Assignment-1: Multivariate regression

Rashik Iram Chowdhury (2111336642), Zarin Akter (2011704042) and Md. Mutasim Farhan (2013123642)
Emails: {rashik.chowdhury@northsouth.edu}, {zarin.akter@northsouth.edu}, {mutasim.farhan@northsouth.edu}

December 19, 2024

I. Introduction

In this assignment, we have performed Multivariate Linear Regression using a Gradient Descent Algorithm (without using a machine learning library) on the Air Quality Dataset [1] and reported Mean Squared Error (MSE) for training and testing datasets. Linear Regression is a supervised machine learning algorithm with a continuous target set (y). Multivariate Linear Regression is a linear regression that works with a target variable (y) and two or more independent features (X). Gradient Descent Algorithm is an optimization algorithm that minimizes the cost function by iteratively adjusting the weights or parameters of the model in the direction of the negative gradient of the cost function until the minimum of the cost function is reached.

II. Method

In this section, we sequentially explore the methodologies for implementing multivariate linear regression using gradient descent on the air quality dataset without using any machine learning library. Fig. 1 shows the workflow of this study.

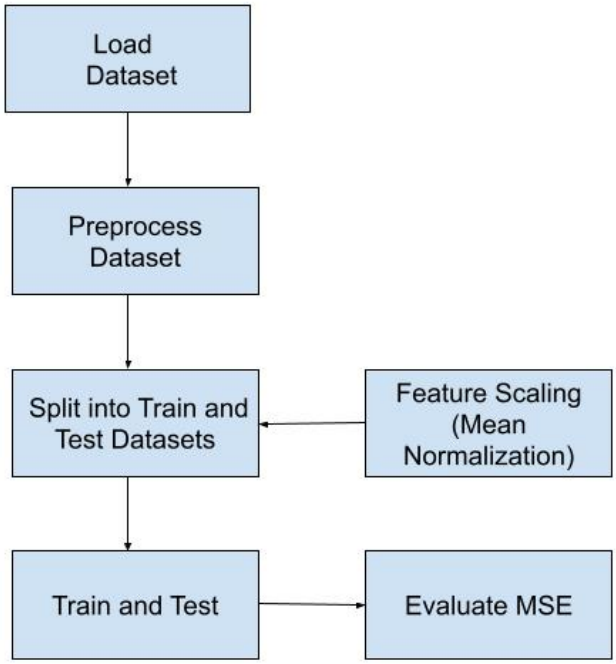

Fig. 1. Methodology Block Diagram

- **Load Dataset:** At first, we downloaded the dataset in xlsx format and then converted the dataset into CSV format and loaded this formatted dataset using pandas library. Then, the dataset was converted to a numpy array.

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("G:\\Assignment\\AirQualityUCI.csv")
4 data = np.array(df)
```

- **Preprocess Dataset:** We have replaced the missing values in the dataset labeled as -200 with the mean of the entire respective columns. Then we dropped the first two columns as they were not applicable for linear Regression.

```
import math
final_dataset = data[:,2:15]
#Replacing Missing Values by mean
total_sum = [0] * len(final_dataset[0])
instances = [0] * len(final_dataset[0])
for i in range(len(total_sum)):
    total_sum[i] = 0
    instances[i] = 0
for columns in range(len(final_dataset[0])):
    for rows in range(len(final_dataset)):
        x = final_dataset[rows][columns]
        if x != -200:
            total_sum[columns] += x
            instances[columns] += 1
means = [round(total_sum / instances,1) if instances != 0 else 0 for
total_sum, instances in zip(total_sum, instances)]
for columns in range(len(final_dataset[0])):
    for rows in range(len(final_dataset)):
        if final_dataset[rows][columns] == -200:
            final_dataset[rows][columns] = means[columns]
```

- **Split into Train and Test Datasets:** We then split the dataset into train and test datasets at a ratio of 75:25, respectively. Among the remaining 13 features, the first 10 were considered features (X), and the last column was considered a target. The remaining dataset columns will later be considered targets for evaluating the model's performance, as shown in the Experiment Results section[III].

```
import math
final_dataset = data[:,2:15]
total_instances = len(final_dataset)
#We could have tried shuffling for a better experiment
Train_size = math.floor(0.75*total_instances)#7017
Train_ds = final_dataset[:Train_size]
Test_ds = final_dataset[Train_size:]
```

A. Feature Scaling (Mean Normalization)

After replacing the dataset with the mean value, we calculated each column's standard deviation to perform mean normalization and replaced the original values with their respective mean normalized values. This feature scaling technique is applied to give equal importance to the data for each feature. The feature scaling function returned the mean and standard deviation from the training dataset. This standard deviation and mean are used to directly calculate the normalized values for the test dataset and override its existing values.

```
def feature_scaling(dataset):
    total_sum = [0] * len(dataset[0])
    instances = [0] * len(dataset[0])
    for columns in range(len(dataset[0])):
        for rows in range(len(dataset)):
            x = dataset[rows][columns]
            if x != -200:
                total_sum[columns] += x
                instances[columns] += 1
        means = [total_sum[i] / instances[i] if instances[i] != 0 else 0
        for i in range(len(total_sum))]
    for columns in range(len(dataset[0])):
        for rows in range(len(dataset)):
            if dataset[rows][columns] == -200:
```

```
                dataset[rows][columns] = means[columns]

    sum_squared_diff = [0] * len(dataset[0])
    for columns in range(len(dataset[0])):
        for rows in range(len(dataset)):
            x = dataset[rows][columns]
            diff = x - means[columns]
            sum_squared_diff[columns] += diff ** 2

   std_deviations = [((sum_squared_diff[i] / (instances[i] - 1)) ** 0.5)
   if instances[i] > 1 else 0 for i in range(len(sum_squared_diff))]

    for columns in range(len(dataset[0])):
        for rows in range(len(dataset)):
            value = dataset[rows][columns]
            normalized_value = (value - means[columns]) / std_deviations[columns]
            if std_deviations[columns] != 0 else 0
            dataset[rows][columns] = normalized_value
    return dataset, means, std_deviations
```

- **Train and Test:** We calculated the mean squared error or cost function after feature scaling and splitting test and train datasets into X_train, y_train, X_test, and y_test. As the training dataset has ten features, weight is considered as a vector and bias is a scalar number. Each instance or row is multiplied with the respective weights as a dot product and added with a bias to compute the hypothesis function. This hypothesis function engenders a predicted target value, which is deducted from the actual value. This difference is then squared over the number of instances in the dataset to calculate the mean squared difference of the hypothesis. Similarly, partial derivatives with respect to the weights and bias term were calculated to compute the gradient descent algorithm, which takes in 8 parameters. We initialized the weight and bias values based on the correlation heatmap (as shown in the Experiment Analysis Section). The gradient descent algorithm returns the optimal weights and bias vector terms values for which the cost function or mean squared error is minimal over the number of epochs or iterations.

```
def cost_function(X, y, w, b):

    m = X.shape[0]  #This is the number of instances in the dataset
    cost = 0

    for i in range(m):
        f_wb = np.dot(X[i], w) + b # This is the hypothesis function
        cost = cost + (f_wb - y[i])**2
        total_cost = 1 / (2 * m) * cost

    return total_cost

  import copy
  def gradient_calculation(X, y, w, b):

    m,n = X.shape  #Here m is number of instances and n is number of features
    dj_dw = np.zeros((n,))
    dj_db = 0.0

    for i in range(m):
        difference = (np.dot(X[i], w) + b) - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + difference * X[i, j]
        dj_db = dj_db + difference

    dj_dw = dj_dw / m
    dj_db = dj_db / m
```

```
    return dj_db, dj_dw

def gradient_descent(X, y, w_in, b_in, cost_function, gradient_calculation,
alpha, iterations):

    J_history = []
    w = copy.deepcopy(w_in)
    b = b_in

    for i in range(iterations):

        dj_db,dj_dw = gradient_calculation(X, y, w, b)

        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        if i<10000:
            J_history.append(cost_function(X, y, w, b))

        if i% math.ceil(iterations / 10) == 0:
            print(f"Iteration␣{i:4d}:␣Cost␣{J_history[-1]:8.2f}")

    return w, b, J_history
```

- **Evaluate MSE:** After utilizing the gradient descent algorithm to find optimal weights and bias values, the cost function was used, where these updated values were given as inputs. The corresponding MSE outputs for the train and test datasets were calculated and printed. To better analyze our hypothesis or model, we used the same weights and bias values as inputs to the cost function for the other two target features separately after updating the y_train and y_test to compute MSE for both targets.

III. Experiment results

We applied the gradient descent algorithm using six different learning rate sets and iterations to compute six different costs (MSE). We took the learning rate and iterations, which gave us the lowest cost for the gradient descent function.

A. Data correlation analysis

A data correlation analysis looks at the direction and strength of a relationship between two or more variables in a dataset. It evaluates the relationship between changes in one variable and changes in another.

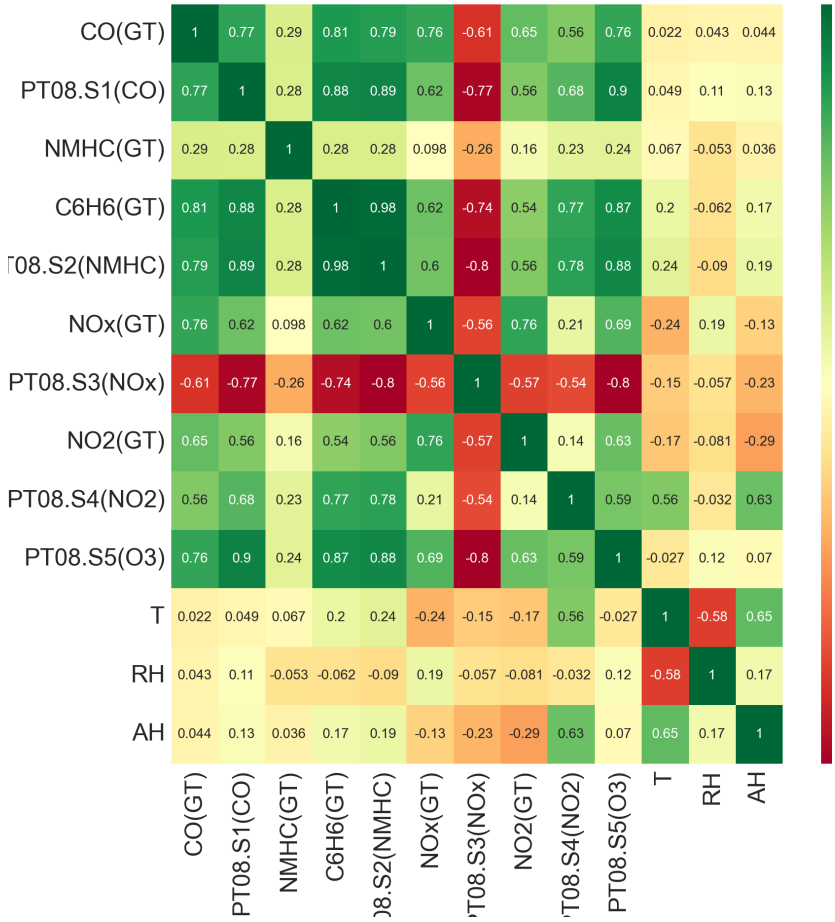

Fig. 2. Correlation between attributes

A correlation heatmap is a visual tool that shows the correlation between many variables as a matrix with different colors. Similar to a color chart, it illustrates how closely various variables are related. The heatmap in Fig. 2 displays the degree of correlation between the features in the dataset utilized in this study. It shows how the independent variables are correlated.

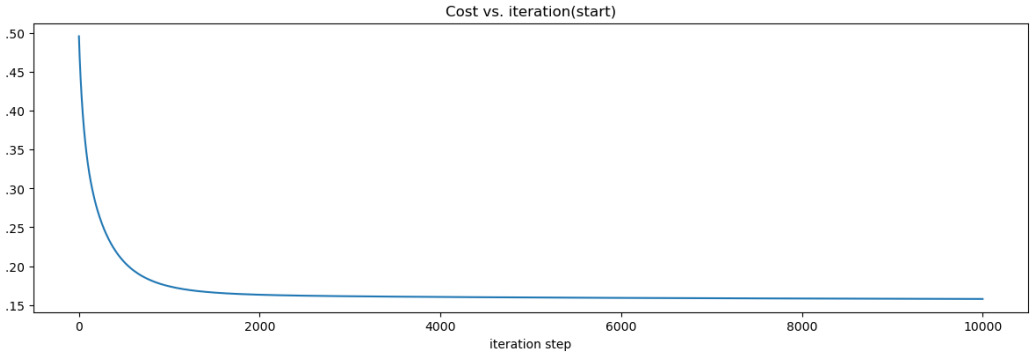

Fig. 3. Cost vs Iteration

Cost over iteration graph shows the training performance of the model by displaying the Mean Squared Error or cost after each iterations.

TABLE I
VARIATION OF MEAN SQUARED ERROR (MSE/COST) WITH THE CHANGE OF LEARNING RATE AND ITERATION

Trials	**Learning rate (α)**	**Iterations**	**MSE (Cost)**
1	0.03	1000	0.16
2	0.03	5000	0.16
3	0.001	3000	0.25
4	0.1	1000	0.16
5	0.01	1000	0.18
6	0.01	10000	0.16

TABLE II
MEAN SQUARED ERROR FOR TARGETS AH, RH, AND T

MSE					
AH		RH		T	
Train	Test	Train	Test	Train	Test
0.158	0.079	0.673	1.119	0.494	0.307

IV. DISCUSSION

In this assignment, we calculated the Mean Squared Error for the Air Quality Dataset using a gradient descent algorithm to find optimized weights and bias values. The hypothesis function obtained using the gradient descent algorithm's weight is very accurate- (obtained when learning rate=0.01, Iteration=10000), producing MSE values approximately within the range of zero for both train (0.158) and test (0.079) datasets. The MSE score for the unseen dataset for all three targets(AH, RH, T) is close to zero, so we can say that our model is very accurate.

REFERENCES

[1] Air quality data set. 2004. Accessed in 2004. https://archive.ics.uci.edu/dataset/360/air+quality.