MemoriaP1

October 5, 2021

Práctica 1: Regresión lineal

Álvar Domingo Fernández y Pablo Jurado López

Imports iniciales

A continuación se importan todas las librerías que serán utilizadas en esta práctica:

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from pandas.io.parsers import read_csv
from matplotlib import cm
```

1.- Regresión lineal con una variable

A partir de un fichero con dos columnas de datos, hemos aplicado la fórmula del descenso de gradiente para minimizar la función de coste.

La siguiente función sirve para cargar los datos que vamos a utilizar desde un archivo .csv:

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

A continuación se muestra la función que aplica la fórmula del descenso de gradiente a partir de X e Y (las dos columnas de la tabla del archivo .csv que hemos leído). Como ejemplo se ha programado para que haga 1500 iteraciones, en las que se irán actualizando las componentes Θ0 y Θ1 al mismo tiempo. Cuando termine, calculará el coste utilizando la función *coste*. Finalmente, se utilizará la librería *matplotlib* para dibujar la recta obtenida en una gráfica.

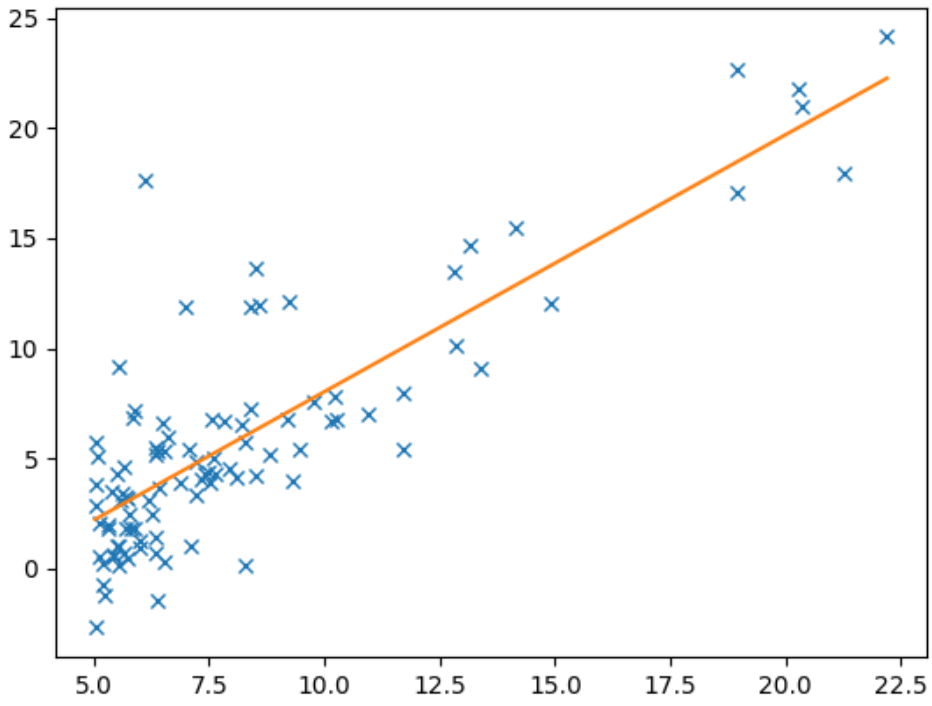
[ ]:
```
def descenso_gradiente(X, Y):
    m = len(X)
    alpha = 0.01
    theta_0 = theta_1 = 0
    for _ in range(1500):
        sum_0 = sum_1 = 0
        for i in range(m):
            sum_0 += (theta_0 + theta_1 * X[i]) - Y[i]
            sum_1 += ((theta_0 + theta_1 * X[i]) - Y[i]) * X[i]
```

```
        theta_0 = theta_0 - (alpha/m) * sum_0
        theta_1 = theta_1 - (alpha/m) * sum_1
    min_x = min(X)
    max_x = max(X)
    min_y = theta_0 + theta_1 * min_x
    max_y = theta_0 + theta_1 * max_x

    Coste = coste(X, Y, (theta_0, theta_1))

    # Dibujamos el resultado
    plt.plot(X, Y, "x")
    plt.plot([min_x, max_x], [min_y, max_y])
    plt.savefig("descenso_gradiente.pdf")

    return (theta_0, theta_1), Coste
```

A continuación se muestra el código de la función *coste*:

[3]:
```
def coste(X, Y, Theta):
    m = len(X)
    sumatorio = 0
    for i in range(m):
        sumatorio += ((Theta[0] + Theta[1] * X[i]) - Y[i]) ** 2
    return sumatorio / (2 * len(X))
```

Y finalmente, la gráfica obtenida en la función del descenso de gradiente:



1.1- Visualización de la función de coste

Para visualizar la función de coste, hemos generado dos gráficas: una de superficie y otra de contorno.

El primer paso ha sido procesar los datos iniciales dentro del rango de la gráfica que queremos hacer, y calcular el coste para cada punto de la gráfica.

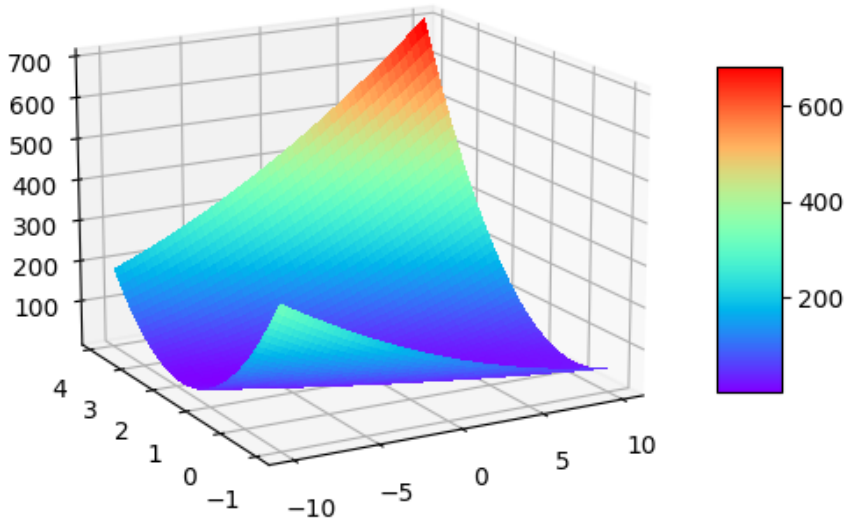
[ ]:
```
def make_data(t0_range, t1_range, X, Y):
    step = 0.1
    Theta0 = np.arange(t0_range[0], t0_range[1], step)
    Theta1 = np.arange(t1_range[0], t1_range[1], step)
    Theta0, Theta1 = np.meshgrid(Theta0, Theta1)
    Coste = np.empty_like(Theta0)
    for ix, iy in np.ndindex(Theta0.shape):
        Coste[ix, iy] = coste(X, Y, [Theta0[ix, iy], Theta1[ix, iy]])
    return [Theta0, Theta1, Coste]
```

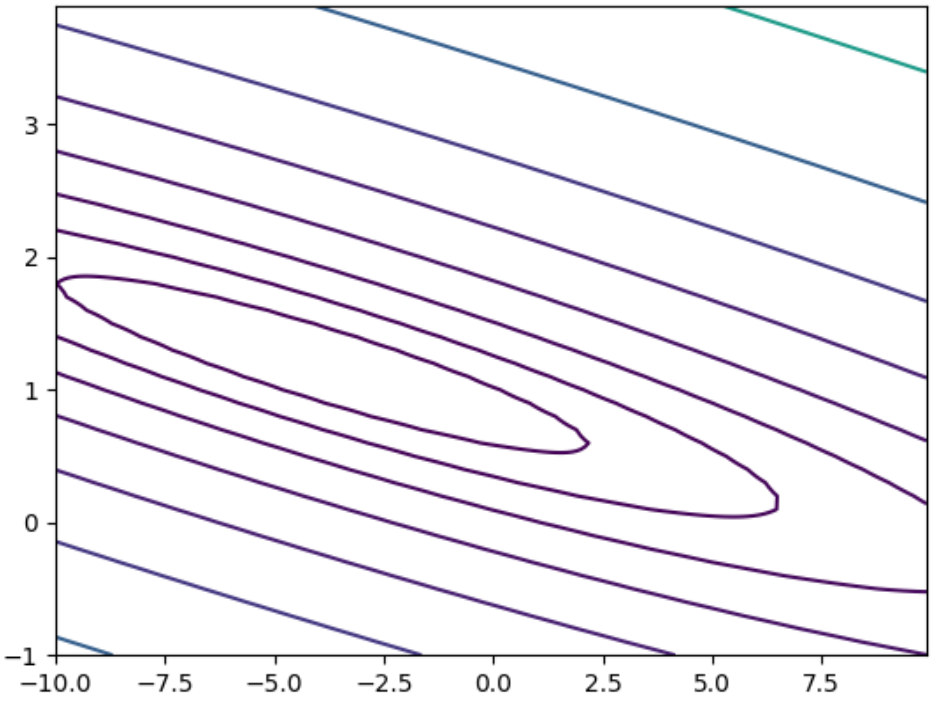
Con la función *dibuja_coste* dibujamos ambas gráficas haciendo uso de la librería *matplotlib*

[ ]:
```
def dibuja_coste(Theta0, Theta1, Coste):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(Theta0, Theta1, Coste,
```

```
                            cmap=cm.rainbow, linewidth=0, antialiased=False)
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()

    fig2 = plt.figure()
    plt.contour(Theta0, Theta1, Coste, np.logspace(-2, 3, 20))
    plt.show()
```

A continuación se muestran ambas gráficas generadas por la anterior función:





2.- Regresión lineal con varias variables

A partir de un fichero con tres columnas de datos, hemos aplicado la fórmula del descenso de gradiente para minimizar la función de coste. Como los rangos de las distintas variables son muy diferentes, lo primero que hemos hecho ha sido normalizar las variables mediante la siguiente función:

[ ]:
```
def normalizar(X):
    mu = X.mean(axis=0)
    sigma = X.std(axis=0)
    X_norm = (X - mu) / sigma

    return X_norm, mu, sigma
```

La función *normalizar* devuelve la matriz que se le haya introducido con sus valores normalizados y la media y la desviación estándar de los datos.

2.1.- Implementación vectorizada del descenso de gradiente

A continuación se ha calculado el descenso de gradiente con una variante vectorizada de la función que se utilizó en el apartado 1, con el propósito de evitar iterar por cada elemento de los datos proporcionados, como haciamos en la variante iterativa:

[ ]:
```
def gradiente_vec(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
```

La función *gradiente_vec* ajusta el valor de Theta en función de los datos proporcionados y el alpha elegido.

Por su parte, la función *descenso_gradiente_vec* realiza el cálculo de *gradiente_vec* un número determinado de iteraciones, a la vez que calcula los costes.

[ ]:
```
def descenso_gradiente_vec(X, Y, alpha):
    Theta = np.zeros(np.shape(X)[1])
    iteraciones = 500
    costes = np.zeros(iteraciones)
    for i in range(iteraciones):
        costes[i] = coste_vectorizado(X, Y, Theta)
        Theta = gradiente_vec(X, Y, Theta, alpha)

    return Theta, costes
```

Para el cálculo de los costes también se ha implementado una función de coste vectorizada:

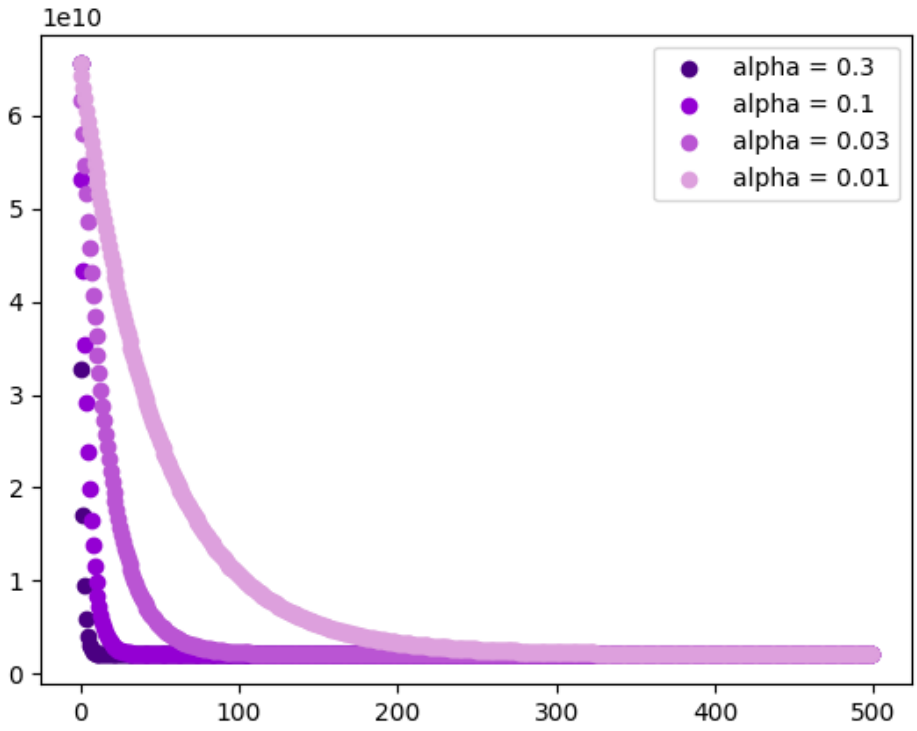
[ ]:
```
def coste_vectorizado(X, Y, Theta):
    H = np.dot(X, Theta)
    Aux = (H - Y) ** 2
    return Aux.sum() / (2*len(X))
```

Con los costes que se han calculado, se ha dibujado una gráfica con *matplotlib* que muestra la evolución de los costes según el valor de alfa con el que se calcule la función. En *apartado_2_1* se puede ver cómo se hace el dibujo de la gráfica, además de las preparaciones iniciales para el descenso de gradiente vectorizado (se normalizan los datos de la matriz X y se le añade una columna llena de unos para que se pueda calcular Theta como producto de matrices)

[ ]:
```
def apartado_2_1():
    data = carga_csv("ex1data2.csv")
    X = data[:, :-1]
    Y = data[:, -1]
    m = np.shape(X)[0]

    X, mu, sigma = normalizar(X)
    X = np.hstack([np.ones([m, 1]), X])
```

```
alphas = [0.3, 0.1, 0.03, 0.01]
colors = ['indigo', 'darkviolet', 'mediumorchid', 'plum']

plt.figure()

for i in range(len(alphas)):
    Theta, costes = descenso_gradiente_vec(X, Y, alphas[i])
    plt.scatter(np.arange(np.shape(costes)[
                0]), costes, c=colors[i], label='alpha = ' + str(alphas[i]))

plt.legend()
plt.savefig("pjbobo.png")
```

A continuación se muestra la gráfica obtenida por esta función:



2.2.- Ecuación normal

Finalmente, calcularemos el valor óptimo de Theta mediante el uso de la ecuación normal, que nos evita tener que hacer bucles como en el método del descenso de gradiente. En este caso no hace falta normalizar los atributos. Para hacer el cálculo simplemente aplicamos la fórmula correspondiente en una línea de código:

[ ]:
```
def ecuacion_normal(X, Y):
    Theta = np.matmul(np.matmul(np.linalg.inv(np.matmul(np.transpose(X), X)),␣
 ↪np.transpose(X)), Y)
    return Theta
```

Una vez calculada, procederemos a comprobar si las predicciones hechas con ambas fórmulas son similares. Para ello calclularemos los valores de Theta y a partir de ellos calcularemos la predicción correspondiente con cada método, por ejemplo, para una casa con una superficie de 1650 metros cuadrados y 3 habitaciones:

[ ]:
```
def apartado_2_2():
    data = carga_csv('ex1data2.csv')
    X = data[:, :-1]
    Y = data[:, -1]
    m = np.shape(X)[0]

    X_norm, mu, sigma = normalizar(X)
    X_norm = np.hstack([np.ones([m, 1]), X_norm])

    theta_vec, costecitos = descenso_gradiente_vec(X_norm, Y, 0.01)

    X = np.hstack([np.ones([m, 1]), X])
    theta_normal = ecuacion_normal(X, Y)

    pred_normal = theta_normal[0] + \
        theta_normal[1] * 1650 + theta_normal[2] * 3
    pred_gradient = theta_vec[0] + theta_vec[1] * \
        ((1650 - mu[0]) / sigma[0]) + theta_vec[2] * ((3 - (mu[1]) / sigma[1]))

    print('Theta de ecuación normal: ', pred_normal, '\n')
    print('Theta de gradiente vectorizado: ', pred_gradient, '\n')
```

Ahora solo queda observar los prints que hace el programa:

Theta de ecuación normal: 293081.4643348959

Theta de gradiente vectorizado: 299500.8939033111

Como se puede observar, los resultados son bastante similares, por lo que se ha podido llegar a la conclusión de que los cálculos realizados, tanto mediante el método de descenso de gradiente como el de la ecuación normal, son correctos.