Machine Learning Assignment 1 Report

Name: Mahmoud Khaled Ibrahim

Id: 40-9953

Introduction:

In this assignment w are required to implements a machine learning model, the computer from the data. This assignment shows the effect of using adding or decreasing features, adding more features make the model more accurate to predict the input data. the project is divided into 3 parts, the first part is training our model, the second part is to validate our mode and the last part is to predict the price of house.

Methodology:

In this assignment jupyter notebook is used and jupyter notebook use python as a coding language, the following line shows the imported or used libraires

os: is used to show the path
numpy: is used as array
pyplo: used to draw graphs
pandas: used to read csv file (our data)

Implementations:

```
In [13]: import os
         import numpy as np
         from matplotlib import pyplot
         import pandas as pd
```

```
In [14]: data=pd.read_csv("house_prices_data_training_data.csv")
         columns =data.columns[3:]
         norm_data = (data[columns]-(data[columns]).mean())/(data[columns]).std()
         data[columns] = norm_data
```

First, we read the data using pandas, then we normalize the data because it makes the model more accurate.

```
In [15]: train, validate , test = np.split(data.sample(frac=1),[int(.6*len(data)),int(.8*len(data))])
```

```
In [16]: train = train.to_numpy()
         test = test.to_numpy()
         validate = validate.to_numpy()
         train_x = train[:,3:]
         train_y = train[:,2]
         train_m =train_y.size
```

```
In [17]: train_x = np.concatenate([np.ones((train_m,1)), train_x], axis =1)
```

After reading our data we split the data into 3 sets. Training, validation and testing. The data is split by percentage 60 for training, 20 for validation and 20 for testing. the n we start working on the training sets.

We add one in the beginning of each row in input data, the one which is added is used as the coefficient of theta zero.

In [18]:
```
def gradientDescentMulti(X, y, theta, alpha, num_iters):
    m = y.shape[0]
    theta = theta.copy()
    J_history = []
    for i in range(num_iters):
        theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)
        J_history.append(computeCostMulti(X, y, theta))
    return theta, J_history
```

The gradientDescentMulti function is used to compute the theta and the cost for this theta. It is used to choose coefficient of theta that will minimize our cost function (J) and the cost function (J) is the difference between input and output. The input data is the features which is X, the output data is the price which is Y.

In [19]:
```
def computeCostMulti(X, y, theta):
    m = y.shape[0]
    J = 0
    h = np.dot(X, theta)
    J = (1/(2 * m)) * np.sum(np.square(h - y))
    return J
```

computeCostMulti function is used to calculate the cost function (J).

In [20]:
```
history=[]
theta_min=[]
alpha = 0.03
num_iters = 100
for i in range(2,20):
    theta=np.zeros(i)
    theta, j_history = gradientDescentMulti(train_x[:,0:i], train_y, theta, alpha, num_iters)
    theta_min.append(theta)
    history.append(j_history[-1])
```

This part of code is used to call the gradientDescentMulti function and calculate the minimum cost and the thetas. For loop is used to increase the features(degree). The output of the for loop is the minimum of the cost and their corresponding thetas.

```
In [21]: validate_x = validate[:,3:]
         validate_y = validate[:,2]
         validate_m =validate_y.size
         validate_x = np.concatenate([np.ones((validate_m,1)), validate_x], axis =1)
```

```
In [22]: j_val=[]
         for i in range(0,18):
             theta=np.zeros(i+2)
             j= computeCostMulti(validate_x[:,0:i+2],validate_y, theta_min[i])
             j_val.append(j)
```

This part here is for the validation part. We use function computeCostMulti to calculate the cost and to check the result with the result of training data sets. To confirm which degree, have the minimum cost

Cost functions of the training data	Cost functions of the validations data
In [23]: history Out[23]: [60284630722.87962, 47905963063.939835, 34468486261.831245, 34407080709.64405, 34443736399.78116, 31878118438.035408, 30572893857.870087, 30133062616.085907, 27304429634.40463, 27440173101.172504, 26959306610.869987, 23490227450.54159, 23478832631.15971, 23504517967.29399, 20555749961.518303, 20397052711.460213, 20377630122.565228, 20355788973.765064]	In [24]: j_val Out[24]: [61225502289.460945, 51045252666.39353, 36388086416.360344, 36434233271.068794, 36444852942.050354, 33237340582.87855, 31592694896.637154, 31100245926.492447, 27626782511.97295, 27604310626.547485, 27267678457.65172, 23896240598.360474, 23976366530.71121, 24074651422.976078, 21030722767.94321, 20897304703.72019, 20586437010.723625, 20578795869.6969]

This table show that degree 18 has the lowest cost function in training and validation data

```
In [31]: j= computeCostMulti(test_x[:,0:-1],test_y, theta_min[-1-1])

In [32]: j
Out[32]: 21296484033.395107
```

Finally, the testing part we compute the cost using test input and output data and the theta of degree 18 which has the lost cost function.

Also when the features increase the cost function tend to be lower and the model shows more accuracy.