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([0.83030589, 0.09337814, 0.19862081, 0.10713303, 0.92236951,
 0.86090518, 0.78576671, 0.90970933, 0.05316094, 0.30686747,
 0.53265911, 0.58747794, 0.89209389, 0.35259571, 0.48741286,
 0.19826289, 0.15141734, 0.22466172, 0.97130649, 0.48014216,
 0.39148417, 0.94396327, 0.4102886 , 0.63048631, 0.16551924,
 0.38196676, 0.21095595, 0.60571522, 0.1501105 , 0.13173063,
 0.78068551, 0.69013612, 0.8123489 , 0.11183235, 0.51470456,
 0.15492827, 0.33363717, 0.72346986, 0.07831907, 0.15333528,
 0.77373734, 0.44761788, 0.09922492, 0.66680928, 0.90397711,
 0.63321308, 0.00819729, 0.12385383, 0.72839008, 0.51423758,
 0.64930657, 0.52497199, 0.78923778, 0.92893823, 0.48251971,
 0.36513817, 0.80569099, 0.58906652, 0.22513738, 0.93063588,
 0.45782933, 0.22387084, 0.52045241, 0.45604173, 0.99082333,
 0.61820533, 0.39628367, 0.89368898, 0.04877636, 0.79802206,
 0.53642724, 0.42229716, 0.72800304, 0.12197306, 0.55109157,
 0.93259241, 0.95819997, 0.41563526, 0.03112604, 0.26728891,
 0.98186441, 0.54521409, 0.03441729, 0.62324683, 0.74137821,
 0.14881611, 0.64757142, 0.34441568, 0.53811934, 0.88192235,
 0.72519274, 0.50474185, 0.09236632, 0.19769291, 0.99755547,
 0.46755014, 0.61311832, 0.07608745, 0.77433035, 0.99397757])

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([[0.83030589],
 [0.09337814],
 [0.19862081],
 [0.10713303],
 [0.92236951],
 [0.86090518],
 [0.78576671],
 [0.90970933],
 [0.05316094],
 [0.30686747]])

Now we fit the data to our model

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

array([3.94068357, 5.53661127])

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 $β\_{0}=$ 0.7488 and slope $β\_{1}=$ array([1.5959277]).

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](../../dsandml/multiplelinreg/index.qmd).

**Share on**

**You may also like**