PRÁCTICA 1 - REGRESIÓN LINEAL

Autores:

- Amaro Blest Polo
- Raúl Blas Ruiz

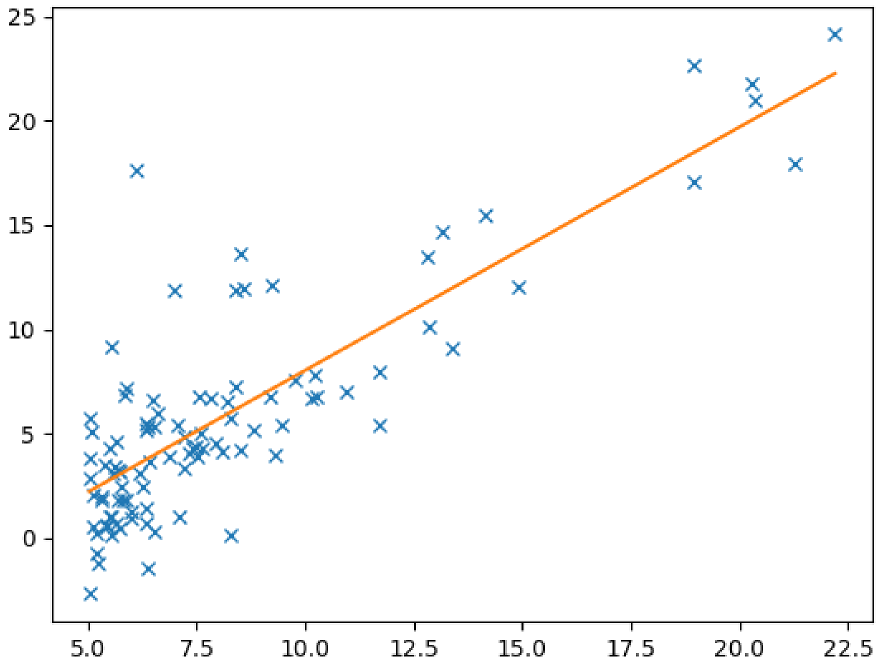
Apartado 1.0 (Regresión lineal con una variable) :

En la primera parte de la práctica aplicamos el método de regresión lineal sobre los datos del fichero ex1data1.csv que representan datos sobre los beneficios (segunda columna en el archivo) de una compañía de distribución de comida en distintas ciudades, en base a su población (primera columna en el archivo).

Para ello primero creamos el método *leeCSV* para leer ficheros .csv usando el método read_csv de Pandas que devuelve un DataFrame y nosotros lo transformamos en un numpy array usando la llamada *to_numpy()* para finalmente devolver los valores leídos en formato float:

```
# Lee un archivo csv pasando el nombre del fichero a leer y devuelve un array de numpy
def leeCSV(file_name):
    """carga el fichero csv especificado y lo
     devuelve en un array de numpy"""
    valores = read_csv(file_name, header = None).to_numpy()
    return valores.astype(float)
```

A continuación, creamos el método *regLinealUnaVariable()*, el cual llama a *leeCSV()* para leer el archivo ex1data1.csv y almacenar en un array X los elementos de la primera columna del archivo y en un array Y los de la segunda. Después aplicamos el algoritmo de descenso de gradiente para calcular los valores theta_0 y theta_1 para posteriormente dibujar la gráfica donde se puede ver reflejado gráficamente el algoritmo:




```python
# Metodo de descenso de gradiente para una sola variable
def regLinealUnaVariable():
    datos = leeCSV("ex1data1.csv")
    #   los arrays de la siguiente manera [:, 0] nos devuelve en X los elementos de la primera columna
    X = datos[:, 0]
    #   los arrays de la siguiente manera [:, 1] nos devuelve en Y los elementos de la primera columna
    Y = datos[:, 1]
    #   m => es el rango de puntos que existen
    m = len(X)
    #   Ratio de aprendizaje del algoritmo de descenso de gradiente
    alpha = 0.01
    #   theta_0 => eje x
    #   theta_1 => eje z
    theta_0 = theta_1 = 0
    #
    for _ in range(1500):
        sum_0 = sum_1 = 0
        #   h0(x) por el modelo lineal es = theta_0 + theta_1 * x
        for i in range(m):
            #   valores de los sumatorios para la formula del descenso de gradiente
            sum_0 += (theta_0 + theta_1 * X[i]) - Y[i]
            sum_1 += ((theta_0 + theta_1 * X[i]) - Y[i]) * X[i]
```

```python
        #   formulas del gradiente para theta0 y theta1
        theta_0 = theta_0 - (alpha / m) * sum_0
        theta_1 = theta_1 - (alpha / m) * sum_1

    #   dibujamos la grafica
    plt.plot(X, Y, "x")
    min_x = min(X)
    max_x = max(X)
    min_y = theta_0 + theta_1 * min_x
    max_y = theta_0 + theta_1 * max_x
    plt.plot([min_x, max_x], [min_y, max_y])
    plt.savefig("resultado.png")
```

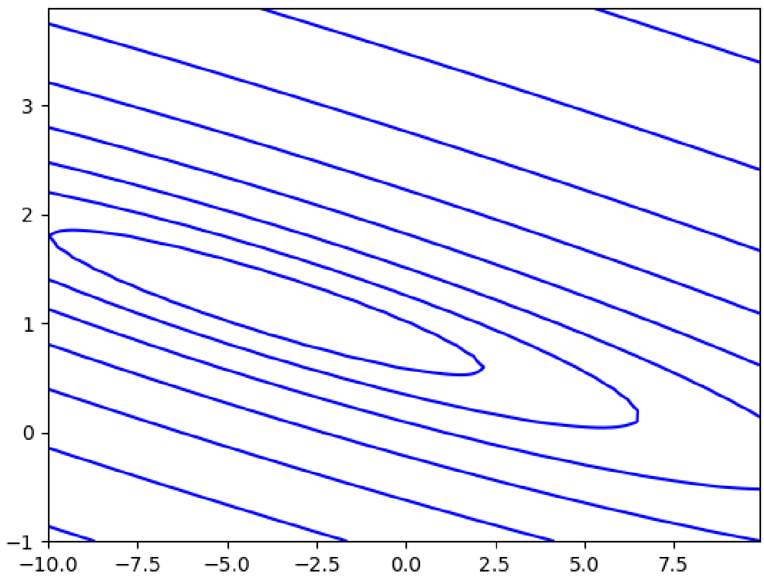
Apartado 1.1 (Visualización de la función de coste) :

En este apartado generamos las gráficas 2D y 3D de la función de coste en unos determinados rangos, en este caso para theta0 será el rango [-10, 10] y para theta1 el rango [-1, 4], de los valores del archivo ex1data1.csv. Para realizar esto, creamos el método *makeData()* que nos devolverá un array Theta0 con los valores desde -10 hasta 10 de 0.1 en 0.1, un array Theta1 con los valores desde -1 hasta 4 de 0.1 en 0.1, y un array con los costes de la función para todas esas tuplas [Theta0[ix, iy], Theta1[ix, iy]].

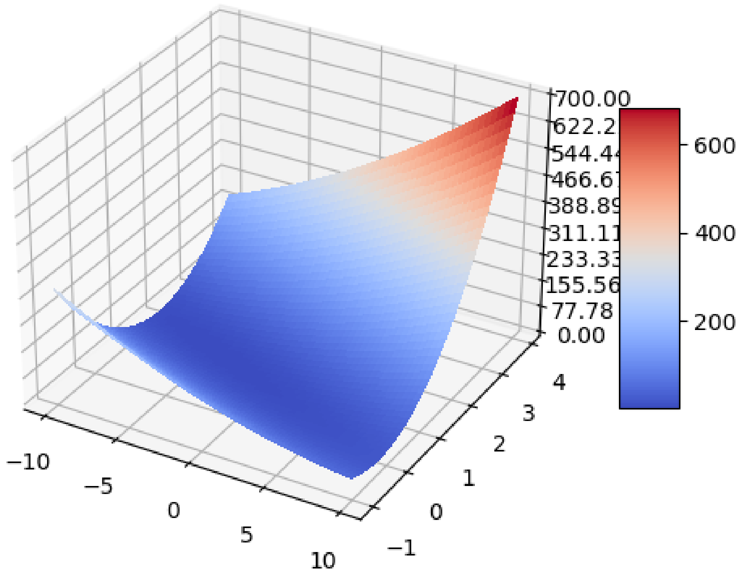
```python
def makeData(t0_range, t1_range, X ,Y):
    step = 0.1
    # np.arange nos crea un numpy array desde t0_range[0] hasta t0_range[1] de step en step
    # ejemplo: np.arange(2, 10, 3) da de resultado [2, 5, 8]
    Theta0 = np.arange(t0_range[0], t0_range[1], step) # [-10, -9.9, -9.8....10]
    Theta1 = np.arange(t1_range[0], t1_range[1], step) # [-1, -0.9, -0.8...4]
    #   Para generar el grid de valores de theta0 = 0 y theta1 = 0 en sus intervalos
    Theta0, Theta1 = np.meshgrid(Theta0, Theta1)

    #   Creamos una copia de theta0
    Coste = np.empty_like(Theta0)
    #   Para todos los elementos de theta0
    for ix, iy in np.ndindex(Theta0.shape):
        #   Actualizamos valor de coste
        Coste[ix, iy] = coste(X, Y ,[Theta0[ix, iy], Theta1[ix, iy]])

    #   Devolvemos la tupla
    return [Theta0, Theta1, Coste]
```

Finalmente dibujamos las representaciones 2D y 3D de la función de coste con los métodos *contourGraph()* y *surfaceGraph()* respectivamente:

```python
def countourGraph(a, b, c):
    plt.contour(a, b, c, np.logspace(-2, 3, 20), colors='blue')
    plt.savefig("representacion2D.png")
```



```python
def surfaceGraph(a, b, c):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(a, b, c, cmap = cm.coolwarm, linewidth = 0, antialiased = False)
    # Customizar el eje Z
    ax.set_zlim(0, 700)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.savefig("representacion3D.png")
```



Apartado 2.0 (Regresión con varias variables) :

Hemos hecho un método para devolver tanto la matriz normalizada, la media y la desviación. Para realizar la media hemos utilizado el método *mean* de numpy. Para realizar la desviación hemos utilizado el método *std* de la librería numpy. Y finalmente para normalizar la matriz hicimos lo siguiente: (matriz - media) / desviación.

```python
#   Devuelve en función de la matriz X, la misma matriz normalizada, la media
#   y desviación estandar de cada atributo
def normalizeMat():
    X_to_norm = np.empty(X.shape)
    mu = np.empty(n+1) #3 columnas para la media de cada atributo
    sigma = np.empty(n+1) #3 columnas para la desviación típica de cada atributo

    for i in range(n+1): #Recorremos ambas columnas
        mu[i] = np.mean(X[:, i])
        sigma[i] = np.std(X[:, i])
        if sigma[i] != 0:
            aux = (X[:, i] - mu[i])/sigma[i]
        else:
            aux = 1
        X_to_norm[:, i] = aux

    return X_to_norm, mu, sigma
```

Apartado 2.1 (Implementación vectorizada del descenso de gradiente) :
Para implementar este apartado empezamos por leer los datos desde *"ex1data2.csv"*, creamos dos arrays con los datos obtenidos y realizamos la normalización de X utilizando el método del apartado 2.0. Posteriormente llamamos al método que implementa el algoritmo del descenso de gradiente y finalmente representamos en una gráfica la evolución de los costes, apreciando la bajada de costes.

```
valoresCasas = leeCSV("ex1data2.csv")
X = valoresCasas[:, :-1]
Y = valoresCasas[:, -1]
m = np.shape(X)[0]
n = np.shape(X)[1]

# Añadimos una columna de 1's a la X para poder multiplicar con theta
X = np.hstack([np.ones([m, 1]), X])

matrizNorm, media, desviacion = normalizeMat()

alpha = 0.01


Theta, costes = gradiente()
evolucion_coste()
```

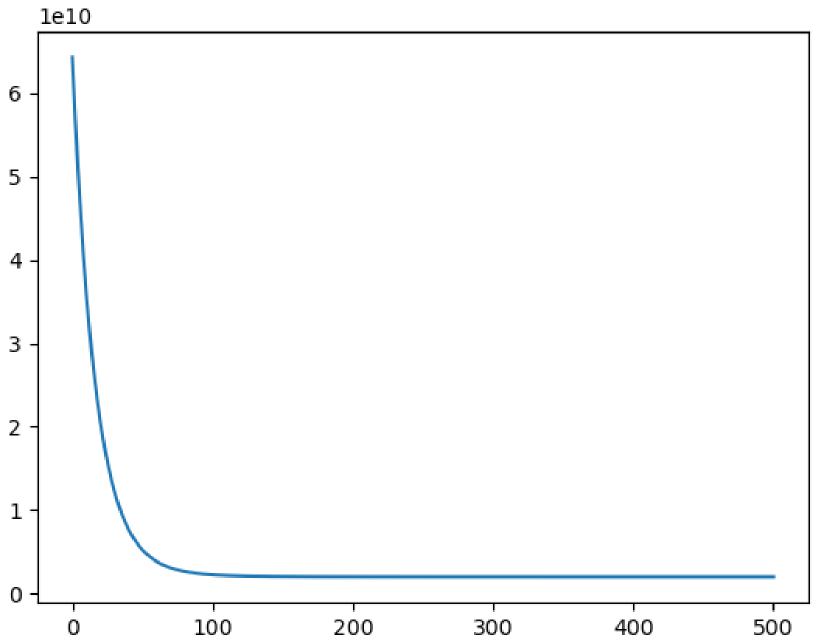
A continuación se muestra la implementación del método gradiente.

```
# Metodo de descenso de gradiente para mas de una variable
def gradiente():
    #inicializamos theta como un vector de ceros de tamaño 3
    theta = np.zeros(n+1, dtype=float)
    #inicializamos costes como un vector de ceros de tamaño 1500 (iteraciones)
    costes = np.zeros(1500, dtype=float)

    for i in range(1500):
        #calculamos h de theta usando la transpuesta de theta por la matriz normalizada
        H = np.dot(matrizNorm, np.transpose(theta))
        #a continuacion calcularemos el valor cada valor de theta, es decir un valor por
        # cada variable
        for j in range(np.shape(matrizNorm)[1]):
            aux_j = (H - Y) * matrizNorm[:, j]
            theta[j] -= (alpha / m) * aux_j.sum()
            #calculamos el coste hasta el theta que llevamos
        costes[i] = costeVariables(matrizNorm, Y, theta)
    return [theta, costes]
```

Para representar gráficamente el coste utilizamos el siguiente método.

```
# Metodo para dibujar la evolucion del coste en la regresion lineal con varias variables
def evolucion_coste():
    plt.figure()
    x = np.linspace(0, 500, 1500, endpoint = True)
    x = np.reshape(x, (1500, 1))
    plt.plot(x, costes)
    plt.savefig("evolucionCostes.png")
```

Este es el resultado de dicha gráfica.



Apartado 2.2 (Ecuación normal) :

Para la ecuación normal hemos utilizado el siguiente método y hemos comprobado que es una alternativa válida al descenso de gradiente.

```
# Ecuacion normal
def ecuacion_normal():
    X_tran = np.transpose(X)
    inver = np.linalg.pinv(np.dot(X_tran, X))
    p = np.dot(inver, X_tran)
    return np.matmul(p, Y)
```