Machine Learning Assignment

Uni-variate and Multivariate Linear Regression with Gradient Descend

Muhammad Shahzeb Kayani
Department of Electrical and Computer Engineering
Air University
Islamabad, Pakistan
shahzebkayani0@gmail.com

***Abstract*—This document briefly discusses the process of implementing Uni-variate and Multivariate linear regression algorithm using the logic of gradient descend in python.**

I. Problem 1: Linear regression (Univariate)

A. Problem Statement

We are approached by a client from a food truck business owner to develop a model to predict profits. The CEO of the food truck dispatch fleet is planning to expand the business in different cities. The company already has trucks in various cities and also have the data for population and profits from those cities. You are required to analyze the data and propose a 'Linear Regression' framework to predict profits, and facilitate in selecting the city for expansion.

B. Loading the data in Framework

So before starting any task, the first and an important goal was to load the data from our file into our arrays. To complete this task, I imported a library named Panda used its DataFrame module. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

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

# Reading the data's file.
data = pd.read_csv('data1.csv', names = ['population', 'profit'])

print(data)
X_df = pd.DataFrame(data.population)
y_df = pd.DataFrame(data.profit)

# Length, or number of observations, in our data
m = len(y_df)
```

Fig. 1. Loading data into our framework

C. Visualizing the data

After reading the data from the file and populating our arrays, the next logical step was to visualize the data so that we can understand it better.To achieve that, I used the module pyplot of Matplotlib library.

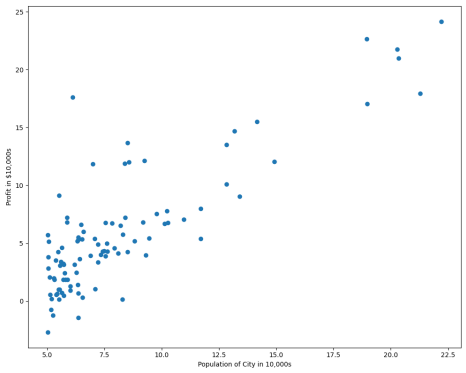

Fig. 2. Visualising our data from the file

We used its scatter module which plots the data. X-axis is population and y-axis is profit. (Fig 2)

D. Applying Linear Regression

1) Visualizing the Hypothesis: Before applying the algorithm of linear regression, it was a better move to know the real location of our initial hypothesis. After visualizing the hypothesis it was obvious that at some point, how this can fit into our data. Hypothesis is shown in fig 3.

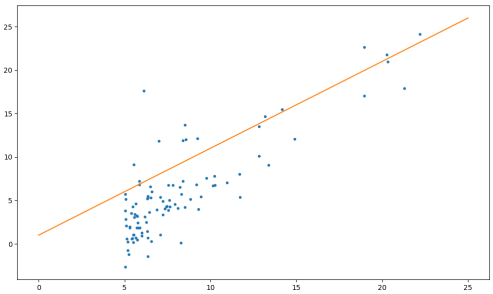

Fig. 3. Visualizing the Hypothesis

2) Cost Function: The Cost function gives us the the idea about how much the difference between our predicted value and the actual value is. I have created a separate function

of ost which is called repeatedly from my gradient descend function. It takes the values of theta and our data-set as input and computes the value of cost.

```
def cost_function(X, y, theta):
    """
    cost_function(X, y, theta) computes the cost of using theta as the
    parameter for linear regression to fit the data points in X and y
    """

    ## Calculate the cost with the given parameters
    J = np.sum((X.dot(theta) - y) ** 2) / 2 / m

    return J

def gradient_descent(X, y, theta, alpha, iterations):
```

Fig. 4. Cost Function

3) Gradient Descend function: This function in my program implements the logics of gradient descend to find the best possible values of theta. It calls the cost function, updates the theta and calls the cost function again until our iterations are completed. It stores the value the cost returns on every iteration which is used to visualize the cost function with respect to the number of iterations.

```
def gradient_descent(X, y, theta, alpha, iterations):
    """
    gradient_descent Performs gradient descent to learn theta
    theta = GRADIENTDESENT(X, y, theta, alpha, num_iters) updates theta by
    taking iterations gradient steps with learning rate alpha
    """
    cost_history = [0] * iterations

    for iteration in range(iterations):
        hypothesis = X.dot(theta)
        loss = hypothesis - y
        gradient = X.T.dot(loss) / m
        theta = theta - alpha * gradient
        cost = cost_function(X, y, theta)
        cost_history[iteration] = cost
        print(cost)
    return theta, cost_history
```

Fig. 5. Gradient Descend Function

4) Graph of Cost Function: The values the cost function returns on every iterations are used here to plot the value of cost on every iteration.

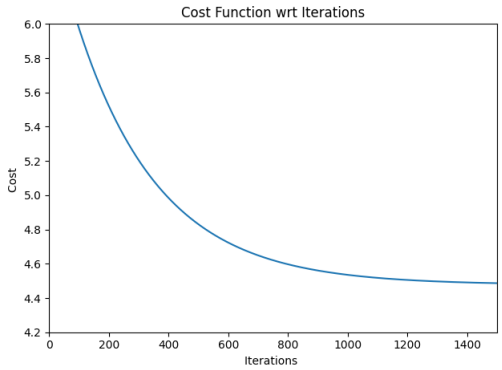

Fig. 6. Graph of Cost Function

5) Plotting the final hypothesis: The final values of theta we get after 1500 iterations are used to plot the line of our hypothesis with out data.

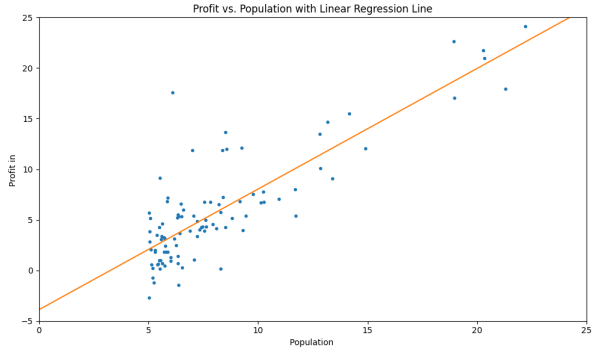

Fig. 7. Plotting final Hypothesis

E. Changing the Learning Rate

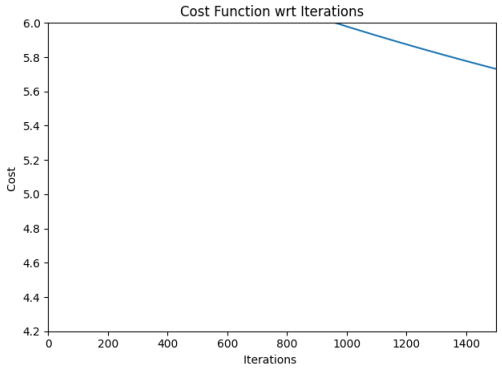

Fig. 8. Decreasing the learning rate

From the figure 8 it can be seen that we decrease the value of alpha i.e. the larning rate, the algorithm takes more iterations to converge to the solution. Figure 9 shows the cost function's values if we increase the value of alpha by a small factor. It converges in less iterations.

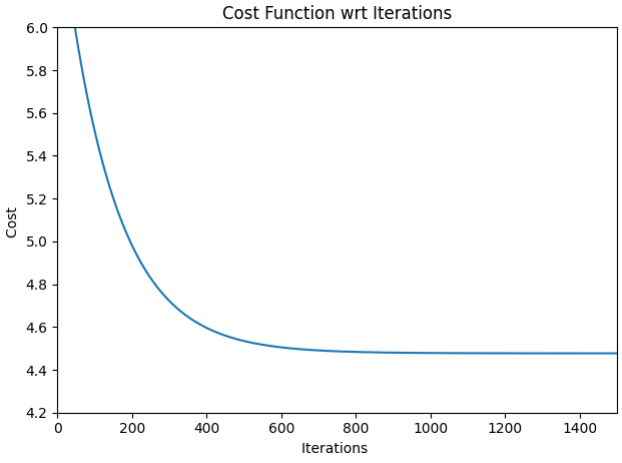

Fig. 9. Increasing the learning rate

II. Problem 2: Linear regression (Multivariate)

In another problem, we are approached by a CEO of a real estate company to design a housing price prediction engine. Suppose, the client wants to sell a house and he wants to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices. You are required to analyze the data and propose a 'Linear Regression' framework to predict housing prices.

A. Loading the data in Framework

So before starting any task, the first and an important goal was to load the data from our file into our arrays.As this is a multivariate linear regression problem, we have to read multiple features now. To complete this task, I imported a library named Panda used its DataFrame module. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. after that, i used the shape module of pandas. The shape attribute of pandas. DataFrame stores the number of rows and columns as a tuple (number of rows, number of columns) . It is also possible to unpack and store them in separate variables.
After that i used the normalize module of sklearn library which normalized my data so that the number and size of calculations can be decreased.

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize

data = pd.read_csv('ex2data2.csv')
data.shape
data = normalize(data, axis=0)

X = data[:, 0:2]
Y = data[:, 2:]
```

Fig. 10. Loading data into our framework

B. Visualizing the data

After reading the data from the file and populating our arrays, the next logical step was to visualize the data so that we can understand it better.To achieve that, I used the module pyplot of Matplotlib library.

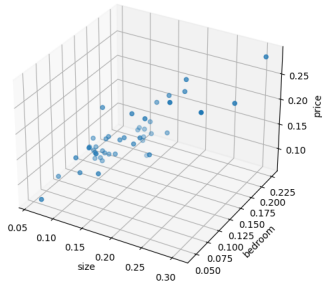

Fig. 11. Loading data into our framework

We used its scatter module which plots the data. X-axis is population and y-axis is profit. (Fig 11).

C. Applying Linear Regression

1) Visualizing the Hypothesis: Before applying the algorithm of linear regression, it was a better move to know the real location of our initial hypothesis. After visualizing the hypothesis it was obvious that at some point, how this can fit into our data.

2) Gradient Function: The gradient function gives us teh updated value of theta when old theta and cost is given to it through its arguments.

```
def gradient_(theta, X, Y)_:
    tempX = np.ones((X.shape[0], X.shape[1] + 1))
    tempX[:,1:] = X
    d_theta = - np.average((Y - h(theta, X)) * tempX, axis=_0)
    d_theta = d_theta.reshape((d_theta.shape[0], 1))
    return d_theta
```

Fig. 12. Cost Function

3) Gradient Descend function: This function in my program implements the logics of gradient descend to find the best possible values of theta. It finds the cost value, updates the theta and finds the cost function again prints the cost value for every 100 iterations until our iterations are completed. It stores the value the cost returns on every iteration which is used to visualize the cost function with respect to the number of iterations.

```
def gradient_(theta, X, Y)_:
    tempX = np.ones((X.shape[0], X.shape[1] + 1))
    tempX[:,1:] = X
    d_theta = - np.average((Y - h(theta, X)) * tempX, axis=_0)
    d_theta = d_theta.reshape((d_theta.shape[0], 1))
    return d_theta


def gradient_descent_(theta, X, Y, learning_rate, max_iteration, gap)_:
    cost = np.zeros(max_iteration)
    for i in range(max_iteration)_:
        d_theta = gradient_(theta, X, Y)
        theta = theta - learning_rate * d_theta
        cost[i] = loss(theta, X, Y)
        if i % gap == 0_:
            print_('iteration : ', i, ' loss : ', loss(theta, X, Y))
    return theta, cost
```

Fig. 13. Gradient Descend Function

4) Graph of Cost Function: The values the cost function returns on every iterations are used here to plot the value of cost on every iteration.

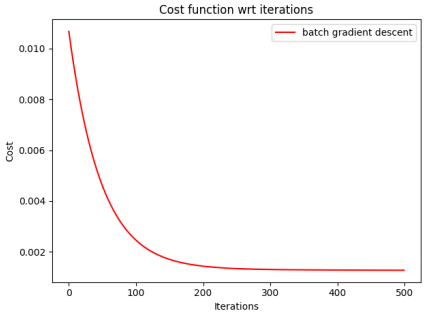

Fig. 14. Graph of Cost Function

5) Plotting the final hypothesis: The final values of theta we get after total iterations are used to plot the line of our hypothesis with out data.

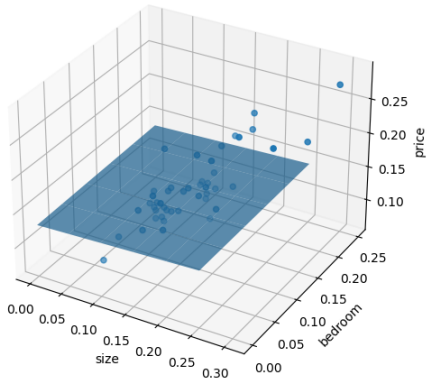

Fig. 15. Plotting final Hypothesis

D. Changing the Learning Rate

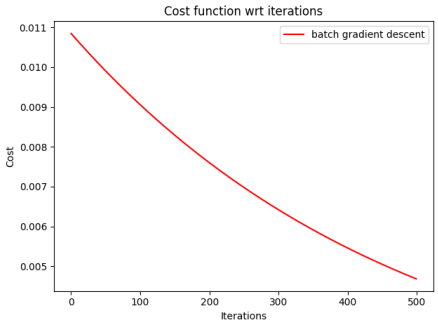

Fig. 16. Decreasing the learning rate

From the figure 16 it can be seen that we decrease the value of alpha i.e. the larning rate, the algorithm takes more iterations to converge to the solution. Figure 17 shows the cost function's values if we increase the value of alpha by a small factor. It converges in less iterations.

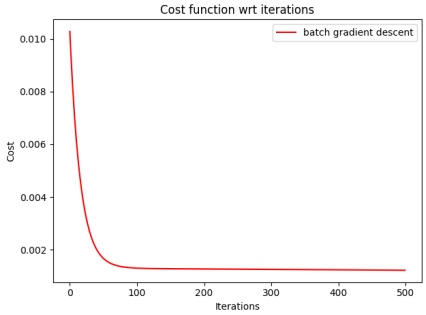

Fig. 17. Increasing the learning rate

E. Output

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
runfile('C:/Users/Shahzeb/PycharmProjects/pythonProject/lk.py', wdir='C:/Users/Shahzeb/PycharmProject
iteration :  0  loss :  0.010282152337747296
iteration :  100  loss :  0.0013034341035584953
iteration :  200  loss :  0.0012711143325765902
iteration :  300  loss :  0.0012549626243101902
iteration :  400  loss :  0.0012391537863914752
Final value of theta is
[[0.12718756]
 [0.05131894]
 [0.02834908]]
```

Fig. 18. Increasing the learning rate

Remarks :

Despite knowing and understanding the logic and math behind this algorithm , it was quite difficult because I was not used to work on Python. Through this assignment ,not only I have learned how to implement linear regression algorithm but my Python skills have also improved. It just needed some understanding of certain libraries and modules in order to make it work. Although, I'm still struggling with plotting the surface and contour graphs because I could not decide the arguments of contour function. But only if i had some more times, I would have been able to figure it out. Maybe if i had tried storing the values of thetas for each iteration in an array, this would have solved half of my problem. The other problem was with the z argument.