Practica 1

Regresión Lineal

**CÓDIGO FUENTE:**

import utils

import linear\_reg as ln

import public\_tests as test

import matplotlib.pyplot as plt

import numpy as np

def linear\_regression(w, b, x):

    return w \* x + b

def our\_test():

    X, Y =utils.load\_data()

    initial\_w = 2.0

    initial\_b = 3.0

    iterations = 1500

    alpha = 0.01

    w , b, history = ln.gradient\_descent(X, Y, initial\_w, initial\_b, ln.compute\_cost, ln.compute\_gradient, alpha , iterations)

    print("w, b found by gradient descent:", w, b)

    range\_MAX = np.max(X)

    range\_MIN = np.min(X)

    X\_fun = np.linspace(range\_MIN, range\_MAX, 256)

    Y\_fun = np.array(linear\_regression(w, b, X\_fun))

    plt.figure()

#Represent Data

    plt.plot(X\_fun, Y\_fun, c = 'blue', label = 'Regression : y = ' + "{:.2f}".format(w) + " \* " + "x + " + "{:.2f}".format(b))

    plt.scatter(X, Y, c = 'red', label = 'Data', marker= 'x')

    plt.legend()

    plt.show()

    plt.savefig('linearRegression\_prediction.png')

def execute\_tests():

    test.compute\_cost\_test(ln.compute\_cost)

    test.compute\_gradient\_test(ln.compute\_gradient)

def main():

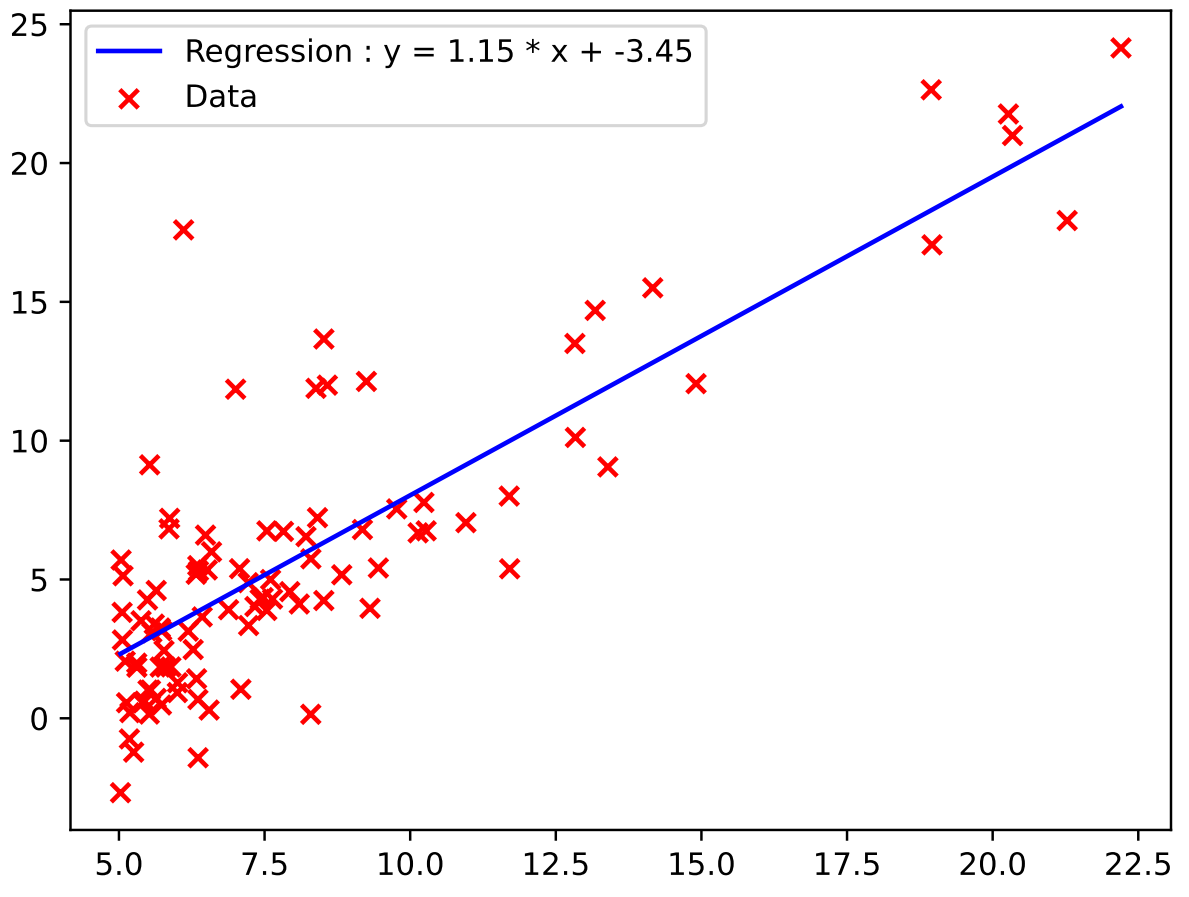
    our\_test()

    execute\_tests()

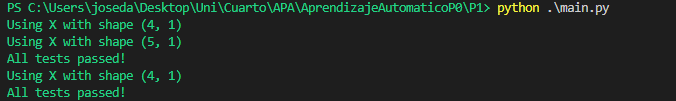
if \_\_name\_\_ == '\_\_main\_\_':

    main()

Ejecutando los datos almacenados en el archivo “ex1data1.txt”, obtenemos una gráfica con sus datos representados, y una recta de regresión que predice la salida en función de los datos de entrenamiento recibidos.  
  
**GRÁFICA:**



Para comprobar que los métodos programados funcionan correctamente, pasamos estos por las funciones de testeo proporcionadas en el archivo public\_test.py.



**IMPLEMENTACIÓN**

Los métodos implementados para el Descenso de Gradiente son:

**Calcular el costo de la función**

def compute\_cost(x, y, w, b):

    """

    Computes the cost function for linear regression.

    Args:

        x (ndarray): Shape (m,) Input to the model (Population of cities)

        y (ndarray): Shape (m,) Label (Actual profits for the cities)

        w, b (scalar): Parameters of the model

    Returns

        total\_cost (float): The cost of using w,b as the parameters for linear regression

               to fit the data points in x and y

    """

    m = x.shape[0]

    total\_cost = 0

    cost\_sum = 0

    for  i in range (m):

        #we calculate the function relative to the parameters

        f\_wb = w \* x[i] + b

        cost = (f\_wb - y[i])\*\*2

        cost\_sum = cost\_sum + cost

    total\_cost =  (1/ ( 2 \* m)) \* cost\_sum

    return total\_cost

**Calcular el gradiente**

#########################################################################

# Gradient function

#

def compute\_gradient(x, y, w, b):

    """

    Computes the gradient for linear regression

    Args:

      x (ndarray): Shape (m,) Input to the model (Population of cities)

      y (ndarray): Shape (m,) Label (Actual profits for the cities)

      w, b (scalar): Parameters of the model

    Returns

      dj\_dw (scalar): The gradient of the cost w.r.t. the parameters w

      dj\_db (scalar): The gradient of the cost w.r.t. the parameter b

     """

    m = x.shape[0]

    dj\_dw = 0

    dj\_db = 0

    for i in range(m):

        f\_wb = w \* x[i] + b

        dj\_db\_i = f\_wb - y[i]

        dj\_dw\_i = (f\_wb - y[i]) \* x[i]

        dj\_db += dj\_db\_i

        dj\_dw += dj\_dw\_i

    dj\_dw = dj\_dw / m

    dj\_db = dj\_db / m

    return dj\_dw, dj\_db

**Descenso de Gradiente**

#########################################################################

# gradient descent

#

def gradient\_descent(x, y, w\_in, b\_in, cost\_function, gradient\_function, alpha, num\_iters):

    """

    Performs batch gradient descent to learn theta. Updates theta by taking

    num\_iters gradient steps with learning rate alpha

    Args:

      x :    (ndarray): Shape (m,)

      y :    (ndarray): Shape (m,)

      w\_in, b\_in : (scalar) Initial values of parameters of the model

      cost\_function: function to compute cost

      gradient\_function: function to compute the gradient

      alpha : (float) Learning rate

      num\_iters : (int) number of iterations to run gradient descent

    Returns

      w : (ndarray): Shape (1,) Updated values of parameters of the model after

          running gradient descent

      b : (scalar) Updated value of parameter of the model after

          running gradient descent

      J\_history : (ndarray): Shape (num\_iters,) J at each iteration,

          primarily for graphing later

    """

    m = len(x)

    J\_history = []

    w = copy.deepcopy(w\_in)

    b = b\_in

    for i in range(num\_iters):

        dj\_dw, dj\_db = gradient\_function(x, y, w, b)

        w -= alpha \* dj\_dw

        b -= alpha \* dj\_db

        if i < 100000:

            cost = cost\_function(x,y,w,b)

            J\_history.append(cost)

    return w, b, J\_history