

Machine learning: linear regression

[Exercise 10 \(linear regression\)](#)[Exercise 11 \(mystery data\)](#)[Exercise 12 \(coefficient of determination\)](#)[Exercise 13 \(cycling weather continues\)](#)

Machine learning: linear regression

Linear regression

Regression analysis tries to explain relationships between variables. One of these variables, called dependent variable, is what we want to "explain" using one or more *explanatory variables*. In linear regression we assume that the dependent variable can be, approximately, expressed as a linear combination of the explanatory variables. As a simple example, we might have dependent variable height and an explanatory variable age. The age of a person can quite well explain the height of a person, and this relationship is approximately linear for kids (ages between 1 and 16). Another way of thinking about regression is fitting a curve to the observed data points. If we have only one explanatory variable, then this is easy to visualize, as we shall see below.

We can apply the linear regression easily with the [scikit-learn](#) package. Let's go through some examples.

First we make the usual standard imports.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn # This imports the scikit-learn library
```

Then we create some data with approximately the relationship

$y = 2x + 1$, with normally distributed errors.

```
np.random.seed(0)
n=20 # Number of data points
x=np.linspace(0, 10, n)
y=x*2 + 1 + 1*np.random.randn(n) # Standard deviation 1
print(x)
print(y)
```

```
[ 0.  0.52631579  1.05263158  1.57894737  2.10526316
 2.63157895  3.15789474  3.68421053  4.21052632  4.73684211
 5.26315789  5.78947368  6.31578947  6.84210526  7.36842105
 7.89473684  8.42105263  8.94736842  9.47368421 10. ]
[ 2.76405235  2.45278879  4.08400114  6.39878794  7.07808431
 5.28588001  8.26587789  8.21706384  9.31783378 10.88428271
11.67035936 14.03322088 14.39261667 14.80588554 16.18070534
17.12314801 19.33618434 18.68957858 20.26043612 20.14590426]
```

Next we import the LinearRegression class.

```
from sklearn.linear_model import LinearRegression
```

Now we can fit a line through the data points (x, y):

```
model=LinearRegression(fit_intercept=True)
model.fit(x[:,np.newaxis], y)
xfit=np.linspace(0,10,100)
yfit=model.predict(xfit[:, np.newaxis])
plt.plot(xfit,yfit, color="black")
plt.plot(x,y, 'o')
# The following will draw as many line segments as there are columns in matrices x and
plt.plot(np.vstack([x,x]), np.vstack([y, model.predict(x[:, np.newaxis])]), color="red")
```



The linear regression tries to minimize the sum of squared errors

$\sum_i (y[i] - \hat{y}[i])^2$; this is the sum of the squared lengths of the red line segments in the above plot. The estimated values $\hat{y}[i]$ are denoted by `yfit[i]` in the above code.

```
print("Parameters:", model.coef_, model.intercept_)
print("Coefficient:", model.coef_[0])
print("Intercept:", model.intercept_)
```

```
Parameters: [ 1.88627741] 2.13794752053
Coefficient: 1.88627741448
Intercept: 2.13794752053
```

In this case, the coefficient is the slope of the fitted line, and the intercept

is the point where the fitted line intersects with the y-axis.

The parameters estimated by the regression algorithm were quite close to the parameters that generated the data: coefficient 2 and intercept 1. Try experimenting with the number of data points and/or the standard deviation, to see if you can improve the estimated parameters.

Multiple features

The previous example had only one explanatory variable. Sometimes this is called a *simple linear regression*. The next example illustrates a more complex regression with multiple explanatory variables.

```
sample1=np.array([1,2,3]) # The three explanatory variables have values 1, 2, and 3,
sample2=np.array([4,5,6]) # Another example of values of explanatory variables
sample3=np.array([7,8,10]) # ...
y=np.array([15,39,66]) + np.random.randn(3) # For values 1,2, and 3 of explanatory v
```

Let's try to fit a linear model to these points:

```
model2=LinearRegression(fit_intercept=False)
x=np.vstack([sample1,sample2,sample3])
model2.fit(x, y)
model2.coef_, model2.intercept_
```

```
(array([ 5.69493795e+00,  3.36972233e+00,  4.20919214e-03]), 0
```

Let's print the various components involved.

```
b=model2.coef_[ :, np.newaxis]
print("x:\n", x)
print("b:\n", b)
print("y:\n", y[:, np.newaxis])
print("product:\n", np.matmul(x, b))
```

```
x:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8 10]]
b:
[[ 5.69493795e+00]
 [ 3.36972233e+00]
 [ 4.20919214e-03]]
y:
[[ 12.44701018]
 [ 39.6536186 ]
 [ 66.8644362 ]]
product:
```

```
[[ 12.44701018]
 [ 39.6536186 ]
 [ 66.8644362 ]]
```

Polynomial regression

It may perhaps come as a surprise that one can fit a polynomial curve to data points using linear regression. The trick is to add new explanatory variables to the model. Below we have a single feature x with associated y values given by third degree polynomial, with some (gaussian) noise added. It is clear from the below plot that we cannot explain the data well with a linear function. We add two new features: x^2 and x^3 . Now the model has three explanatory variables, x , x^2 and x^3 . The linear regression will find the coefficients for these variables.

```
x=np.linspace(-50,150,50)
y=0.15*x**3 - 20*x**2 + 5*x - 4 + 5000*np.random.randn(50)
plt.scatter(x, y, color="black")
model_linear=LinearRegression(fit_intercept=True)
model_squared=LinearRegression(fit_intercept=True)
model_cubic=LinearRegression(fit_intercept=True)
x2=x**2
x3=x**3
model_linear.fit(np.vstack([x]).T, y)
model_squared.fit(np.vstack([x,x2]).T, y)
model_cubic.fit(np.vstack([x,x2,x3]).T, y)
xf=np.linspace(-50,150, 50)
yf_linear=model_linear.predict(np.vstack([x]).T)
yf_squared=model_squared.predict(np.vstack([x,x2]).T)
yf_cubic=model_cubic.predict(np.vstack([x,x2,x3]).T)
plt.plot(xf,yf_linear, label="linear")
plt.plot(xf,yf_squared, label="squared")
plt.plot(xf,yf_cubic, label="cubic")
plt.legend()
print("Coefficients:", model_cubic.coef_)
print("Intercept:", model_cubic.intercept_)
```

Coefficients: [-36.65414588 -20.17228669 0.15359003]

Intercept: -167.160466064



Linear and squared are not enough to explain the data, but the linear regression manages to fit quite well a polynomial curve to the data points, when cubic variables are included!

Exercise 10 (linear regression)

This exercise can give two points at maximum!

Part 1.

Write a function `fit_line` that gets one dimensional arrays `x` and `y` as parameters. The function should return the tuple (`slope`, `intercept`) of the fitted line. Write a main program that tests the `fit_line` function with some example arrays. The main function should produce output in the following form:

```
Slope: 1.0
Intercept: 1.166666666667
```

Part 2.

Modify your main function to plot the fitted line using `matplotlib`, in addition to the textual output. Plot also the original data points.

Exercise 11 (mystery data)

Read the tab separated file `mystery_data.tsv`. Its first five columns define the features, and the last column is the response. Use `scikit-learn`'s `LinearRegression` to fit this data. Implement function `mystery_data` that reads this file and learns and returns the regression coefficients for the five features. You don't have to fit the intercept. The `main` method should print output in the following form:

```
Coefficient of X1 is ...
Coefficient of X2 is ...
Coefficient of X3 is ...
Coefficient of X4 is ...
Coefficient of X5 is ...
```

Which features you think are needed to explain the response `Y`?

Exercise 12 (coefficient of determination)

This exercise can give two points at maximum!

Using the same data as in the previous exercise, instead of printing the regression coefficients, print the *coefficient of determination*. The coefficient of determination, denoted by R^2 , tells how well the linear regression fits the data. The maximum value of the coefficient of determination is 1. That means the best possible fit.

Part 1.

Using all the features (`X1` to `X5`), fit the data using a linear regression

(include the intercept). Get the coefficient of determination using the `score` method of the `LinearRegression` class. Write a function `coefficient_of_determination` to do all this. It should return a list containing the R2-score as the only value.

Part 2.

Extend your function so that it also returns R2-scores related to linear regression with each single feature in turn. The `coefficient_of_determination` (https://en.wikipedia.org/wiki/Coefficient_of_determination) function should therefore return a list with six R2-scores (the first score is for five features, like in Part 1). To achieve this, your function should call both the `fit` method and the `score` method six times.

The output from the main method should look like this:

```
R2-score with feature(s) X: ...
R2-score with feature(s) X1: ...
R2-score with feature(s) X2: ...
R2-score with feature(s) X3: ...
R2-score with feature(s) X4: ...
R2-score with feature(s) X5: ...
```

How small can the R2-score be? Experiment both with fitting the intercept and without fitting the intercept.

Exercise 13 (cycling weather continues)

Write function `cycling_weather_continues` that tries to explain with linear regression the variable of a cycling measuring station's counts using the weather data from corresponding day. The function should take the name of a (cycling) measuring station as a parameter and return the regression coefficients and the score. In more detail:

Read the weather data set from the `src` folder. Read the cycling data set from folder `src` and restrict it to year 2017. Further, get the sums of cycling counts for each day. Merge the two datasets by the year, month, and day. Note that for the above you need only small additions to the solution of exercise `cycling_weather`. After this, use forward fill to fill the missing values.

In the linear regression use as explanatory variables the following columns `'Precipitation amount (mm)'`, `'Snow depth (cm)'`, and

'Air temperature (degC)'. Explain the variable (measuring station), whose name is given as a parameter to the function `cycling_weather_continues`. Fit also the intercept. The function should return a pair, whose first element is the regression coefficients and the second element is the score. Above, you may need to use the method `reset_index` (its counterpart is the method `set_index`).

The output from the main function should be in the following form:

```
Measuring station: x
Regression coefficient for variable 'precipitation': x.x
Regression coefficient for variable 'snow depth': x.x
Regression coefficient for variable 'temperature': x.x
Score: x.xx
```

Use precision of one decimal for regression coefficients, and precision of two decimals for the score. In the main function test your solution using some measuring station, for example Baana.

Additional information

- The [scikit-learn](#) library concentrates on machine learning. Check out library [statsmodels](#) for a more statistical viewpoint to regression.

Summary (week 5)

- `pd.concat` and `pd.merge` can both combine two DataFrames, but the way the combining is done differs. The function `pd.concat` concatenates based on *indices* of DataFrames, whereas `pd.merge` combines based on the *content* of common variable(s).
- The option `join="outer"` to `pd.concat` can create missing values, but `join=inner` cannot. The former gives the union of indices and the latter gives the intersection of indices.
- With `pd.concat` overlapping indices can:
 - cause an error
 - cause renumbering of indices
 - create hierarchical indices
- Merging can join elements
 - one-to-one
 - one-to-many
 - many-to-many

- In grouping a DataFrame can be thought to be split into smaller DataFrames. The major classes of operations on these groups are:
 - aggregate
 - filter
 - transform (retains shape)
 - apply
- Series which are indexed by time are called time series
- Linear regression can be used to find out linear relationships between variables
 - can have more than one feature (explanatory variable)
 - fitting polynomials is still linear regression

You have reached the end of this section! Continue to the next section:

➔ 2. Pandas (continues)

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

1. Machine learning: linear regression

2. Pandas (continues)

This course is created by the Agile Education Research -research group [of the University of Helsinki](#).

[Credits and about the material.](#)



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI



MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI