Jose Daniel Rave Robayo
Ángel López Benítez

Practica 1

Regresión Lineal

CÓDIGO FUENTE:

```python
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)
```

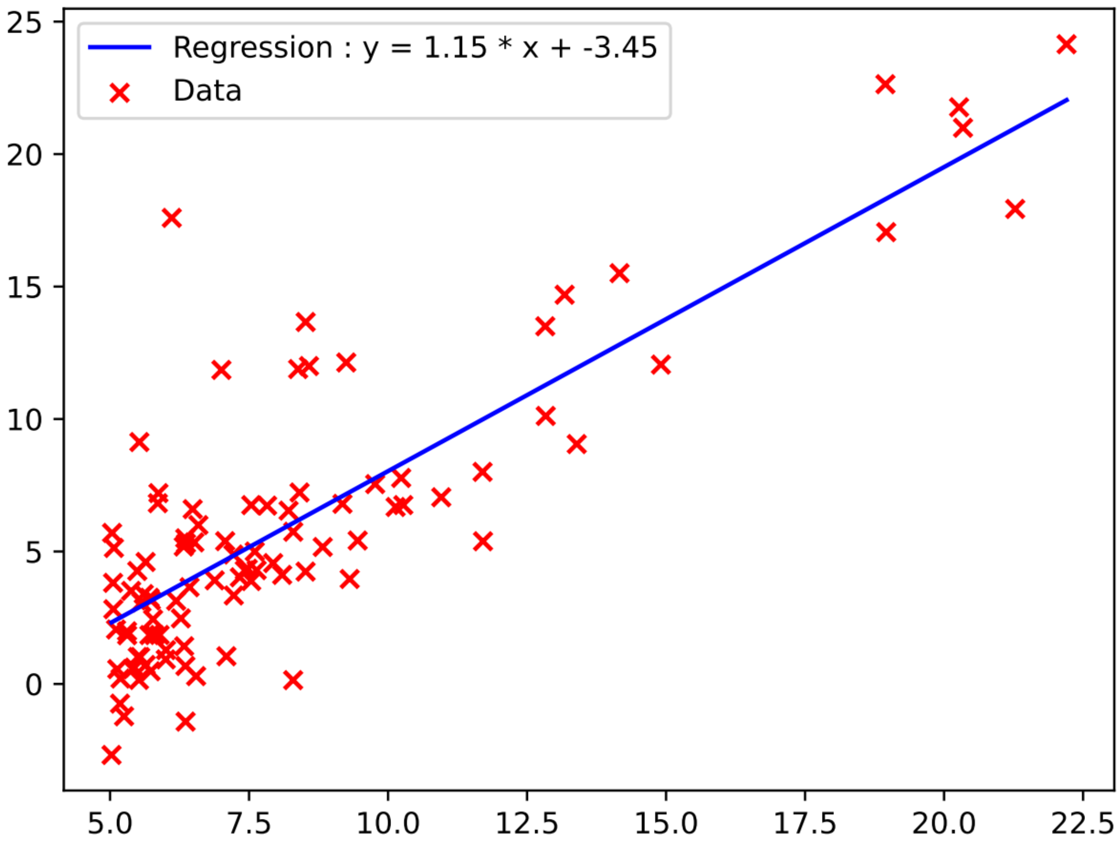
```
    test.compute_gradient_test(ln.compute_gradient)


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.

```
PS C:\Users\joseda\Desktop\Uni\Cuarto\APA\AprendizajeAutomaticoP0\P1> python .\main.py
Using X with shape (4, 1)
Using X with shape (5, 1)
All tests passed!
Using X with shape (4, 1)
All tests passed!
```

Jose Daniel Rave Robayo
Ángel López Benítez

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
```