**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:

