# ML5-Homework-H5_4c

August 11, 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); Homework 5.4 part c:"mLR (k=2) manual calculations of Adj.$R^2$ & Jupyter Notebook (Python) to check results"

So let's start with the next level of Linear Regression, which is multipe 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 = [[1, 2], [3, 3], [2, 2], [4, 3]]
y = [3, 4, 4, 6]
x, y = np.array(x), np.array(y)
```

[2]:

```python
# 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:  [[1 2]
 [3 3]
 [2 2]
 [4 3]]
y looks like:  [3 4 4 6]
```

## 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]:

```python
# The result of this statement is the variable model referring to the object of␣
 ↪type 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]:

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

```
coefficient of determination: 0.9473684210526315
intercept: 4.25
coefficients: [ 1.5 -1.5]
```

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

In this example, the intercept is approximately 4.25, and this is the value of the predicted response when = = 0. The increase of by 1 yields the rise of the predicted response by 1.5. Similarly, when grows by 1, the response declined by -1.5.

## 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:
[2.75 4.25 4.25 5.75]
```

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:
[2.75 4.25 4.25 5.75]
```

[7]:
```
# print current date and time
import time
print("date",time.strftime("%d.%m.%Y %H:%M:%S"))
print ("end")
```

```
date 11.08.2020 00:20:04
end
```