Práctica 1: Regresión Lineal

Ana Martín Sánchez, Nicolás Pastore Burgos

21/09/2021

1 Descripción de la práctica

En esta práctica, se pedía aplicar el método de regresión lineal sobre dos conjuntos de datos.

En primer lugar, se nos da un conjunto de datos que relaciona dos variables: la población de una ciudad con los beneficios de una compañía de distribución de comida en dicha ciudad. Después, se nos presenta un conjunto de datos que relaciona el precio de casas vendidas en Portland con el tamaño en pies cuadrados, el número de habitaciones y el precio de dicha casa.

Para el primer conjunto de datos, se puede aplicar el método de regresión lineal con una variable; es evidente que, en el segundo caso, no es así.

Para agilizar esta segunda parte, es necesario normalizar los atributos, antes de aplicar un descenso de gradiente.

Por último, aplicaremos el método de la ecuación normal sobre el segundo conjunto de datos, con el fin de comprobar que nuestros cálculos anteriores son correctos.

2 Solución propuesta

2.1 Resultados obtenidos

2.1.1 Parte 1

En esta parte hemos conseguido los reultados esperados; hemos podido implementar una función de descenso de gradiente que es capaz, dados unos datos, de calcular una ecuación lineal que minimice los costes. Para comprobarlo, hemos utilizado el ejemplo propuesto en el enunciado (para una población de 70.000 habitantes, los beneficios estimados son de 45.282$).

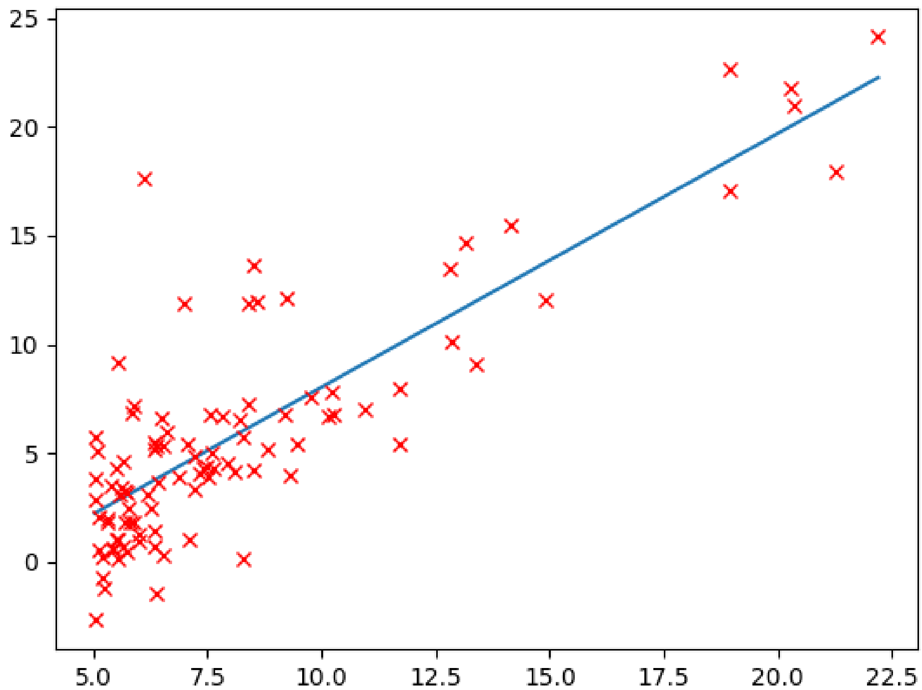

Figure 1: Gráfica que muestra la recta calculada para los datos proporcionados por el enunciado.

Además, hemos generado las gráficas que muestran los costes, como se muestran a continuación.

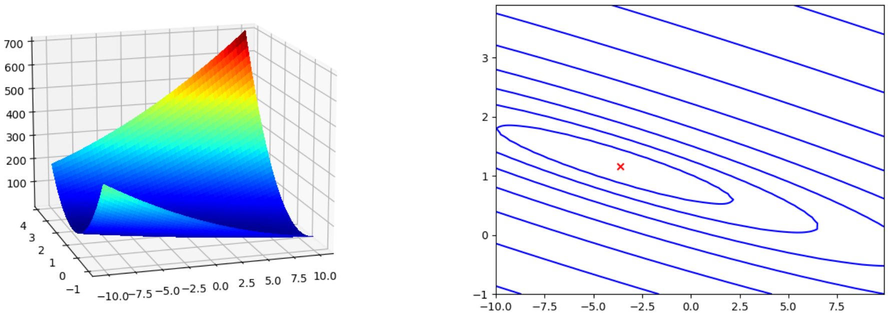

Figure 2: Representación de una sección de la ecuación que siguen las thetas.

2.1.2 Parte 2

En esta segunda parte, también hemos implementado un descenso de gradiente, pero con un número n de atributos. Además hemos implementado la ecuacuón normal.

Para comprobar que ambas implementaciones eran correctas, hemos utilizado el ejemplo propuesto en el enunciado (la entrada [1650, 3]), y ambas nos daban el mismo resultado (293098.46), con una diferencia de menos del 0.01% entre ambos resultados.

Como comprobación adicional, hemos observado que la función de costes devuelve un resultado cada vez menor, como se puede observar a continuación.

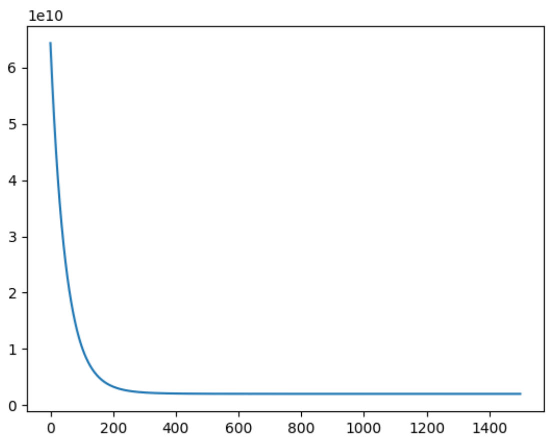

Figure 3: La función de costes es cada vez menor.

2.2 Implementación

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.core.fromnumeric import shape
from numpy.core.numeric import ones_like, zeros_like
from pandas.io.parsers import read_csv
from mpl_toolkits.mplot3d import Axes3D, axes3d
from matplotlib import cm, colors
from matplotlib.ticker import LinearLocator, FormatStrFormatter

def loadCSV(fileName):
    return read_csv(fileName, header=None).to_numpy().astype(float)

def make_data (t0_range, t1_range, x, y):
    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] = coste(Theta0[ix, iy], Theta1[ix, iy], x, y, len(x))

    return [Theta0, Theta1, Coste]

def paint(x, y, t0, t1, t0Mat, t1Mat, costeMat):
    fig = plt.figure()

    # to avoid a warning we do...
    ax = Axes3D(fig, auto_add_to_figure=False)
    fig.add_axes(ax)

    surf = ax.plot_surface(t0Mat,t1Mat,costeMat, cmap=cm.jet, linewidth=0, antialiased=False)
    plt.show()

    plt.scatter(t0, t1, marker='x', color='red')
    plt.contour(t0Mat,t1Mat,costeMat, np.logspace(-2,3,20), colors='blue')
    plt.show()

    minX = min(x)
    maxX = max(x)
    minY = t0 + t1 * minX
    maxY = t0 + t1 * maxX
    plt.plot([minX, maxX], [minY, maxY])
    plt.plot(x, y, "x", color='red')
    plt.show()

    print([t0, t1])

def minimizeCost(x, y):
```

```python

    m = len(x)
    alpha = 0.01

    theta0 = theta1 = 0

    for _ in range(1500):
        """
            Esto visualiza la progresion de la pendiente gradualmente en funcion de los erroes
        """
        # minX = min(x)
        # maxX = max(x)
        # minY = theta0 + theta1 * minX
        # maxY = theta0 + theta1 * maxX
        # plt.plot([minX, maxX], [minY, maxY], "--", linewidth=0.5)

        """
            Ajusta una vez la theta0 y theta1 segun el error calculado
        """
        sum0 = np.sum((x * theta1 + theta0) - y)
        sum1 = np.sum(((x * theta1 + theta0) - y) * x)

        theta0 = theta0 - (alpha/m) * sum0
        theta1 = theta1 - (alpha/m) * sum1

    # 70.000 habs = 4.53...
    # print(theta0 + theta1*7)

    return [theta0, theta1]

def coste (theta0, theta1, x, y, m):
    return np.sum(((theta0 + theta1 * x) - y) ** 2) / (2 * m)

def parte1():
    data = loadCSV("Data/ex1data1.csv")

    x = data[:, 0]
    y = data[:, 1]

    t0, t1 = minimizeCost(x, y)

    t0Mat, t1Mat, costeMat = make_data([-10,10], [-1,4],x, y)
    paint(x, y, t0, t1, t0Mat, t1Mat, costeMat)

"""
    ==================================================================
    ==================================================================
    ==================================================================
    ==================================================================
"""

def normalizaMat(mat):
```

```python
    mu = np.array(np.mean(mat, axis=0))
    sigma = np.array(np.std(mat, axis=0))

    # res = (mat - mu) / sigma
    # np.nan_to_num(res, False, 1.0)

    # [1, 1541.3, 4]
    matNorm = ones_like(mat)
    # [0, 1]
    for i in np.arange(np.shape(mat)[1]-1):
        # [1, 2]
        matNorm[:, i+1] = (mat[:, i+1] - mu[i+1]) / sigma[i+1]

    return matNorm, mu, sigma

def costeVec(x, y, thetas):
    xTh = np.dot(x, thetas)
    return np.sum((xTh - y)**2) / (2*len(x))

def descensoGradiente(x, y, alpha):
    m = np.shape(x)[0]
    n = np.shape(x)[1]

    #thetas = np.zeros(n)
    thetas2 = np.zeros(n)

    costes = np.zeros(1500)

    for i in range(len(costes)):
        xTh = np.dot(x, thetas2)

        NuevaTheta = thetas2
        Aux = xTh - y
        for j in range(n):
            Aux_j = Aux * x[:, j]
            NuevaTheta[j] -= (alpha / m) * Aux_j.sum()
        costes[i] = costeVec(x, y, thetas2)

        thetas2 = NuevaTheta

    # for i in range(len(costes)):
    #     xTh = np.dot(x, thetas)

    #     temp = np.dot(np.transpose(x), (xTh - y))
    #     thetas = thetas - (alpha/m) * temp
    #     costes[i] = costeVec(x, y, thetas)

    plt.plot(np.arange(len(costes)), costes)

    plt.show()

    return thetas2

```

```python
def ecuacionNormal(x,y):
    return np.dot(np.linalg.pinv(np.dot(np.transpose(x), x)), np.dot(np.transpose(x), y))

def parte2():
    data = loadCSV("Data/ex1data2.csv")

    x = data[:, :-1]
    y = data[:, -1]

    m = np.shape(x)[0]
    n = np.shape(x)[1]

    xNew = np.hstack([np.ones([m, 1]), x])

    xNorm, mu, sigma = normalizaMat(xNew)

    alpha = 0.01
    thetasDG = descensoGradiente(xNorm, y, alpha)
    thetasEN = ecuacionNormal(xNew,y)

    example = [1.0, 1650.0, 3.0]
    exampleNorm = ones_like(example)

    # for i in np.arange(len(example)-1):
    #     exampleNorm[i+1] = (example[i+1] - mu[i+1]) / sigma[i+1]

    exampleNorm[1:] = (example[1:] - mu[1:]) / sigma[1:]


if __name__ == "__main__":
    parte1()
    parte2()
```

main.py