Photo by Jeremy Bishop on Unsplash

Linear Regression Algorithm from Scratch in Python: Step by Step

Learn the concepts of linear regression and develop a complete linear regression algorithm from scratch in python


Rashida Nasrin Sucky Jul 4 · 5 min read ⋆

The most basic machine learning algorithm has to be the linear regression algorithm with a single variable. Nowadays, there are so many advanced machine learning algorithms, libraries, and techniques available that linear regression may seem to be not important. But It is always a good idea to learn the basics. That way you will grasp the concepts very clearly. In this article, I will explain the linear regression algorithm step by step.

Ideas and Formulas

Linear regression uses the very basic idea of prediction. Here is the formula:

$Y = C + BX$

We all learned this formula in school. Just to remind you, this is the equation of a straight line. Here, Y is the dependent variable, B is the slope and C is the intercept. Typically, for linear regression, it is written as:

$$h = \theta_0 + \theta_1 X$$

Here, 'h' is the hypothesis or the predicted dependent variable, X is the input feature, and theta0 and theta1 are the coefficients. Theta values are initialized randomly to start with. Then using gradient descent, we will update the theta value to minimize the cost function. Here is the explanation of cost function and gradient descent.

Cost Function and Gradient Descent

The cost function determines how far the prediction is from the original dependent variable. Here is the formula for that

$$J(\theta_0, \theta_1) = \frac{1}{2m}\Sigma(h_i - y_i)^2$$

The idea of any machine learning algorithm is to minimize the cost function so that the hypothesis is close to the original dependent variable. We need to optimize the theta

value to do that. If we take the partial derivative of the cost function based on theta0 and theta1 respectively, we will get the gradient descent. To update the theta values we need to deduct the gradient descent from the corresponding theta values:

$$\theta_0 = \theta_0 - \alpha \frac{d}{d\theta_1} J(\theta_o)$$

$$\theta_1 = \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

After the partial derivative, the formulas above will turn out to be:

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \Sigma (h_i - y_i)$$

$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \Sigma (h_i - y_i) X$$

Here, m is the number of training data and alpha is the learning rate. I am talking about one variable linear regression. That's why I have only two theta values. If there are many variables, there will be theta values for each variable.

Working Example

The dataset I am going to use is from Andrew Ng's machine learning course in Coursera. Here is the process of implementing a linear regression step by step in Python.

1. Import the packages and the dataset.

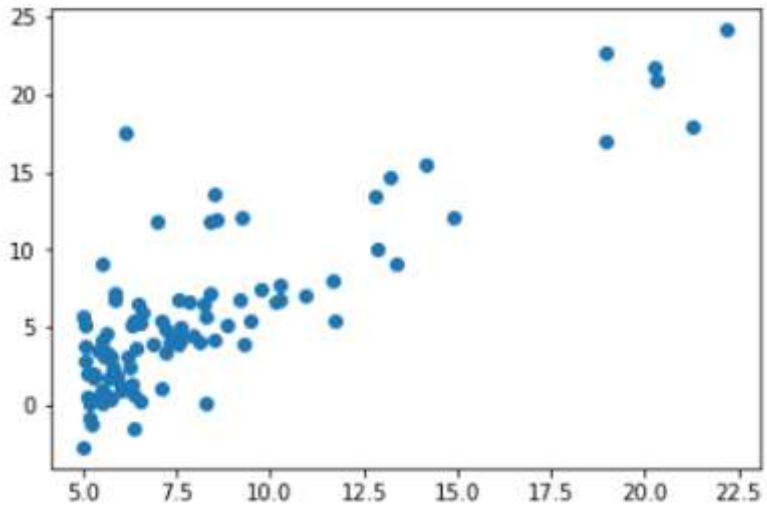
```
import numpy as np
import pandas as pd
df = pd.read_csv('ex1data1.txt', header = None)
df.head()
```

	0	1

0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

In this dataset, column zero is the input feature and column 1 is the output variable or dependent variable. We will use column 0 to predict column 1 using the straight-line formula above.

2. Plot column 1 against column 0.



The relation between the input variable and the output variable is linear. Linear regression works best when the relationship is linear.

3. Initialize the theta values. I am initializing the theta values as zeros. But any other values should also work as well.

```
theta = [0,0]
```

4. Define the hypothesis and the cost function as per the formulas discussed before.

```
def hypothesis(theta, X):
    return theta[0] + theta[1]*X
```

```
def cost_calc(theta, X, y):
    return (1/2*m) * np.sum((hypothesis(theta, X) - y)**2)
```

5. Calculate the number of training data as the length of the DataFrame. And then define the function for gradient descent. In this function, we will update the theta values until the cost function is it's minimum. It may take any number of iteration. In each iteration, it will update the theta values and with each updated theta values we will calculate the cost to keep track of the cost.

```
m = len(df)
def gradient_descent(theta, X, y, epoch, alpha):
    cost = []
    i = 0
    while i < epoch:
        hx = hypothesis(theta, X)
        theta[0] -= alpha*(sum(hx-y)/m)
        theta[1] -= (alpha * np.sum((hx - y) * X))/m
        cost.append(cost_calc(theta, X, y))
        i += 1
    return theta, cost
```

6. Finally, define the predict function. It will get the updated theta from gradient descent function and predict the hypothesis or the predicted output variable.

```
def predict(theta, X, y, epoch, alpha):
    theta, cost = gradient_descent(theta, X, y, epoch, alpha)
    return hypothesis(theta, X), cost, theta
```

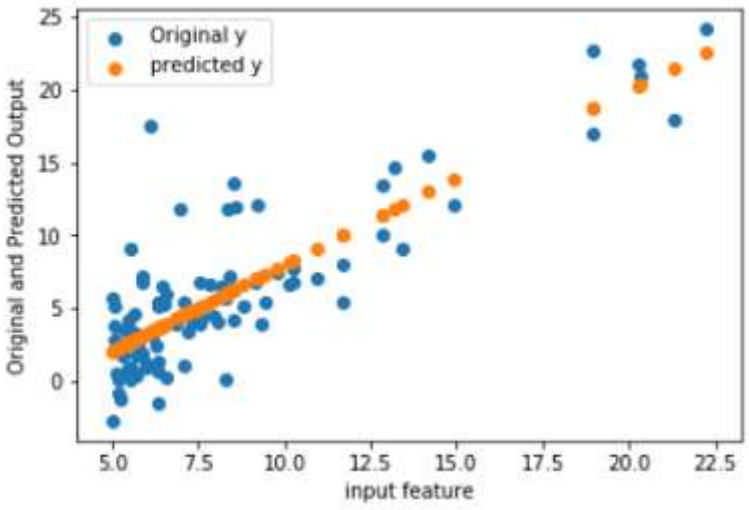
7. Using the predict function, find the hypothesis, cost, and updated theta values. I choose the learning rate as 0.01 and I will run this algorithm for 2000 epochs or iterations.

```
y_predict, cost, theta = predict(theta, df[0], df[1], 2000, 0.01)
```

The final theta values are -3.79 and 1.18.

8. Plot the original y and the hypothesis or the predicted y in the same graph.

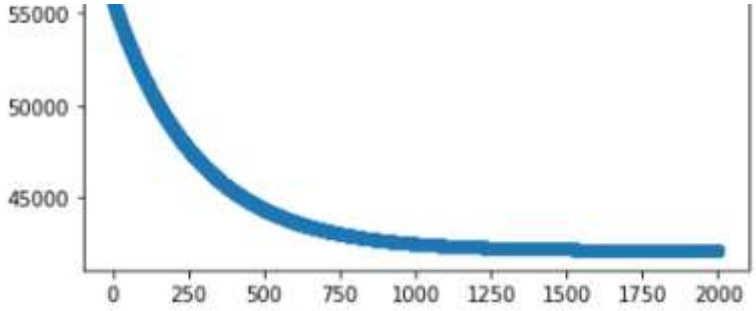
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(df[0], df[1], label = 'Original y')
plt.scatter(df[0], y_predict, label = 'predicted y')
plt.legend(loc = "upper left")
plt.xlabel("input feature")
plt.ylabel("Original and Predicted Output")
plt.show()
```



The hypothesis plot is a straight line as expected from the formula and the line is passing through in an optimum position.

9. Remember, we kept track of the cost function in each iteration. Let's plot the cost function.

```
plt.figure()
plt.scatter(range(0, len(cost)), cost)
plt.show()
```



As I mentioned before, our purpose was to optimize the theta values to minimize the cost. As you can see from this graph, the cost went down drastically in the beginning and then it became stable. That means the theta values are optimized correctly as we expected.

I hope this was helpful. Here is the link to the dataset used in this article:

rashida048/Machine-Learning-With-Python

Contribute to rashida048/Machine-Learning-With-Python development by creating an account on GitHub.

github.com

Here is the solution to some other machine learning algorithms:

Multivariate Linear Regression in Python Step by Step

Logistic Regression with Python Using Optimization Function