(x)
      print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[ 5.77760476  8.012953  12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

You can predict the output values also (which is similar to run “`.predict()`”), by multiplying each column of the input with the appropriate weight, summing the results and adding the intercept to the sum.

```
[6]: y_pred = model.intercept_ + np.sum(model.coef_ * x, axis=1)
      print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[ 5.77760476  8.012953  12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

You can apply this model to new data as well:

```
[7]: x_new = np.arange(10).reshape((-1, 2))
      print(x_new)
      y_new = model.predict(x_new)
      print(y_new)
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
[ 5.77760476  7.18179502  8.58598528  9.99017554 11.3943658 ]
```

```
[8]: # print current date and time
      import time
      print("***** Current Date and Time ***** ")
      print("Date and Time: ",time.strftime("%d.%m.%Y %H:%M:%S"))
      print("End")
```

```
***** Current Date and Time *****
Date and Time:  30.08.2020 23:36:13
End
```