Practica 1: Regresión lineal

Alberto Muñoz Fernández

Óscar García Castro

```
import numpy as np
import matplotlib.pyplot as plt
import copy
import math
import public_tests as tst
import utils

#######################################################################
# Cost function
#
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
    """
    sum = 0
    m = len(x)

    for i in range(m):
        sum += math.pow(w*x[i] + b - y[i], 2)

    return sum/(2*m)


#######################################################################
# 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 (ndarray): 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
     """
    dj_dw = 0
    dj_db = 0
    m = len(x)

    for i in range(m):
        dj_db += w*x[i] + b - y[i]
        dj_dw += (w*x[i] + b - y[i])*x[i]

    return dj_dw/m, dj_db/m


##########################################################################
# 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
    """
    J_history = []
    w = copy.deepcopy(w_in)
    b = b_in
    J_history += [cost_function(x, y, w, b)]
```

```
    for i in range(num_iters):
        gw, gb = gradient_function(x, y, w, b)
        w -= alpha*gw
        b -= alpha*gb
        J_history += [cost_function(x, y, w, b)]

    return w, b, J_history

def draw_data(x, y, w, b):
    plt.figure()
    plt.scatter(x, y, c='red', marker="x")
    plt.axline((0, b), (10, w*10 + b))
    plt.show()

def main():
    x, y = utils.load_data()
    #tst.compute_cost_test(compute_cost)
    tst.compute_gradient_test(compute_gradient)
    w, b, J_hist = gradient_descent(x, y, 0, 0, compute_cost,
compute_gradient, 0.01, 1500)
    draw_data(x, y, w, b)

main()
```

Resultados del test: