MULTIPLE LINEAR REGRESSION IN PYTHON

MACHINE LEARNING

CODING <LANE>

BY JAY PATEL

Implementing Linear Regression with multiple features in Python

Let's start by first importing our dependencies. I will import numpy, pandas and matplotlib. numpy is used when we are dealing with the matrices. pandas is used when we are dealing with the data, and in machine learning we are always dealing with the data. and matplotlib is used while we are dealing with the graphs.

These three are one of the most important libraries when we are implementing machine learning in python so make sure you have knowledge about that.

Below are the links for Numpy and Pandas Tutorial :

Numpy : click here

Pandas : click here

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Now lets load our dataset. I have divided the data set into two sets, one for training and one for testing.

You can download/view dataset from here : train_dataset, test_dataset
Once you open the above files, copy paste the entire content in your notepad and save it.

```
train = pd.read_csv("train_data.csv")
test = pd.read_csv("test_data.csv")
```

Let's see how our data set actually looks like.

```
train.head()
```

MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold
60	3	8450	1	3	3	0	4	...	0	0	0	0	0	2
20	3	9600	1	3	3	0	2	...	0	0	0	0	0	5
60	3	11250	1	0	3	0	4	...	0	0	0	0	0	9
70	3	9550	1	0	3	0	0	...	272	0	0	0	0	2
60	3	14260	1	0	3	0	2	...	0	0	0	0	0	12

Our data set has two unwanted features, (Unnamed and Id) so, we will remove those. At the end you can see we have a sales price.

sales price is our prediction price, our Y. And all other features are our X. I have attached a file below which describes our features in detail. Go have a look on it. house-features

Also the original dataset had a lot of categorical data, which had character or string in their datafield. Now we cannot train our machine learning model with categorical (or string) data fields. So I have already pre-processed the data, converting categorical features into numerical features.

Apart from the above pre-processing, I have also done a lot of other preprocessing which was essential to make our dataset trainable.

I will cover the pre-processing of the dataset in my future videos, so stay tuned for that ! And I will also cover how I have pre-processed this dataset, which was originally downloaded from [kaggle:House Price]

Now we will drop off those two unwanted features.

```
train = train.drop(["Unnamed: 0", "Id"], axis = 1)
test = test.drop(["Unnamed: 0", "Id"], axis = 1)
```

And now let's separate the X and Y.

```
train_data = train.values
Y = train_data[:, -1].reshape(train_data.shape[0], 1)
X = train_data[:, :-1]
```

```
test_data = test.values
Y_test = test_data[:, -1].reshape(test_data.shape[0], 1)
X_test = test_data[:, :-1]
```

Now let's see what is the shape of our dataset.

```
print("Shape of X_train :", X.shape)
print("Shape of Y_train :", Y.shape)
print("Shape of X_test :", X_test.shape)
print("Shape of Y_test :", Y_test.shape)
```

```
Shape of X_train : (1200, 70)
Shape of Y_train : (1200, 1)
```

```
Shape of X_test : (258, 70)
Shape of Y_test : (258, 1)
```

So, we have data of 1200 houses in our dataset, and each house has 70 features. Similarly, we have 258 houses in our dataset.

Now let's have a quick overview of the linear regression.

If you do not know anything about linear regression then go to this video Linear Regression, where I have explained what is linear regression.

We know that in linear regression we make our predictions by this equation :

$$y\_pred = \theta_n x_n + \theta_{n-1} x_{n-1} + \theta_{n-2} x_{n-2} + \ldots + \theta_2 x_2 + \theta_1 x_1 + \theta_0$$

And in python, we can write above equation as the matrix multiplication of theta and X

```
y_pred = matrix_mul(X,theta)
```

Overview of Linear Regression

Straight Line

y_pred = $\theta_n x_n + \theta_{n-1} x_{n-1} + \theta_{n-2} x_{n-2} + \ldots + \theta_2 x_2 + \theta_1 x_1 + \theta_0$

y_pred = $matrix\_mul(X, theta)$

Cost Function

cost = $\frac{1}{2m} \Sigma[(y\_pred - Y)^2]$

Gradient Descent

d_theta = $\frac{1}{m}[matrix\_mul(X^T, y\_pred - Y)]$

theta = theta - $\alpha * d\_theta$

Now, to do a proper matrix multiplication of X and theta, we will need to add a column of 1s before all the features of X. The reason for doing so is that we are multiplying θ 2 with X 2 , θ 1 with X 1 and there is no X0 to multiple with θ 0 . So we will add 1 at the place of X0.

```
X = np.vstack((np.ones((X.shape[0], )), X.T)).T
X_test = np.vstack((np.ones((X_test.shape[0], )), X_test.T)).T
```

Now, we know that the cost function is the error representation and we find the error by subtracting our prediction value with the actual value. So our cost function will be calculated by the above formula.

To know in detail about the Linear Regression Cost function, go to this video : Cost Function

Now to minimize the error (or Cost) we need to use something called gradient descent and the gradient descent basically works by reducing the cost value in such a way that it reaches its local minima.

If you want to know about the gradient descent then click Gradient Descent which will take you to the video of the gradient descent where i have explained the gradient descent as simple as possible and in as much detail as possible.

The equation for the gradient descent is given above.

So let's make our linear regression model in python.

1. I'm taking four parameters X, Y, learning rate (which is alpha) and iterations. Iterations specifies how many times we want to run the loop.
2. Define m as the size of the data set(which is currently 1200).
3. And theta will be the vector of zeros. so it will be a matrix of size (n, 1) where n is the number of features. so basically of the size (70, 1)
4. We are running the loop for iteration time. And in every iteration we will compute our above 4 equations.
5. We will also keep track of our cost at every iteration, by maintaining a cost_list.
6. And finally, return the theta parameter (which will be trained) and cost_list.

```
def model(X, Y, learning_rate, iteration):
    m = Y.size
    theta = np.zeros((X.shape[1], 1))
    cost_list = []

    for i in range(iteration):

        y_pred = np.dot(X, theta)

        cost = (1/(2*m))*np.sum(np.square(y_pred - Y))

        d_theta = (1/m)*np.dot(X.T, y_pred - Y)
        theta = theta - learning_rate*d_theta

        cost_list.append(cost)

        # to print the cost for 10 times
        if(i%(iteration/10) == 0):
            print("Cost is :", cost)

    return theta, cost_list
```

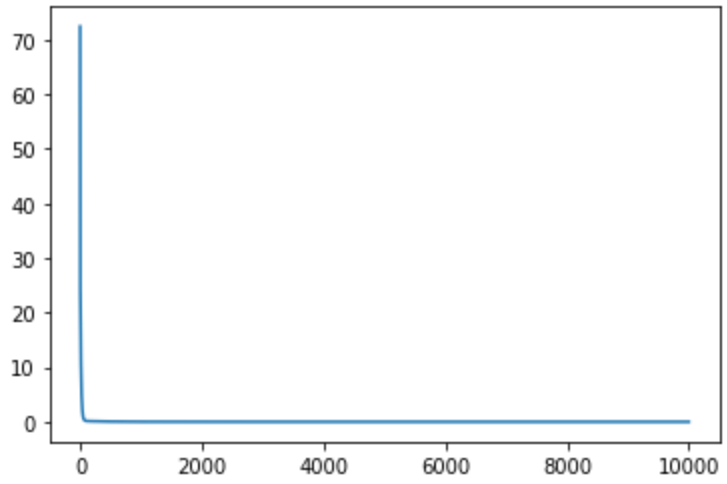
Now let's call our model.

```
iteration = 10000
learning_rate = 0.000000005
theta, cost_list = model(X, Y, learning_rate = learning_rate, iteration =
iteration)
```

```
Cost is : 72.37539364066856
Cost is : 0.027904168310316866
Cost is : 0.017251065372144152
Cost is : 0.016355272705548277
Cost is : 0.016158836087530753
Cost is : 0.016040958498450615
Cost is : 0.015946827323753437
Cost is : 0.01586789631723002
Cost is : 0.015800568014785396
Cost is : 0.015742355306482898
```

We can see our cost decreasing with every iteration. We can also plot a graph of cost vs iteration

```
rng = np.arange(0, iteration)
plt.plot(rng, cost_list)
plt.show()
```



Now the fun part!

We will test the accuracy of our model on a test dataset. And for testing accuracy, I'm going to calculate the error, and subtract the 1 by error, to get the accuracy.

Below is the equation for the error :

error = (1/m) * ∑|y_pred−Y|

In python, it will be :

```
y_pred = np.dot(X_test, theta)
error = (1/X_test.shape[0])*np.sum(np.abs(y_pred - Y_test))
```

```
print("Test error is :", error*100, "%")
print("Test Accuracy is :", (1- error)*100, "%")
```

```
Test error is : 12.959999999999999 %
Test Accuracy is : 87.03999999999999 %
```

Our model has 87 percent accuracy and we achieved that with just a few lines of code !!

Congratulations !!

Subscribe to *CODING LANE* for more amazing content :

CODING
<LANE>