Simple Linear Regression

Rafiq Islam

2024-08-29

Table of contents

# Simple Linear Regression

A simple linear regression in multiple predictors/input variables/features/independent variables/ explanatory variables/regressors/ covariates (many names) often takes the form

$$y=f\left(x\right)+ϵ=βx+ϵ$$

where $β\in R^{d}$ are regression parameters or constant values that we aim to estimate and $ϵ∼N\left(0,1\right)$ is a normally distributed error term independent of $x$ or also called the white noise.

In this case, the model:

$$y=f\left(x\right)+ϵ=β\_{0}+β\_{1}x+ϵ$$

Therefore, in our model we need to estimate the parameters $β\_{0},β\_{1}$. The true relationship between the explanatory variables and the dependent variable is $y=f\left(x\right)$. But our model is $y=f\left(x\right)+ϵ$. Here, this $f\left(x\right)$ is the working model with the data. In other words, $\hat{y}=f\left(x\right)=\hat{β}\_{0}+\hat{β}\_{1}x$. Therefore, there should be some error in the model prediction which we are calling $ϵ=∥y−\hat{y}∥$ where $y$ is the true value and $\hat{y}$ is the predicted value. This error term is normally distributed with mean 0 and variance 1. To get the best estimate of the parameters $β\_{0},β\_{1}$ we can minimize the error term as much as possible. So, we define the residual sum of squares (RSS) as:

Using multivariate calculus we see

Setting the partial derivatives to zero we solve for $\hat{β\_{0}},\hat{β\_{1}}$ as follows

and,

Therefore, we have the following

Simple Linear Regression slr is applicable for a single feature data set with contineous response variable.

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear\_model import LinearRegression

## Assumptions of Linear Regressions

* **Linearity:** The relationship between the feature set and the target variable has to be linear.
* **Homoscedasticity:** The variance of the residuals has to be constant.
* **Independence:** All the observations are independent of each other.
* **Normality:** The distribution of the dependent variable $y$ has to be normal.

## Synthetic Data

To implement the algorithm, we need some synthetic data. To generate the synthetic data we use the linear equation $y\left(x\right)=2x+\frac{1}{2}+ξ$ where $ξ∼N\left(0,1\right)$

X=np.random.random(100)
y=2\*X+0.5+np.random.randn(100)

Note that we used two random number generators, np.random.random(n) and np.random.randn(n). The first one generates $n$ random numbers of values from the range (0,1) and the second one generates values from the standard normal distribution with mean 0 and variance or standard deviation 1.

plt.figure(figsize=(9,6))
plt.scatter(X,y)
plt.xlabel('$X$')
plt.ylabel('y')
plt.gca().set\_facecolor('#f4f4f4')
plt.gcf().patch.set\_facecolor('#f4f4f4')
plt.show()



## Model

We want to fit a simple linear regression to the above data.

slr=LinearRegression()

Now to fit our data $X$ and $y$ we need to reshape the input variable. Because if we look at $X$,

X

array([0.30263204, 0.71911511, 0.60111679, 0.81207209, 0.51220523,
 0.13864203, 0.09487777, 0.59218777, 0.93430835, 0.90794118,
 0.35268946, 0.23657889, 0.84352341, 0.8012883 , 0.60809384,
 0.90604196, 0.50631382, 0.19307431, 0.17927141, 0.49153917,
 0.31321907, 0.93340072, 0.93489852, 0.47334186, 0.80583991,
 0.95604947, 0.38549975, 0.91528135, 0.47068498, 0.74111256,
 0.60744536, 0.06839393, 0.1610581 , 0.80911128, 0.39389842,
 0.83171313, 0.24267416, 0.67708592, 0.71577897, 0.64179481,
 0.35635554, 0.00870491, 0.77516491, 0.93913072, 0.47803778,
 0.38600407, 0.85890286, 0.94359959, 0.96053754, 0.5138371 ,
 0.76376814, 0.69781668, 0.12042369, 0.41784811, 0.27430382,
 0.11695504, 0.90234104, 0.71554777, 0.81765437, 0.61977452,
 0.58577465, 0.43940755, 0.34634601, 0.92553264, 0.01422604,
 0.36100593, 0.17912609, 0.55319236, 0.4421812 , 0.15117741,
 0.66477042, 0.8787667 , 0.75527606, 0.03721131, 0.51901777,
 0.23459943, 0.86390759, 0.25114361, 0.12860083, 0.60852062,
 0.64893646, 0.88720909, 0.91294491, 0.03151154, 0.53788969,
 0.63857043, 0.32694038, 0.50132148, 0.27392601, 0.57677026,
 0.13069802, 0.84053692, 0.09612515, 0.00766597, 0.65605718,
 0.92896946, 0.98512227, 0.08407923, 0.26704253, 0.42102619])

It is a one-dimensional array/vector but the slr object accepts input variable as matrix or two-dimensional format.

X=X.reshape(-1,1)
X[:10]

array([[0.30263204],
 [0.71911511],
 [0.60111679],
 [0.81207209],
 [0.51220523],
 [0.13864203],
 [0.09487777],
 [0.59218777],
 [0.93430835],
 [0.90794118]])

Now we fit the data to our model

slr.fit(X,y)
slr.predict([[2],[3]])

array([4.63116759, 6.70407594])

We have our $X=2,3$ and the corresponding $y$ values are from the above cell output, which are pretty close to the model $y=2x+\frac{1}{2}$.

intercept = round(slr.intercept\_,4)
slope = slr.coef\_

Now our model parameters are: intercept $β\_{0}=$ 0.4854 and slope $β\_{1}=$ array([2.07290834]).

plt.figure(figsize=(9,6))
plt.scatter(X,y, alpha=0.7,label="Sample Data")
plt.plot(np.linspace(0,1,100),
 slr.predict(np.linspace(0,1,100).reshape(-1,1)),
 'k',
 label='Model $\hat{f}$'
)
plt.plot(np.linspace(0,1,100),
 2\*np.linspace(0,1,100)+0.5,
 'r--',
 label='$f$'
)
plt.xlabel('$X$')
plt.ylabel('y')
plt.legend(fontsize=10)
plt.gca().set\_facecolor('#f4f4f4')
plt.gcf().patch.set\_facecolor('#f4f4f4')
plt.show()



So the model fits the data almost perfectly.

Up next [multiple linear regression](../../posts/multiplelinreg/index.qmd).

**Share on**

**You may also like**