Practica 1: Regresión Lineal

Nuria Bango Iglesias (nubango@ucm.es)

Álvar Julián de Diego López (alvarded@ucm.es)

1. Regresión lineal con una variable

En esta primera parte de la práctica vamos a aplicar el método de regresión lineal sobre los datos del fichero ex1data1.csv, que representan datos sobre los beneficios de una compañia en base a la población de distintas ciudades.

Primero cargamos los datos y los pasamos a un array de numpy.

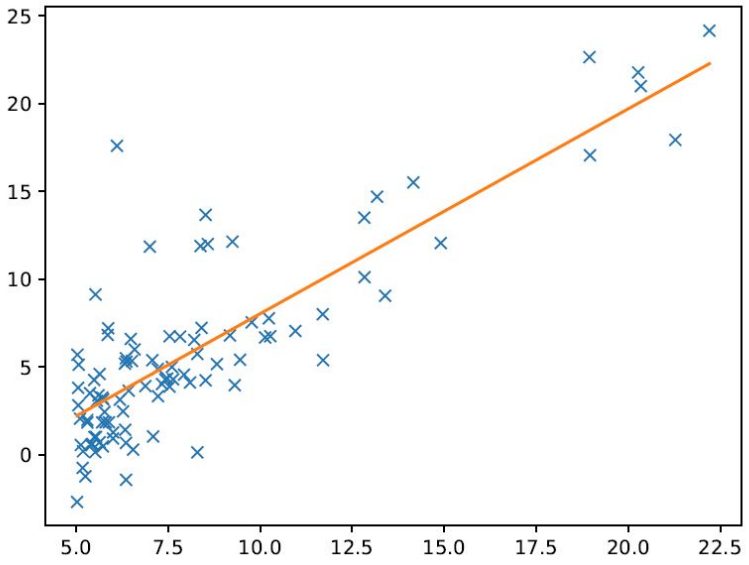
```
def carga_csv(file):
    valores = read_csv ( file ,  header=None) . to_numpy ()
    return valores.astype(float)
```

Tenemos que encontrar los valores de θ adecuados para generar una hipótesis $h_\theta(x) = \theta_0 + \theta_1 x$ minimizando la función de coste $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$. Para ello utilizamos el método de descenso de gradiente acercándonos iterativamente al valor de theta que minimiza la función de coste.

```
X = datos[:, 0]
Y = datos[:, 1]
m = len(X)
alpha = 0.01
#inicializamos theta0 y theta1 a 0
theta_0 = theta_1 = 0
#subrayado para que no se defina ninguna variable nueva
for _ in range(1500):
    sum_0 = sum_1 = 0
    for i in range(m):
        sum_0 +=  (theta_0 + theta_1 * X[i]) - Y[i]
        sum_1 += ((theta_0 + theta_1 * X[i]) - Y[i]) * X[i]
    theta_0 = theta_0 - (alpha / m) * sum_0
    theta_1 = theta_1 - (alpha / m) * sum_1
```

Utilizando este código hemos obtenido los valores de theta0 y theta1 que definen la recta que minimiza el coste. En la siguiente gráfica se puede observar la recta de regresión

obtenida con las thetas. Donde el eje x representa la poblacion de las ciudades el 10ks yel eje y representa los ingresos en 10ks.



Esta gráfica la hemos obtenido con el siguiente código.

```
#grafica recta con menos coste a raiz de encontrar theta0 y theta1
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.pdf")
```

1.1 Visualización de la función de coste

Para comprender el comportamiento de la funcion de coste J(θ) vamos a generar gráficas que muestren su comportamiento.

```
#grafica 3D
fig = plt.figure()
ax= fig.gca(projection = '3d')  #ax=Axes3D(fig)

A, B, Z = make_data([-5, 5], [-5, 5], X, Y)

surf = ax.plot_surface(A, B, Z, cmap=cm.coolwarm, linewidth = 0, antialiased = False)

plt.show()
```

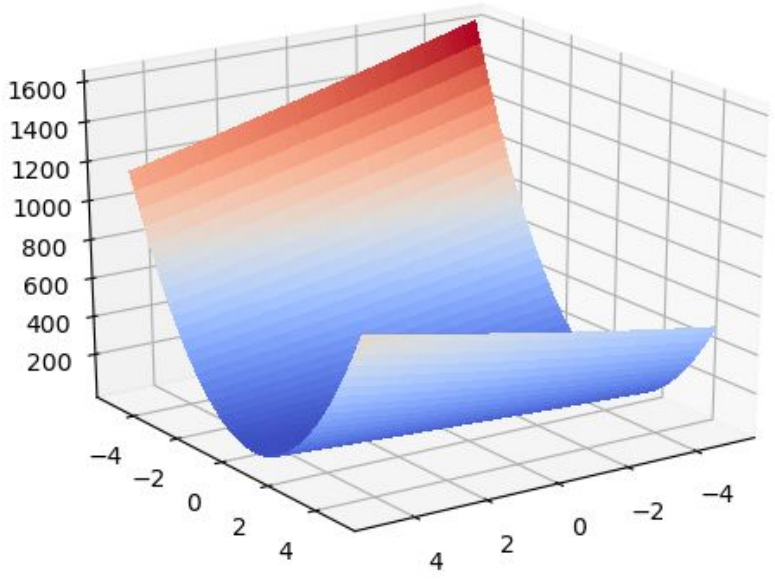
Para generar los valores A, B, Z que plot_surface usa como atributos utilizamos la función make_data que devuelve las tres matrices necesarias para dibujar la gráfica.

```python
def make_data(t0_range,t1_range,X,Y):
    """GeneralasmatricesX,Y,ZparagenerarunplotenD"""
    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)
    #Theta0yTheta1tienenlasmismadimensiones,deformaque
    # #cogiendounelementodecadaunosegeneranlascoordenadasx,y
    # #detodoslospuntosdelarejilla

    Coste=np.empty_like(Theta0)
    for ix, iy in np.ndindex(Theta0.shape):
        Coste[ix,iy]= coste(X,Y,[Theta0[ix,iy],Theta1[ix,iy]])

    return Theta0,Theta1,Coste
```

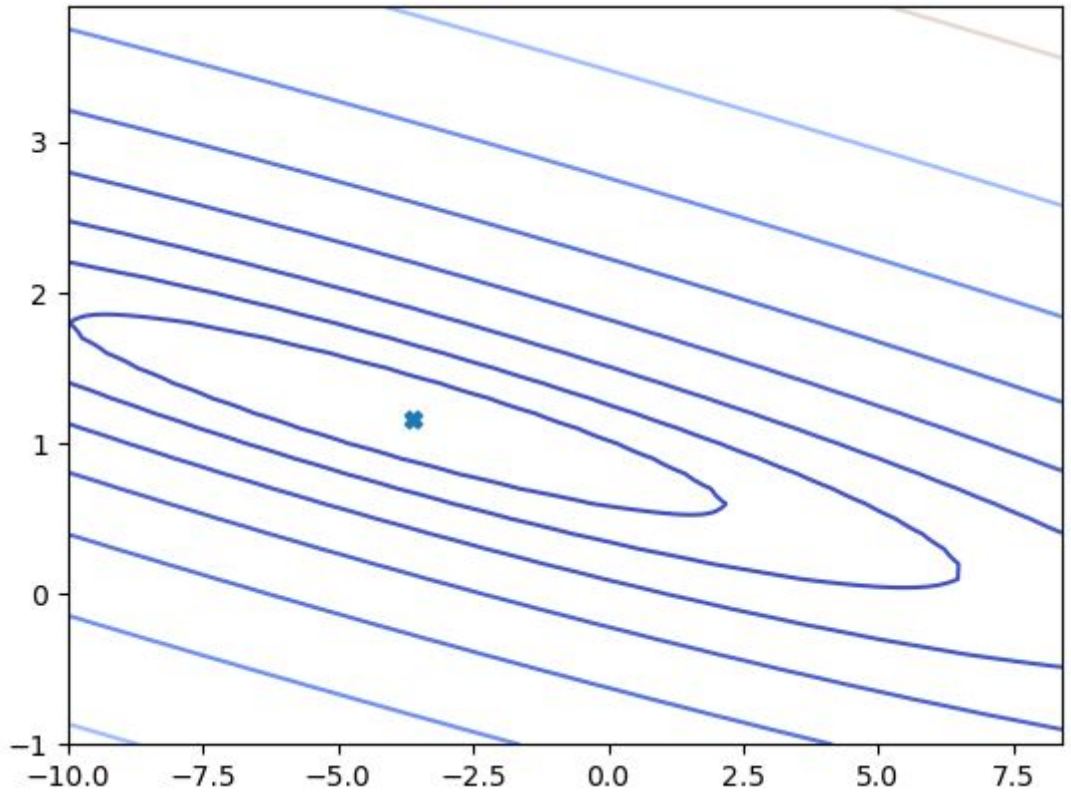
El coste se calcula con la función coste $J(\theta) = 12m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ que pasado a código se ve así.

```python
def h(x, Theta):
    return Theta[0] + Theta [1]*x

#funcion de coste
def coste(X,Y,Theta):
    m = np.shape(X)[0]
    return 1/(2*m) * np.sum((h(X, Theta) - Y)**2)
```

La gráfica resultante se ve así. Donde los ejes horizontales son los valores para theta0 y theta1 y el eje vertical es el coste para cada una de las combinaciones de valores.



A parte de esta gráfica también hemos generado una gráfica del contorno de la gráfica previa utilizando una escala logarítmica para el eje z con 20 intervalos entre 0.01 y 100, lo que se consigue pasando `np.logspace(-2, 3, 20)` como cuarto argumento de la función contour. También mostramos el mínimo obtenido por el descenso de gradiente con una 'x'.



2.Regresión lineal con varias variables

Para implementar la regresión lineal de forma vectorizada con varias variables, lo primero que tenemos que hacer es añadir una columna de unos a la izquierda en la matriz de ejemplos de entrenamiento, para que el valor de la hipótesis se pueda obtener como el producto de los vectores $h_\theta(x) = \theta_0 + \theta_1 x_1 = \theta^T x$

```
datos = carga_csv('ex1data2.csv')
X = datos[:, :-1]
np.shape(X)
Y = datos[:, -1]
np.shape(Y)
m = np.shape(X)[0]
n = np.shape(X)[1]
alpha = 0.01
# añadimos una columna de 1's a la X
X = np.hstack([np.ones([m, 1]), X])
```

Como el rango de los distintos atributos es muy diferente, vamos a normalizarlos sustituyendo cada valor por el cociente entre su diferencia con la media y la desviación estándar de ese atributo en los ejemplos de entrenamiento

```
def normaliza_datos(X):

    media = np.zeros(X[1].size)
    desviacion = np.zeros(X[1].size)

    for x in range(X[1].size):
        xData = X[:,x]
        media[x] = np.median(xData)
        desviacion[x] = np.std(xData)

    xNormalized = (X - media) / desviacion

    return xNormalized
```

Al realizar el descenso de gradiente habiendo normalizado la ecuación obtenemos que los valores de las thetas son: [3404.12659574 1545.53068151 1316.73866325]

2.2 Ecuación normal

Este problema se puede resolver utilizando el método de la ecuación normal que obtiene un sólo paso por el valor óptimo para theta con la expresión $\theta = (X^T X)^{-1} X^T \sim y$.
En este caso no normalizamos los atributos.

```
def ecuacionNormal(X, Y):
    Thetas = np.zeros(X[1].size)

    Xt = X.T
    aux = np.dot(Xt, X)
    aux2 = np.linalg.pinv(aux)
    aux3 = np.dot(aux2, Xt)
    Thetas = np.dot(aux3, Y)

    return Thetas
```

En este caso las thetas nos dan: [89597.90954361 139.21067402 -8738.01911255]