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

where are regression parameters or constant values that we aim to estimate and is a normally distributed error term independent of or also called the white noise.

In this case, the model:

Therefore, in our model we need to estimate the parameters . The true relationship between the explanatory variables and the dependent variable is . But our model is . Here, this is the working model with the data. In other words, . Therefore, there should be some error in the model prediction which we are calling where is the true value and is the predicted value. This error term is normally distributed with mean 0 and variance 1. To get the best estimate of the parameters 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 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 has to be normal.

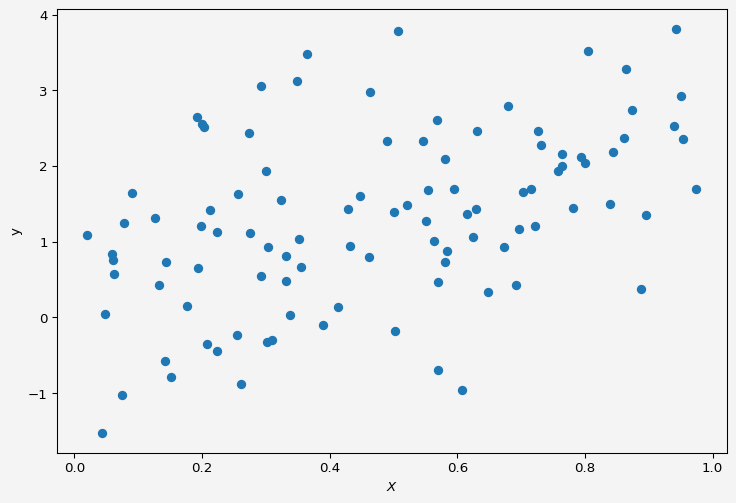
## Synthetic Data

To implement the algorithm, we need some synthetic data. To generate the synthetic data we use the linear equation where

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 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 and we need to reshape the input variable. Because if we look at ,

X

array([0.13192748, 0.27530538, 0.97421768, 0.8433925 , 0.56756411,  
 0.12629875, 0.69268034, 0.43231723, 0.30956587, 0.22386588,  
 0.14427525, 0.58318817, 0.22409484, 0.1417919 , 0.25596245,  
 0.67276394, 0.73056711, 0.36484496, 0.26084139, 0.86462418,  
 0.07532042, 0.46257951, 0.35591348, 0.21205794, 0.05997942,  
 0.6071526 , 0.7034083 , 0.95405751, 0.33168156, 0.55443977,  
 0.3234807 , 0.09076498, 0.72133235, 0.27370033, 0.62918338,  
 0.58039627, 0.33740262, 0.15220881, 0.07766889, 0.38913672,  
 0.34921629, 0.30350947, 0.72582633, 0.20359268, 0.83888612,  
 0.76363015, 0.6486986 , 0.19306764, 0.50057316, 0.29236597,  
 0.30217536, 0.44736167, 0.4294394 , 0.95113998, 0.02062629,  
 0.52197444, 0.58018347, 0.19866838, 0.35267808, 0.507251 ,  
 0.3003091 , 0.25427574, 0.46172496, 0.94253861, 0.49056938,  
 0.87405307, 0.71526649, 0.04733166, 0.50267264, 0.06246349,  
 0.56307785, 0.76361216, 0.33196235, 0.78081598, 0.20745899,  
 0.79979678, 0.56928079, 0.19211671, 0.63095776, 0.94000894,  
 0.59446273, 0.7938012 , 0.86140654, 0.0433311 , 0.55143528,  
 0.2928895 , 0.17731209, 0.57001821, 0.80475733, 0.05841778,  
 0.8953888 , 0.887658 , 0.62484985, 0.69645696, 0.61535276,  
 0.75741431, 0.2001698 , 0.41260239, 0.67928224, 0.54583253])

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([[0.13192748],  
 [0.27530538],  
 [0.97421768],  
 [0.8433925 ],  
 [0.56756411],  
 [0.12629875],  
 [0.69268034],  
 [0.43231723],  
 [0.30956587],  
 [0.22386588]])

Now we fit the data to our model

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

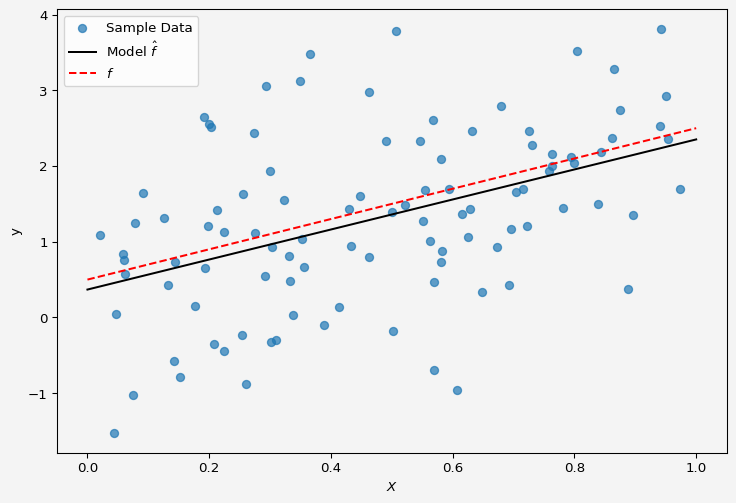
array([4.33443785, 6.317455 ])

We have our and the corresponding values are from the above cell output, which are pretty close to the model .

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

Now our model parameters are: intercept np.float64(0.3684) and slope array([1.98301715]).

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**