Published in Towards Data Science

Rashida Nasrin Sucky Follow

Jun 25, 2020 · 6 min read ★ · Listen

Photo by Michael Dziedzic on Unsplash

Multivariate Linear Regression in Python Step by

Linear regression is probably the most simple machine learning algorithm. It is very good for starters because it uses simple formulas. So, it is good for learning machine-learning concepts. In this article, I will try to explain the multivariate linear regression step by step.

Concepts and Formulas

Linear regression uses the simple formula that we all learned in school:

$Y = C + AX$

Just as a reminder, Y is the output or dependent variable, X is the input or the independent variable, A is the slope, and C is the intercept.

For the linear regression, we follow these notations for the same formula:

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

If we have multiple independent variables, the formula for linear regression will look like:

$$h = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \cdots$$

Here, 'h' is called the hypothesis. This is the predicted output variable. **Theta0 is the bias term** and all the other theta values are coefficients. They are initiated randomly in the beginning, then optimized with the algorithm so that this formula can predict the dependent variable closely.

Cost Function and Gradient Descent

When theta values are initiated in the beginning, the formula is not trained to predict the dependent variable. The hypothesis is far away from the original output variable 'Y'. This is the formula to estimate the cumulative distance of all the training data:

$$J(\theta_0, \theta_1, \theta_2, ..) = \frac{1}{2m} \sum (h_i - y_i)^2$$

This is called **the cost function**. If you notice, it deducts y(the original output) from the

original output and the predicted output is closer. To achieve just that, we need to optimize the theta values.

Here is how we update the theta values. We take the partial differential of the cost function with respect to each theta value and deduct that value from the existing theta value,

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

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

Here, alpha is the learning rate and it is a constant. I am not showing the same formula for all the theta values. But It is the same formula for all the theta values. After the differentiation, the formula comes out to be:

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

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

This is called gradient descent.

Implementation of the Algorithm Step by Step

The dataset I am going to use is from Andre Ng's machine learning course in Coursera. I will provide the link at the bottom of this page. Please feel free to download the dataset and practice with this tutorial. I encourage you to practice with the dataset as you read if this is new to you. That is the only way to understand it well.

In this dataset, there are only two variables. But I developed the algorithm for any number of variables. If you use the same algorithm for 10 variables or 20 variables, it should work as well. I will use Numpy and Pandas library in Python. All these rich libraries of Python made the machine learning algorithm a lot easier. Import the packages and the dataset:


	0	1	2
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

1. Add a column of ones for the bias term. I chose 1 because if you multiply one with any value, that value does not change.

```
df = pd.concat([pd.Series(1, index=df.index, name='00'), df], axis=1)
df.head()
```

	00	0	1	2
0	1	2104	3	399900
1	1	1600	3	329900
2	1	2400	3	369000
3	1	1416	2	232000
4	1	3000	4	539900

2. Define the input variables or the independent variables X and the output variable or dependent variable y. In this dataset, columns 0 and 1 are the input variables and column 2 is the output variable.

```
X = df.drop(columns=2)
y = df.iloc[:, 3]
```

3. Normalize the input variables by dividing each column by the maximum values of that column. That way, each column's values will be between 0 to 1. This step is not essential. But it makes the algorithm to reach it's optimum faster. Also, if you notice the dataset, elements of

```
for i in range(1, len(X.columns)):
    X[i-1] = X[i-1]/np.max(X[i-1])
X.head()
```

	00	0	1
0	1	0.469853	0.6
1	1	0.357302	0.6
2	1	0.535954	0.6
3	1	0.316213	0.4
4	1	0.669942	0.8

4. Initiate the theta values. I am initiating them as zeros. But any other number should be alright.

```
theta = np.array([0]*len(X.columns))

#Output: array([0, 0, 0])
```

5. Calculate the number of training data that is denoted as m in the formula above:

```
m = len(df)
```

6. Define the hypothesis function

```
def hypothesis(theta, X):
    return theta*X
```

7. Define the cost function using the formula of the cost function explained above

```
def computeCost(X, y, theta):
    y1 = hypothesis(theta, X)
    y1=np.sum(y1, axis=1)
    return sum(np.sqrt((y1-y)**2))/(2*47)
```

theta values until the cost function reaches its minimum.

```
def gradientDescent(X, y, theta, alpha, i):
    J = []  #cost function in each iterations
    k = 0
    while k < i:
        y1 = hypothesis(theta, X)
        y1 = np.sum(y1, axis=1)
        for c in range(0, len(X.columns)):
            theta[c] = theta[c] - alpha*(sum((y1-y)*X.iloc[:,c])/len(X))
        j = computeCost(X, y, theta)
        J.append(j)
        k += 1
    return J, j, theta
```

9. Use the gradient descent function to get the final cost, the list of cost in each iteration, and the optimized parameters theta. I chose alpha as 0.05. But you can try with some other values like 0.1, 0.01, 0.03, 0.3 to see what happens. I ran it for 10000 iterations. Please try it with more or fewer iterations to see the difference.

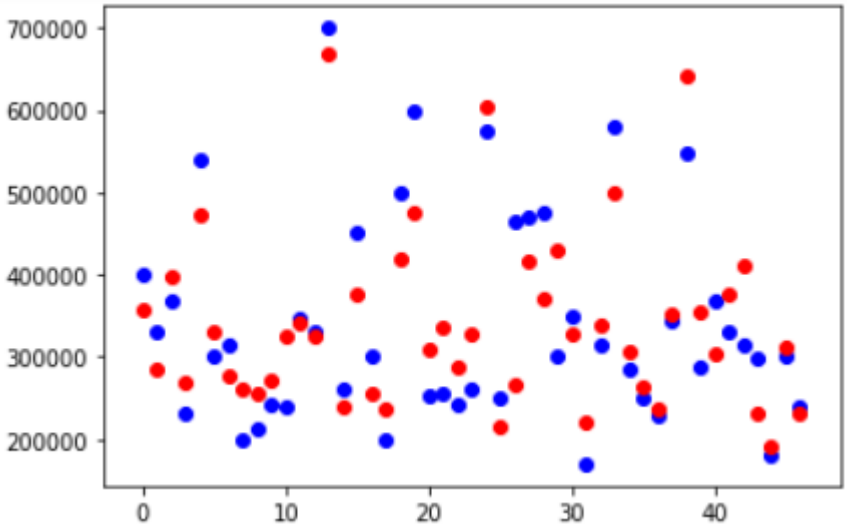
```
J, j, theta = gradientDescent(X, y, theta, 0.05, 10000)
```

10. Predict the output using the optimized theta

```
y_hat = hypothesis(theta, X)
y_hat = np.sum(y_hat, axis=1)
```

11. Plot the original y and the predicted output ‘y_hat’

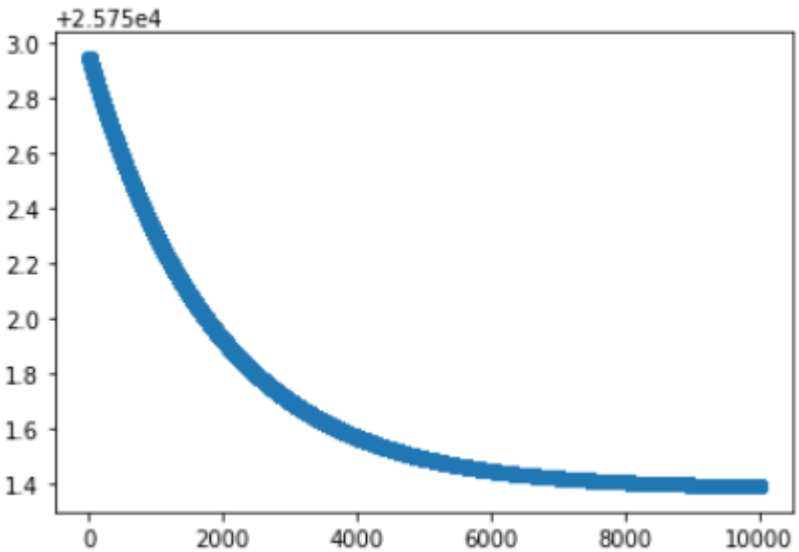
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(x=list(range(0, 47)),y= y, color='blue')
plt.scatter(x=list(range(0, 47)), y=y_hat, color='black')
plt.show()
```

Some of the output points are almost overlapping with the predicted outputs. Some are close but not overlapping.

12. Plot the cost of each iteration to see the behavior

```
plt.figure()
plt.scatter(x=list(range(0, 10000)), y=J)
plt.show()
```



The cost kept going down with each iteration. This is the indication that the algorithm worked well.

rashida048/Machine-Learning-With-Python

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

github.com

I hope it was helpful. Please feel free to follow me on Twitter and like my Facebook page.