University of Asia Pacific

Book Report

Multiple Linear Regression From Scratch

Course Title: Artificial Intelligence and Expert Systems Lab

Submitted by:

Sourav Paul

17201030

Submitted To:

Dr. Nasima Begum

Assistant Professor , Department of CSE

University Of Asia Pacific

Problem Statement:

The advertising experiment between Social Media Budget and Sales using Multivariable Linear Regression with open source dataset.

Libraries Used In This Project:

1. Numpy
2. Pandas
3. Matplotlib

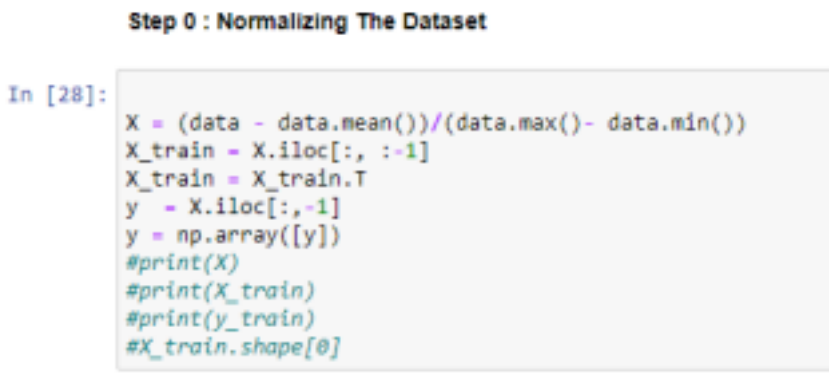
Dataset:

www.kaggle.com/fayejavad/marketing-linear-multiple-regression

Documentation:

Step 0 : Normalizing the Dataset

More often than the data we use in ML models need to be normalized in order to make all the data be in a common range. In this case all the values are between -1 to 1 after normalization.



Step 1: Parameter Initialization

Step 1 : Initializing Parameters

```
In [29]: def initialize_parameters(lenw):
             theta = np.random.randn(1,lenw)
             theta_not = 0
             return theta,theta_not
```

This function initializes the ‘theta’ vector with random data and ‘theta_not’ variable with 0.

Step 2: Hypothesis Function

Step 2 : Hypothesis Function

```
In [30]: def hypothesis_function(X,theta,theta_not):
             h = np.dot(theta,X) + theta_not
             return h
```

This function creates the hypothesis vector ‘h’ which is the resultant of the dot product of ‘X’ and ‘theta’ plus ‘theta_not’.

Step 3: Calculating the Cost Function

Step 3 : Calculating Cost function

```
In [31]: def cost_function(h,y):
             m = y.shape[1]
             J = (1/2*m)*np.sum(np.square(h-y))
             return J
```


This function calculates the cost function from ‘h’ vector and ‘y’ vector by calculating the sum of squared difference between ‘h’ and ‘y’.

Step 4: Calculating the Cost Function

```
def back_prop(X,y,h):
    m = y.shape[1]
    dh = (1/m)*(h-y)
    d_theta= np.dot(dh,X.T)
    d_theta_not = np.sum(dh)
    return d_theta,d_theta_not
```

The ‘back_prop’ function works as a helper function of the ‘gradient_descent’ function. This function calculates the value of ‘d_theta’ and ‘d_theta_not’ , values of which are then used in ‘gradient_descent’ function to calculate the new value of ‘theta’ and ‘theta_not’.

```
def gradient_descent(theta,theta_not,d_theta,d_theta_not,learning_rate):
    theta = theta-learning_rate*d_theta
    theta_not = theta_not- learning_rate*d_theta_not
    return theta,theta_not
```


Step 5: Iteration Till Convergence

```
def MLR(x_train,y,learning_rate,epochs):
    lenw = x_train.shape[0]
    theta,theta_not = initialize_parameters(lenw)  #step 1
    costs_train = []

    for i in range(1,epochs+1):
        h = hypothesis_function(x_train,theta,theta_not) #step 2
        cost_train = cost_function(h,y) #step 3
        d_theta,d_theta_not = back_prop(X_train,y,h) #step 4
        theta,theta_not = gradient_descent(theta,theta_not,d_theta,d_theta_not,learning_rate)   #step 5
        if i%10==0:
            costs_train.append(cost_train)

        print('Epochs' +str(i)+'/'+str(epochs)+' : ')
        print('Training Cost '+ str(cost_train))

    plt.plot(costs_train)
    plt.xlabel('Iteration (per tens)')
    plt.ylabel('Training cost')
    plt.title('Learning cost ' +str(learning_rate))
    plt.show()
```

The MLR function uses all the functions before to create a Multivariate Linear Regression Model. It iterates steps 2-5 ‘epochs’ number of times. Then it plots the values of cost in a graph.

Step 6 : Calling The Function

```
MLR(X_train,y,0.2,1000)
```

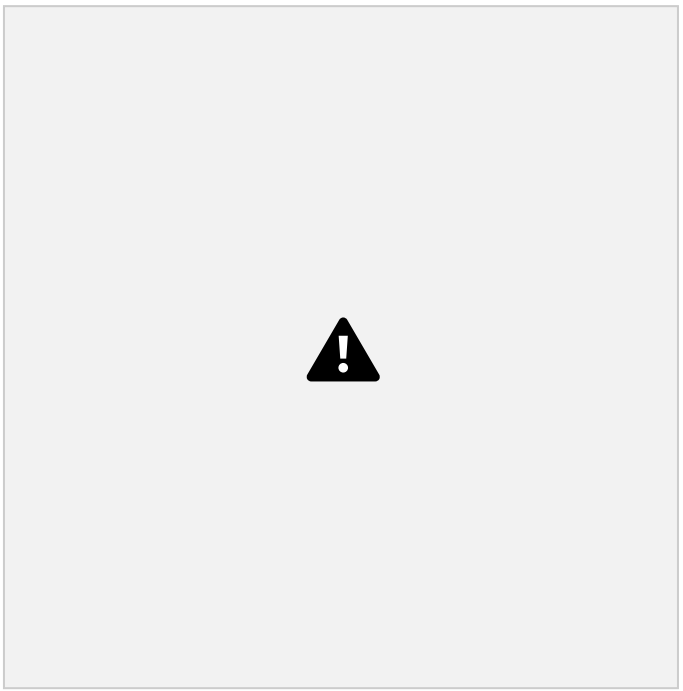
Calling the MLR function with the X_train dataset and 0.2 learning rate for 1000 iterations.

Result :

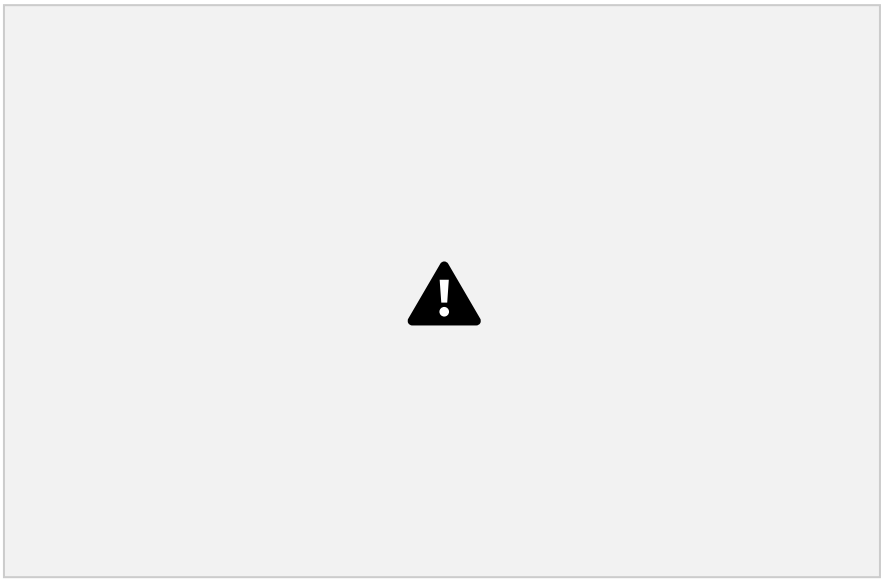
Cost Initially :

```
Epochs1/1000 :
Training Cost 7435.258833384056
Epochs2/1000 :
Training Cost 7186.925317898538
Epochs3/1000 :
Training Cost 6947.2656945058725
Epochs4/1000 :
Training Cost 6715.969523258287
Epochs5/1000 :
Training Cost 6492.737681949253
Epochs6/1000 :
Training Cost 6277.281947178197
Epochs7/1000 :
Training Cost 6069.324591125366
Epochs8/1000 :
Training Cost 5868.59799344109
Epochs9/1000 :
Training Cost 5674.844267676484
Epochs10/1000 :
```

Cost After Converging :



Cost Graph:



Conclusion

This is not the end. We can make the program much more optimized .