Aprendizaje Automático y minería de datos: Práctica 1

Autores: *Tatiana Duarte Balvís, Miguel Mur Cortés*

1. *Objetivo:*

El objetivo de la práctica es aplicar el método de regresión lineal sobre distintos conjuntos de datos. En nuestro caso, analizaremos en la primera parte la relación entre los beneficios de una compañía de comida en base a la población de la ciudad. En la segunda parte, analizaremos la relación entre tres variables: El precio de una casa, el número de habitaciones y su superficie.

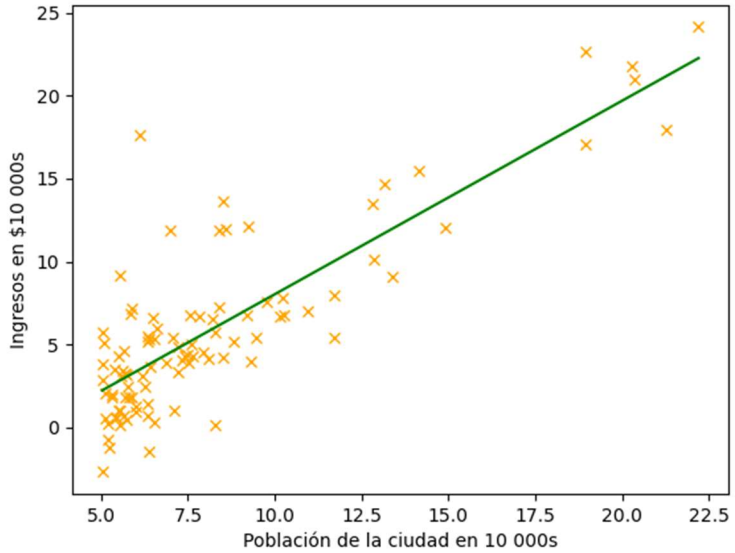
2. *Regresión lineal con una variable:*

En esta función se implementa el método de descenso de gradiente para minimizar la función de coste. Devolvemos los valores mínimos θ_0 y θ_1. Este método sigue la siguiente fórmula:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

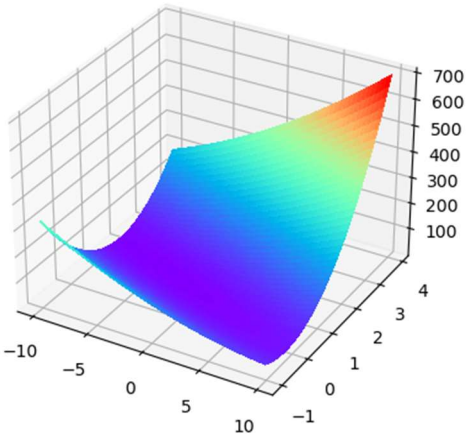
```
def descenso_gradiente(x, y, alpha = 0.01, num_it = 1500):
    m = len(x)
    theta0 = theta1 = 0

    for aux in range(num_it):
        sum0 = sum1 = 0
        for i in range(m): #h = theta0 + theta1 * x[i]
            sum0 += (theta0 + theta1 * x[i]) - y[i]
            sum1 += ((theta0 + theta1 * x[i]) - y[i]) * x[i]
        theta0 = theta0 - (alpha/m) * sum0
        theta1 = theta1 - (alpha/m) * sum1

    return (theta0, theta1)


def coste(x, y, arrayThetas): #Funcion de coste
    m = len(x)
    sum0 = 0
    for i in range(m):
            sum0 += ((arrayThetas[0] + arrayThetas[1] * x[i]) - y[i])**2

    return sum0 / (2*m)
```

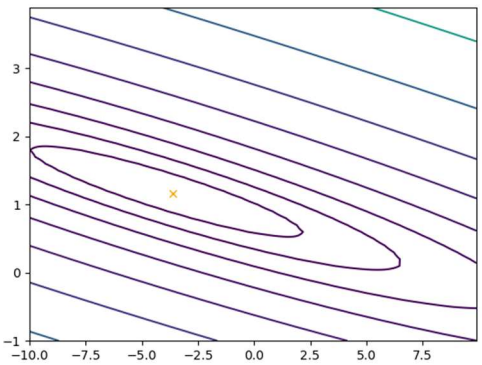
Con los datos proporcionados, obtenemos los parámetros θ que mejor definen la recta que se ajusta a los datos de entrenamiento.



Para visualizar la función de coste, hemos implementado este método:

```
def make_data(rangoT0, rangoT1, x, y):
    step = 0.1
    theta0 = np.arange(rangoT0[0], rangoT0[1], step)
    theta1 = np.arange(rangoT1[0], rangoT1[1], step)
    theta0, theta1 = np.meshgrid(theta0, theta1)

    costeF = np.empty_like(theta0)
    for ix, iy in np.ndindex(theta0.shape): #Itera por todas las dimens
ione s de la matriz
        costeF[ix, iy] = coste(x, y, [theta0[ix, iy], theta1[ix, iy]])

    return (theta0, theta1, costeF)
```





3. Regresión lineal con varias variables:

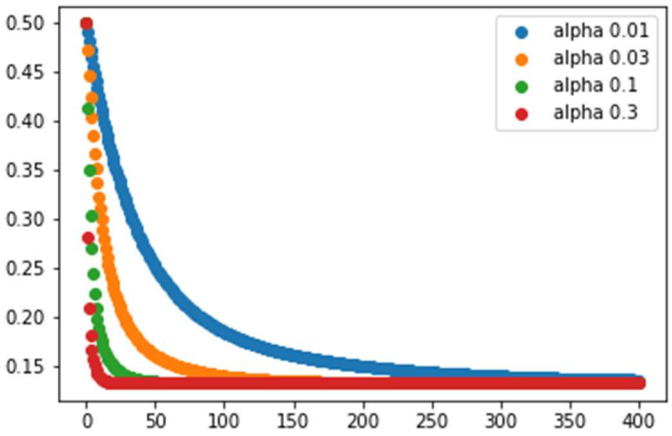
Para realizar el descenso de gradiente sobre los datos proporcionados, es necesario normalizar los datos:

```
def matriz_norm(x):
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0)
    x_norm = (x - mu) / sigma
    return (x_norm, mu, sigma)
```

Una vez normalizados los datos, aplicamos el descenso de gradiente.

```
def coste_vect(x, y, theta):
     h = np.dot(x, theta)
     aux = (h - y) ** 2
     return aux.sum() / (2 * len(x))

def gradiente(x, y, Theta, alpha):
    NuevaTheta = Theta
    m = np.shape(x)[0]
    n = np.shape(x)[1]
    h = np.dot(x, Theta)
    aux = (h - y)
    for i in range(n):
        aux_i = aux * x[:, i]
        NuevaTheta[i] -= (alpha / m) * aux_i.sum()
    return NuevaTheta

def descenso_gradiente_vect(x, y, alpha, num_it):
    thetas = np.zeros(np.shape(x)[1])
    costes = np.zeros(num_it)

    for i in range(num_it):
        aux = gradiente(x, y, thetas, alpha)
        costes[i] = coste_vect(x, y, thetas)
        thetas = aux

    return thetas, costes
```

Experimentando con distintos valores α, construimos una gráfica donde se muestra la evolución de la función de coste a medida que avanza el descenso de gradiente.



Los valores θ óptimos también se pueden obtener en un solo paso utilizando el método de la ecuación normal según la siguiente fórmula:

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

```
def ecuacion_normal(x, y):
    x_t = np.transpose(x)
    aux = np.linalg.pinv(np.dot(x_t, x))
    aux = np.dot(aux, x_t)
    return np.dot(aux, y)
```

Para demostrar que nuestros cálculos son correctos, comprobamos los resultados obtenidos por ambos métodos sobre un ejemplo de una casa con una superficie de 1.650 pies cuadrados y 3 habitaciones.

```
def compara_funciones():
    datos = carga_csv("ex1data2.csv")
    x = datos[:, :-1]
    y = datos[:, -1]
    x_norm, mu, sigma = matriz_norm(x)
    x = np.hstack([np.ones([np.shape(x)[0],1]), x])
    x_norm = np.hstack([np.ones([np.shape(x)[0],1]), x_norm])

    resultado_gradiente = descenso_gradiente_vect(x_norm, y, 0.01, 800)
    resultado_normal = ecuacion_normal(x, y)

    prediccion_gradiente = resultado_gradiente[0][0] + resultado_gradie
nte[0][1] * ((1650 - mu[0]) / sigma[0])
    + resultado_gradiente[0][2] * ((3 - mu[1]) / sigma[1])
    prediccion_normal = resultado_normal[0] + resultado_normal[1] * 165
0 + resultado_normal[2] * 3

    print("Descenso de gradiente: " + str(prediccion_gradiente))
    print("Ecuación normal: " + str(prediccion_normal))
```

4. Código:

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

def carga_csv(file_name):
    valores = read_csv(file_name,  header=None).to_numpy()
    return valores.astype(float)

def descenso_gradiente(x, y, alpha = 0.01, num_it = 1500):
    m = len(x)
    theta0 = theta1 = 0

    for aux in range(num_it):
        sum0 = sum1 = 0
        for i in range(m): #h = theta0 + theta1 * x[i]
            sum0 += (theta0 + theta1 * x[i]) - y[i]
            sum1 += ((theta0 + theta1 * x[i]) - y[i]) * x[i]
        theta0 = theta0 - (alpha/m) * sum0
        theta1 = theta1 - (alpha/m) * sum1

    return (theta0, theta1)


def coste(x, y, arrayThetas): #Funcion de coste
    m = len(x)
    sum0 = 0
    for i in range(m):
            sum0 += ((arrayThetas[0] + arrayThetas[1] * x[i]) - y[i])**
2

    return sum0 / (2*m)


def make_data(rangoT0, rangoT1, x, y):
    step = 0.1
    theta0 = np.arange(rangoT0[0], rangoT0[1], step)
    theta1 = np.arange(rangoT1[0], rangoT1[1], step)
    theta0, theta1 = np.meshgrid(theta0, theta1)

    costeF = np.empty_like(theta0)
    for ix, iy in np.ndindex(theta0.shape): #Itera por todas las dimens
iones de la matriz
        costeF[ix, iy] = coste(x, y, [theta0[ix, iy], theta1[ix, iy]])
```

```
    return (theta0, theta1, costeF)


def regresion_lineal_una_variable():
    datos = carga_csv("ex1data1.csv")
    x = datos[:, 0]
    y = datos[:, 1]

    thetas = descenso_gradiente(x, y)

    plt.xlabel("Población de la ciudad en 10 000s")
    plt.ylabel("Ingresos en $10 000s")
    plt.plot(x, y, "x", c="orange")
    minX = min(x)
    maxX = max(x)
    minY = thetas[0] + thetas[1] * minX
    maxY = thetas[0] + thetas[1] * maxX
    plt.plot([minX, maxX], [minY, maxY], c="green")
    plt.savefig("descensoGrad.png")

    x, y, z = make_data((-10, 10), (-1, 4), x, y)

    fig = plt.figure()
    ax = fig.gca(projection = "3d")
    ax.plot_surface(x, y, z, cmap = cm.rainbow, linewidth=0, antialiase
d=False)
    plt.savefig("grafica3D.png")

    plt.figure()
    plt.plot(thetas[0], thetas[1], "x", c="orange")
    plt.contour(x, y, z, np.logspace(-2, 3, 20))
    plt.savefig("graficaContorno.png")

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def matriz_norm(x):
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0)
    x_norm = (x - mu) / sigma
    return (x_norm, mu, sigma)


def coste_vect(x, y, theta):
     h = np.dot(x, theta)
     aux = (h - y) ** 2
     return aux.sum() / (2 * len(x))


def descenso_gradiente_vect(x, y, alpha, num_it):
```

```
    thetas = np.zeros(np.shape(x)[1])
    costes = np.zeros(num_it)

    for i in range(num_it):
        aux = gradiente(x, y, thetas, alpha)
        costes[i] = coste_vect(x, y, thetas)
        thetas = aux

    return thetas, costes

def gradiente(x, y, Theta, alpha):
    NuevaTheta = Theta
    m = np.shape(x)[0]
    n = np.shape(x)[1]
    h = np.dot(x, Theta)
    aux = (h - y)
    for i in range(n):
        aux_i = aux * x[:, i]
        NuevaTheta[i] -= (alpha / m) * aux_i.sum()
    return NuevaTheta

def regresion_varias_variables():
    datos = carga_csv("ex1data2.csv")
    datos_norm, mu, sigma = matriz_norm(datos)
    x = datos_norm[:, :-1]
    y = datos_norm[:, -1]

    x = np.hstack([np.ones([np.shape(x)[0],1]),x])

    plt.figure()
    alphas = [0.01, 0.03, 0.1, 0.3]
    for a in alphas:
        thetas, costes = descenso_gradiente_vect(x, y, a, 400)
        plt.scatter(np.arange(np.shape(costes)[0]), costes, label = 'al
pha ' + str(a))

    plt.legend()
    plt.savefig("resultadoAlphas.png")

def ecuacion_normal(x, y):
    x_t = np.transpose(x)
    aux = np.linalg.pinv(np.dot(x_t, x))
    aux = np.dot(aux, x_t)
    return np.dot(aux, y)

def compara_funciones():
    datos = carga_csv("ex1data2.csv")
    x = datos[:, :-1]
```

```
    y = datos[:, -1]
    x_norm, mu, sigma = matriz_norm(x)
    x = np.hstack([np.ones([np.shape(x)[0],1]), x])
    x_norm = np.hstack([np.ones([np.shape(x)[0],1]), x_norm])

    resultado_gradiente = descenso_gradiente_vect(x_norm, y, 0.01, 800)
    resultado_normal = ecuacion_normal(x, y)

    prediccion_gradiente = resultado_gradiente[0][0] + resultado_gradie
nte[0][1] * ((1650 - mu[0]) / sigma[0])
    + resultado_gradiente[0][2] * ((3 - mu[1]) / sigma[1])
    prediccion_normal = resultado_normal[0] + resultado_normal[1] * 165
0 + resultado_normal[2] * 3

    print("Descenso de gradiente: " + str(prediccion_gradiente))
    print("Ecuación normal: " + str(prediccion_normal))

regresion_lineal_una_variable()
#regresion_varias_variables()
#compara_funciones()
```