Entrega Práctica 1: Regresión lineal

Autores	Correo
Clara Daniela Sima	csima@ucm.es
Stiven Arias Giraldo	starias@ucm.es

Parte 1 - Regresión lineal con una variable X

Cálculo iterativo del descenso de gradiente mediante regresión lineal con **una sola variable X** para determinar un valor **Y**, mediante el calculo progresivo de nuevos valores para **theta_0** y **theta_1** y **alpha constante** aprendiendo a través de datos recogidos en **ex1data1.csv**. Donde los datos de la primera columna representan la población de varios lugar y los beneficios que otorga una tienda en función de dicha población.

Sección de código

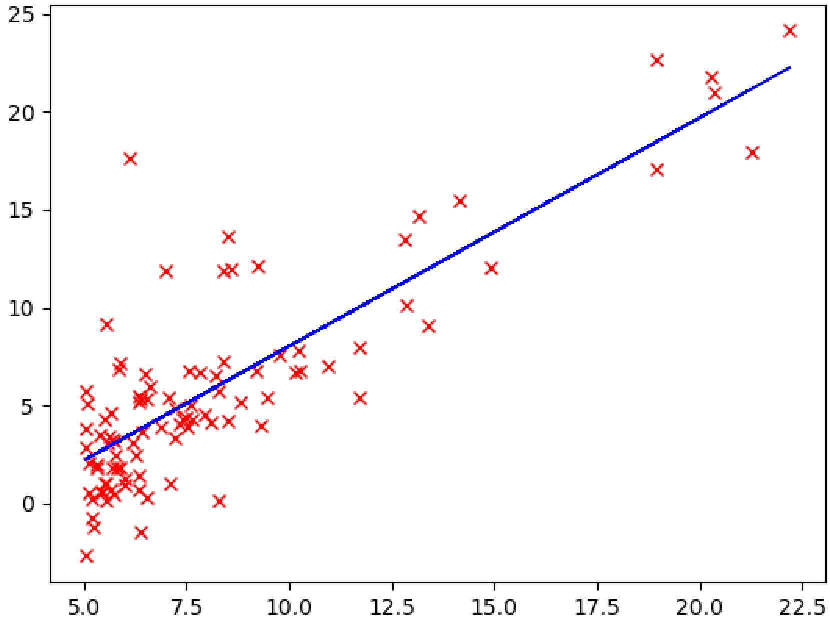
```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def function_J(m, X, Y, theta0, theta1):
    """
    Calculate the function J(theta)
    """
    sum = 0
    for i in range(m):
        sum = sum + (hipotesis(X[i], theta0, theta1) - Y[i]) ** 2

    result = (1 / (2 * m)) * sum
    return  result.astype(float)

def hipotesis(x, theta0, theta1):
    """
    Calculate the hypothesis function h(x) = theta0 + theta1 * x
    """
    result = theta0 + theta1 * x
    return result.astype(float)

def diff(x, y, theta0, theta1):
    """
        return h(xi) - yi
    """
    result = hipotesis(x, theta0, theta1) - y
    return result.astype(float)
```

```
def new_theta_0(m, X, Y, theta0, theta1, alpha):
    """
    Calculate the new value of theta0
    """
    sum = 0
    for i in range(m):
        sum = sum + diff(X[i], Y[i], theta0, theta1)

    result = theta0 - (alpha / m) * sum
    return  result.astype(float)

def new_theta_1(m, X, Y, theta0, theta1, alpha):
    """
    Calculate the new value of theta1
    """
    sum = 0
    for i in range(m):
        sum = sum + diff(X[i], Y[i], theta0, theta1) * X[i]

    result = theta1 - (alpha / m) * sum
    return  result.astype(float)

def read_data():
    """
    Read dthe data of the file and return the result as a float
    """
    valores = read_csv("./ex1data1.csv", header=None).to_numpy()
    return valores.astype(float)


def make_data(m, t0_range, t1_range, X, Y):
    """
    Calculate the matrix of Theta0 and Theta1.
    """
    step = 0.1
    Theta0 = np.arange(t0_range[0], t0_range[1], step)
    Theta1 = np.arange(t1_range[0], t1_range[1], step)
    Theta0, Theta1 = np.meshgrid(Theta0, Theta1)
    Coste = np.empty_like(Theta0)
    for ix, iy in np.ndindex(Theta0.shape):
        Coste[ix][iy] = function_J(m, X, Y, Theta0[ix][iy], Theta1[ix][iy])

    return [Theta0, Theta1, Coste]

def gradient():
    """
    Main function to calculate the descent of the gradient
    """
    # Valores de muestra
    valores = read_data()
    # Población
    X = valores[:, 0]
    # Beneficio
    Y = valores[:, 1]
```

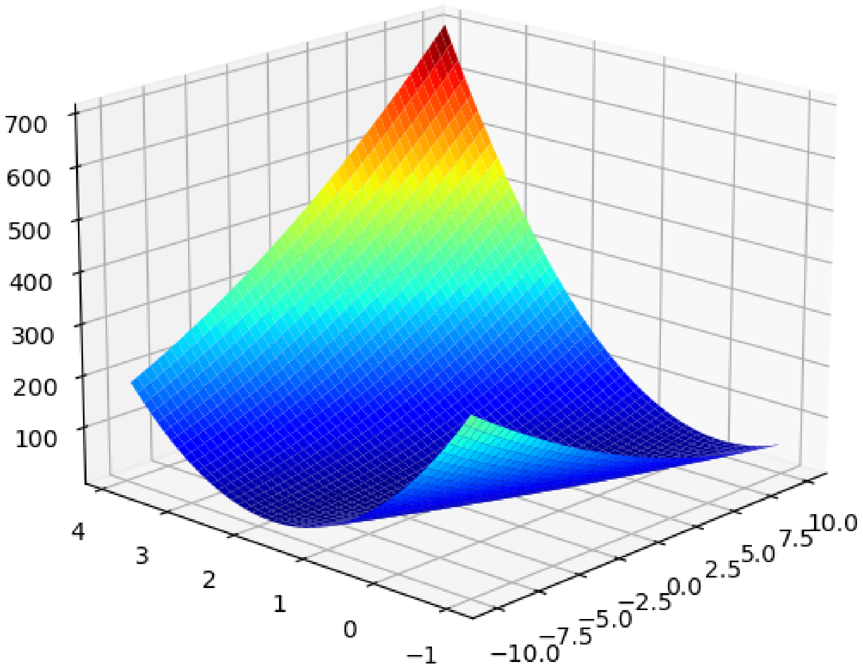
```
    # Número m de datos recogidos
    m  = len(X)

    theta0, theta1 = 0.0, 0.0
    alpha = 0.01

    # Current value of the function J(theta)
    curr_J = function_J(m, X, Y, theta0, theta1)
    min_J = curr_J
    min_t0, min_t1 = 0.0, 0.0

    for i in range(1500):
        # Calculate the new values to theta_0 and theta_1
        temp0  = new_theta_0(m, X, Y, theta0, theta1, alpha)
        temp1  = new_theta_1(m, X, Y, theta0, theta1, alpha)
        theta0, theta1 = temp0, temp1

        # Calculate the new cost of J function
        curr_J = function_J(m, X, Y, theta0, theta1)
        if curr_J < min_J:
            min_J = curr_J
            min_t0, min_t1 = theta0, theta1

        #print("Para theta0 = {} // theta1 = {}: J = {}".format(theta0, theta1,
curr_J))

    # Graph drawing
    makeData = make_data(m, [-10, 10], [-1, 4], X, Y)
    #print(makeData)

    fig = plt.figure()

    # Lineal Function Graph
    #plt.plot(X, Y, "x", c='red')
    #C = min_t0 + min_t1 * X
    #plt.plot(X, C, color="blue", linewidth=1.0, linestyle="solid")

    # 3D graph
    #ax = Axes3D(fig)
    #ax.plot_surface(makeData[0], makeData[1], makeData[2], cmap='jet')

    # Contour Graph
    plt.contour(makeData[0], makeData[1], makeData[2], np.logspace(-2, 3, 20),
cmap='jet')
    plt.plot(min_t0, min_t1, "x")

    plt.show()


# main
gradient()
```

Gráficas - Parte 1

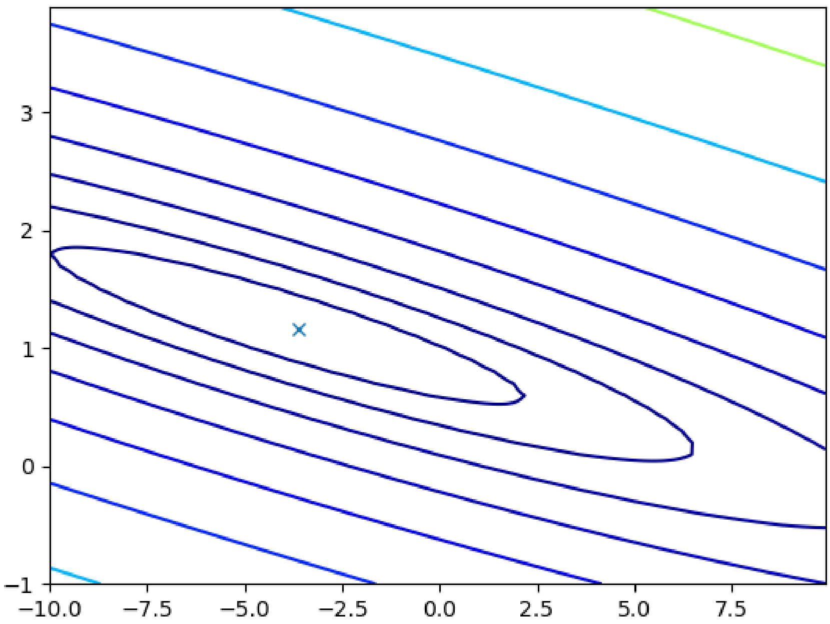
Distibución de los datos en función de X = población , Y = beneficios y la función de regresión lineal obtenida para determinar futuros valores de los beneficios en función de la población que tenga.



Representa el descenso del gradiente para X = población, Y = beneficios y Z = Costes (obtenidos mediante la función de coste J).



Representa el descenso de gradiente mediante una gráfica de contorno en un determinado rango de valores.



Parte 2 - Regresión lineal con múltiples variables X vectorizado y con ecuaión normal

Cálculo del descenso de gradiente mediante regresión lineal con **múltiples variables X** para determinar un valor **Y**, mediante el calculo progresivo de nuevos valores para diferentes **Theta_i** y **alpha constante** aprendiendo a través de datos recogidos en **ex1data2.csv**. Donde los datos de las columnnas representan respectivamente el tamaño de un piso en pies cuadrados, número de habtiaciones y el precio.

- Cálculo vectorizado - Ecuación normal -

Sección de código

```
import numpy as np
from numpy.lib import diff
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt

avg = 0

def read_data():
    """
    Reads the data of the file and return the result as a float
    """
    valores = read_csv("./src/ex1data2.csv", header=None).to_numpy()
    return valores.astype(float)

def function_J(m, X, Y, Theta):
```

```
    """
    Calculates the cost J with vectors
    """
    X_Theta = np.dot(X, Theta)
    diff = X_Theta - Y

    return (1 / (2 * m)) * np.transpose(diff) * diff

def new_Theta(m, n, alpha, Theta, X, Y):
    """
    Calculates the new values of the Theta matrix
    """
    # The new value of theta
    NewTheta = Theta

    # Contains the hypotesis function of every row
    H = np.matmul(X, NewTheta)

    # diff
    Diff = H - Y

    # Calculate every new Theta of the matrix Theta
    for i in range(n):
        Prod = Diff * X[:, i]
        NewTheta[i] -= (alpha / m) * Prod.sum()

    return NewTheta

def normalize_X(X):
    """
    Normalizes each value of the X array: xi = [xi - average(x)] / desviation_i
    """
    avg = np.mean(X)
    desv = np.std(X)
    # When all values are the same, the desv = 0
    if desv == 0:
        return X

    return [((x - avg) / desv).astype(float) for x in X]

def add_colum_ones():
    return 1

def gradient():
    valores = read_data()
    # Add all the rows and the col(len - 1)
    X = valores[:, :-1]
    # The -1 value add the col(len - 1)
    Y = valores[:, -1]
    # Row X
    m = np.shape(X)[0]
    # Cols X
    # Add a column of 1's to X
    X = np.hstack([np.ones([m, 1]), X])
```

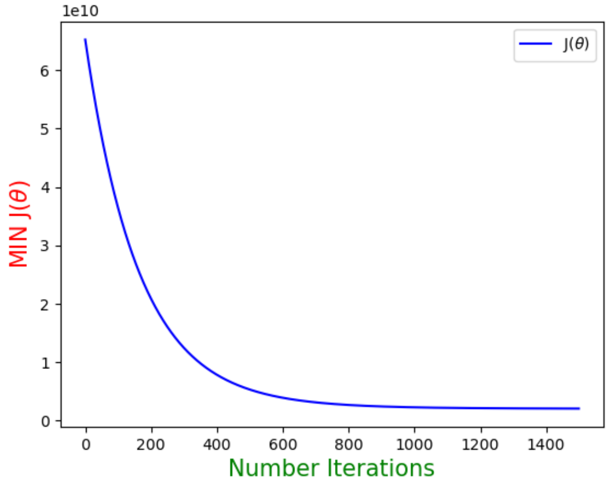
```
    n = np.shape(X)[1]

    for i in range(n):
        aux = X[:, i]
        #print("X before: {}".format(aux))
        X[:, i] = normalize_X(aux)
        #print("X after: {}".format(aux))

    # Theta need to have the same values as the columns of X
    Theta = np.zeros(n)
    alpha = 0.03

    # No. experiments
    exp = 1500
    # The X values for the graph
    axisX = np.arange(0, exp)
    # The Y values for the graph
    axisY = np.zeros(exp)

    for i in range(exp):
        # New Values of Theta
        Theta = new_Theta(m, n, alpha, Theta, X, Y)
        # Min J
        J = function_J(m, X, Y, Theta)
        axisY[i] = J.sum()

    #fig = plt.figure()
    #plt.title(r'$\alpha$: ' + str(alpha))
    #plt.xlabel('Number Iterations', c = 'green', size='15')
    #plt.ylabel(r'MIN J($\theta$)', c = 'red', size = '15')
    #plt.plot(axisX, axisY, "-", c='blue', label = r'J($\theta$)')
    #plt.legend(loc='upper right')
    #plt.show()
    return Theta

def new_normal_Theta(X, Y):
    #Theta = (X(trans) * X)^-1 * X(trans) * Y
    X_t = np.transpose(X)
    Prod = np.dot(X_t, X)
    Inv = np.linalg.pinv(Prod)
    b = np.dot(Inv, X_t)
    newTheta = np.matmul(b, Y)

    return newTheta

def normal_equation():
    valores = read_data()
    # Add all the rows and the col(len - 1)
    X = valores[:, :-1]
    # The -1 value add the col(len - 1)
    Y = valores[:, -1]
    # Row X
    m = np.shape(X)[0]
    # Cols X
```

```python
    # Add a column of 1's to X
    X = np.hstack([np.ones([m, 1]), X])
    n = np.shape(X)[1]

    Theta = np.zeros(n)

    Theta = new_normal_Theta(X, Y)

    return Theta

def hypotesis(Theta, X):
    H = np.matmul(X, Theta)

    return H.sum()

def calculate_normal_hypotesis(X, Theta):
    valores = read_data()
    # Add all the rows and the col(len - 1)
    X = valores[:, :-1]
    # Row X
    m = np.shape(X)[0]
    # Cols X
    # Add a column of 1's to X
    X = np.hstack([np.ones([m, 1]), X])
    n = np.shape(X)[1]
    X = np.vstack([X, [1, 1650, 3]])

    for i in range(n):
        aux = X[:, i]
        X[:, i] = normalize_X(aux)

    h = hypotesis(Theta, X[m])
    return h


Theta = gradient()
NormalTheta = normal_equation()
X = [1, 1650, 3]

print("Hipotesis: ", calculate_normal_hypotesis(X, Theta))
print("Normal hipotesis: ", hypotesis(NormalTheta, X))
print("Theta: ", Theta)
print("Theta shape: ", np.shape(Theta))
print("Normal Theta: ", NormalTheta)
print("Normal Theta shape: ", np.shape(NormalTheta))
```

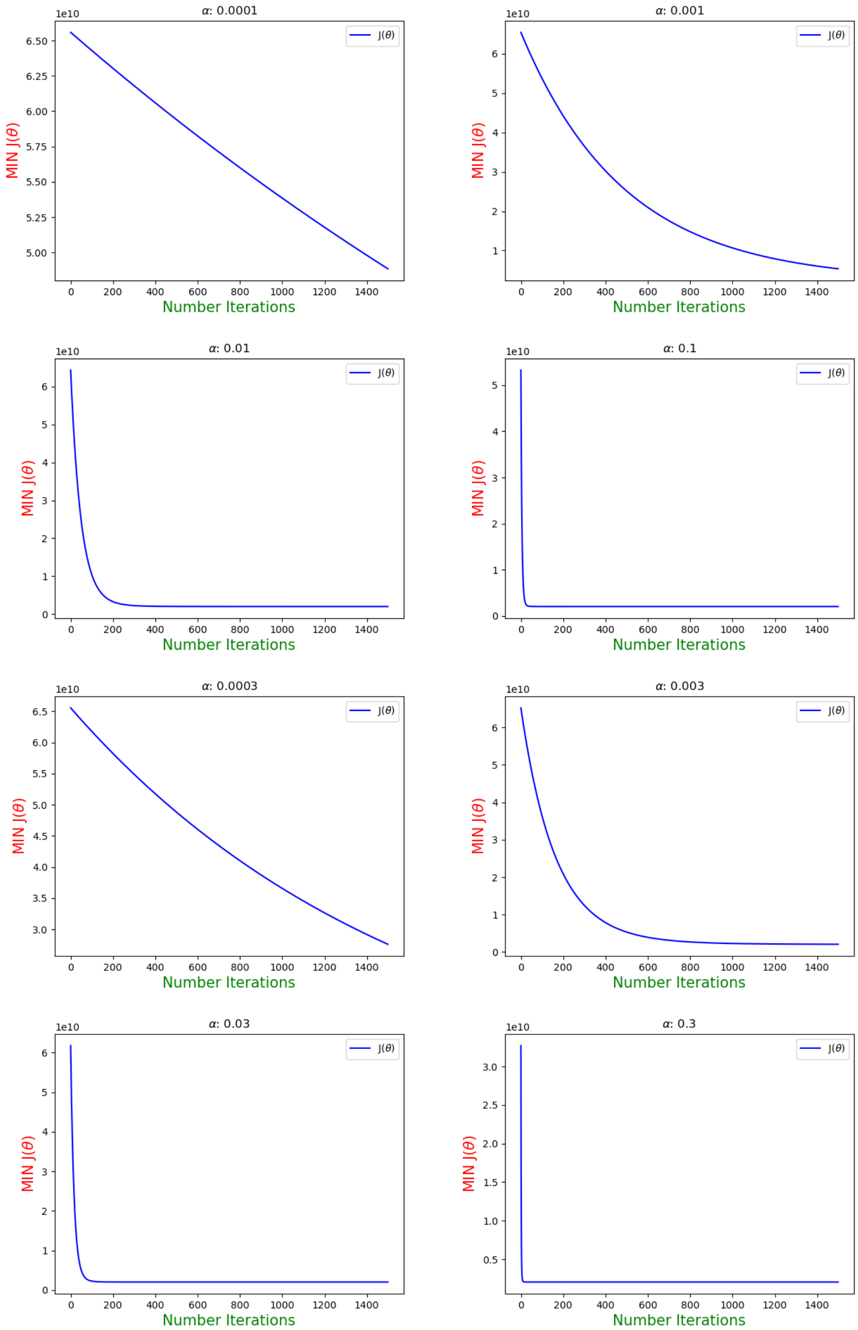
Gráficas - Parte 2

- Cálculo vectorizado -

Progresión del coste de J en cada nuevo experimento realizado



Cálculos del coste de J con diferentes valores de alpha



Salida de consola para comprobar los datos de la ecuación normal y el cálculo vectorizado

```
Hipotesis:  293676.11388239474
Normal hipotesis:  293081.4643349892
Theta:  [340412.65957447 109447.79634183  -6578.35472634]
Theta shape:  (3,)
Normal Theta:  [89597.90954361   139.21067402 -8738.01911255]
Normal Theta shape:  (3,)
PS J:\Cuarto\AA\Practica1>
```