Universidad Complutense de Madrid
Facultad de Informática
Aprendizaje Automático y Big Data

Memoria Práctica 1.

Profesor:
- Alberto Díaz Esteban.

Alumnos:
- Marina de la Cruz López.
- Diego Alejandro Rodríguez Pereira.

Introducción

En esta práctica vamos a implementar tres versiones en las que se utilizará el método de regresión lineal. La primera versión será regresión lineal para una variable, utilizando descenso de gradiente, y las dos versiones siguientes serán regresiones lineales para múltiples variables, la primera versión utilizando descenso de gradiente, y la segunda versión utilizando el método de ecuación normal.

Para la regresión lineal para una variable comprobaremos los resultados obtenidos usando gráficas que nos ayuden a entender el progreso y resultado obtenidos.

Para la regresión lineal para múltiples variables, se utilizará como comprobación, la comparación de los resultados obtenidos entre la versión de descenso de gradiente y la versión de ecuación normal.

Primera Parte: Regresión Lineal con una Variable

Para esta primera parte, se ha realizado una implementación de regresión lineal para una única variable. El problema a resolver consiste en predecir los beneficios de una compañía de distribución de comida.

Los datos en los que se ha trabajado contienen 2 columnas y 96 filas. Las dos columnas consisten en: la primera contiene el número de habitantes de una población de una ciudad en base a los 10.000 habitantes, y la segunda columna, los beneficios obtenidos. En este caso nuestra variable "target" será la segunda columna, qué son los valores a predecir.

Para comenzar se han cargado los datos del fichero "ex1data1.csv" proporcionado por el profesor. Luego estos datos se han dividido en dos *arrays*, uno para la Y y el otro para la X, siendo la X la primera columna, es decir, el número de habitantes, y la Y la segunda columna, es decir, el beneficio obtenido.

Luego se ha recogido el número de ejemplos de entrenamiento, que en este caso han sido 96. Después se ha definido la variable *alpha* a 0.01. Posteriormente se ha inicializado las Thetas (θ), tanto θ_0 como θ_1 al valor 0, como en este caso solo tenemos 2 hacemos las asignaciones a mano para Theta0 ($\theta 0$) y Theta1 ($\theta 1$) en cada iteración del bucle, para que de esta manera se evite sobrescribir los anteriores valores de la asignación, por lo que se ha computado primero h(x) y posteriormente se ha usado el resultado para calcular los ambos valores.

Hemos repetido este bucle antes mencionado unas 1500 iteraciones, y para cada iteración hemos guardado el valor de la función de coste en una lista para luego poder comprobar que el coste efectivamente decrece en cada iteración y así demostrar que nuestra función funciona correctamente.

Gráficas Parte 1

En este apartado mostraremos algunas de las gráficas obtenidas en esta implementación. Posteriormente discutiremos los resultados obtenidos.

La primera gráfica obtenida se muestra a continuación en la Figura1, en la que se puede observar la función de la recta obtenida al aplicar el descenso de gradiente. Los puntos marcados con una 'x' azul son cada uno de los datos de entrenamiento (cada fila de la tabla), y la línea roja es la recta obtenida. Esta recta, como se puede comprobar en la gráfica, se ajusta relativamente bien a los datos, por lo que refleja adecuadamente la tendencia de los datos.

De esta manera se puede predecir con cierto margen de error los beneficios obtenidos por la empresa a partir de una población dada.

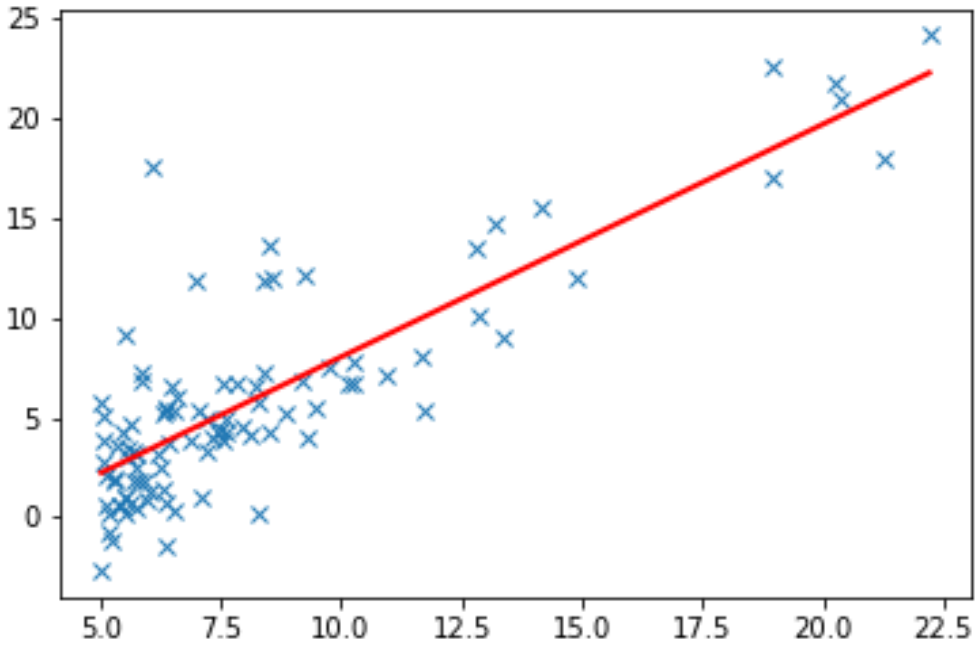

Figura 1

Visualización de la función de coste en cada iteración

A continuación, mostramos el valor de la función de coste en cada una de las iteraciones realizadas. En la figura2 se puede observar la gráfica obtenida de la función de coste. Se puede comprobar en esta como el valor de la función decrece en cada una de las iteraciones del bucle. Con lo que se comprueba lo visto en clase, en donde en las primeras iteraciones la función de coste es grande y va decreciendo en gran medida. Pero una vez que llega a cierto umbral, en este caso alrededor de las 1.000 iteraciones, el coste disminuye muy poco, acercándose así el mínimo óptimo.

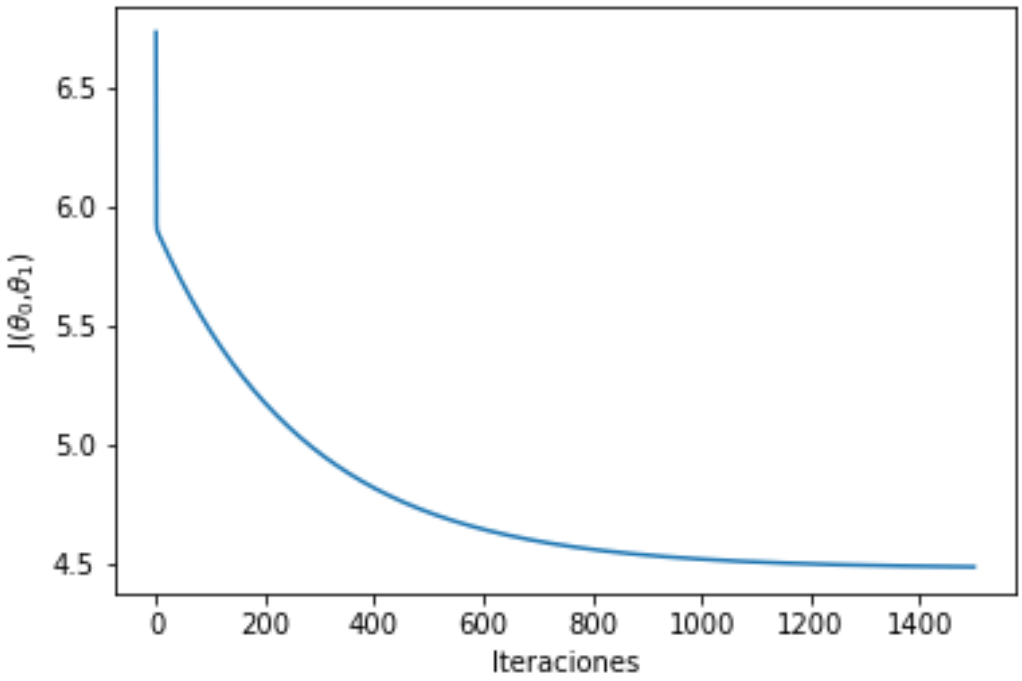

Figura 2

Gráfica de contorno de la función de coste

En esta figura mostramos la gráfica de contorno obtenida mediante la función "*contour*" de la librería "*matplotlib*". Para esta gráfica se ha definido los intervalos [-10, 10], [-1, 4] y posteriormente mostramos el punto 'x' en rojo, como el último valor obtenido de Theta0 y de Theta1 respecto del descenso de gradiente. Por lo que se puede concluir de la gráfica, así como hemos visto en clase, que entre más cercano esté este valor al centro, entonces mejor será el resultado obtenido. Para el caso realizado, el punto 'x' pintado en la gráfica está cerca del centro, aunque se desvía ligeramente hacía la derecha.

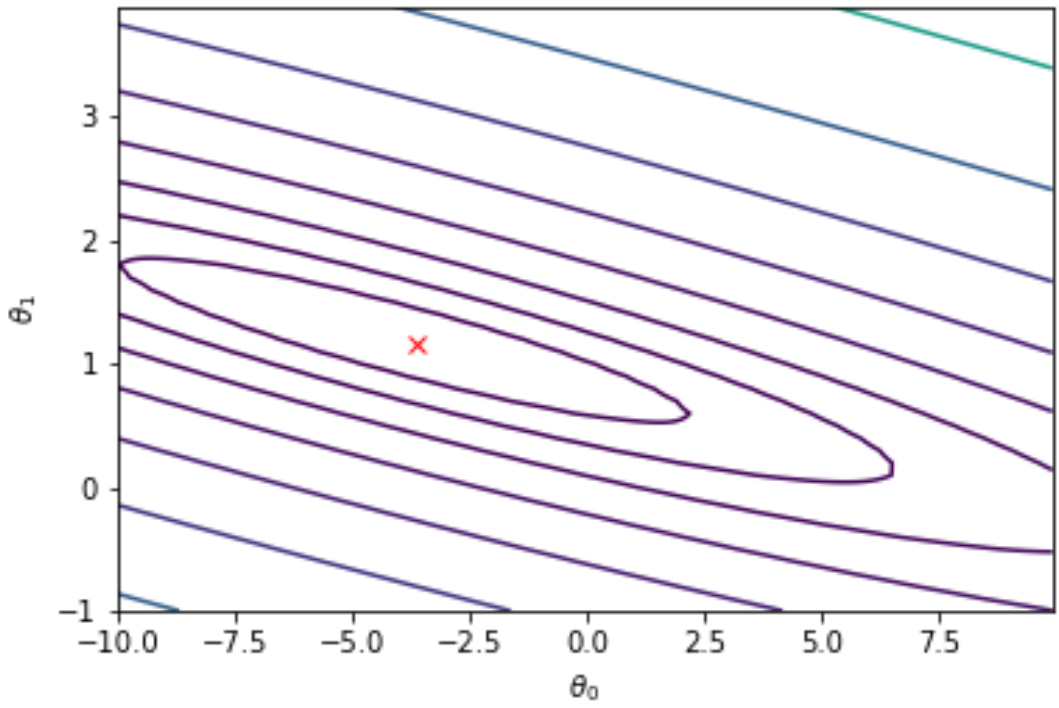

Figura 3

Por último, tenemos la Figura4, en la que se muestra el resultado obtenido al aplicar la función *plot_surface* de la librería matplotlib de Python. Esta imprime una proyección en 3D de la función de coste. Los intervalos que se han considerado para la gráfica son los mismos que en la figura anterior (Figura3), es decir, [-10, 10], [-1, 4].

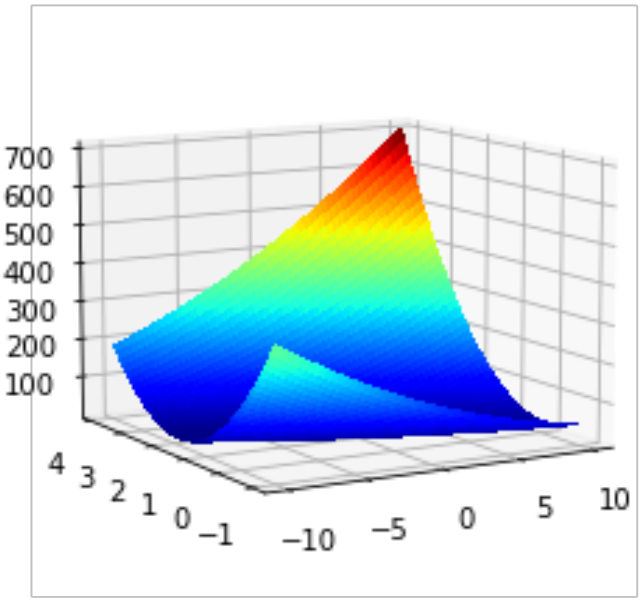

Figura 4

Resultados Obtenidos

La función de coste obtenida en este apartado se ha comprobado que decrece sucesivamente en cada iteración del bucle. Partiendo su coste desde los 6,5 (J(θ0, θ1) == 6,5), hasta llegar a al coste de 4,5. Lo cual consideramos que es una bajada significativa y que permite obtener con ello valores muy cercanos a los reales. A pesar de esto, consideramos que podría realizarse más iteraciones para obtener resultados aún más precisos, pero esto ocasiona dos desventajas, la primera es que aumentaría el tiempo de ejecución del algoritmo, en especial si este se utiliza con una gran cantidad de datos (mayor a la utilizada en este apartado), y que a su vez la función de la recta estaría sobreajustandose a los datos.

Los valores para theta0 y theta1 obtenidos para ‘ex1data1.csv’ son:

theta0: -3.63029, theta1: 1.166362

Podemos comprobar que los valores obtenidos son correctos con el ejemplo del ejercicio, intentado predecir los beneficios aproximados para una población de 70.000, x= 7, obtenemos y=4.5342, es decir 45.342$.

Segunda Parte: Regresión Lineal para varias Variables

Para esta segunda parte se ha realizado el problema de Regresión Lineal usando varias variables de dos formas distintas. La primera implementación se ha realizado programando el descenso de gradiente con múltiples parámetros, y la segunda implementación se ha realizado utilizando la ecuación normal. Explicaremos el procedimiento utilizado en ambas implementaciones y posteriormente se mostrarán los resultados que se han obtenido para luego discutir la similitud entre ambas implementaciones respecto a los valores obtenidos.

Para esta segunda parte se han utilizado un problema y datos distintos al apartado anterior. En este caso se aplicará el método de regresión lineal sobre unos datos que contienen información sobre los precios de casas vendidas en la ciudad de Portland, Oregon. De esta manera predecir futuros precios. La tabla está compuesta por 3 columnas, siendo la primera el tamaño de la casa en pies cuadrados, la segunda columna el número de habitaciones que esta tiene, y por último la variable a predecir, el precio. Este ejemplo contiene 46 filas, es decir, 46 distintas casas.

Descenso de Gradiente (primera implementación)

La primera implementación del método de regresión lineal para varias variables es el descenso de gradiente.

En primer lugar, se cargan los datos del fichero *'ex1data2.csv'* utilizando la librería de Pandas, las cuales luego se dividen en dos partes. La primera, una matriz X que contiene los datos de las primeras dos columnas, y la segunda un *array* Y que contiene la variable '*target'*.

Luego de cargar los datos, se normalizan estos, ya que para este problema existe una gran diferencia entre variables de la matriz. Por lo tanto, hemos normalizado los datos de la matriz X para que todos se encuentren en la misma escala y que el descenso de gradiente evolucione de forma similar para todas las variables. Ya que el tamaño de la casa al ser en pies cuadrados es mucho mayor en unidades que el número de habitaciones que contiene la casa.

Después hemos añadido una columna adicional a la matriz X, rellenada con unos (1s), para que de esta manera se pudiesen vectorizar los cálculos a realizar. Posteriormente se obtienen en número de filas y de columnas, se inicializan las thetas a cero y se define **α** (*alpha*) al valor 0,005.

Como estamos trabajando con matrices para obtener los valores de h(X) (hipótesis) hemos creado una función *h_vec* que devuelve el np.dot de la matriz X y la lista de Thetas que queremos obtener del descenso de gradiente. El np.dot nos va a devolver un *array* que contiene la suma de la multiplicación de cada columna de la matriz por cada theta (que corresponde al producto de matrices) que viene a ser la fórmula de la expresión que tratamos de ajustar para cada fila de X (es decir, cada dato).

Una vez obtenido el vector con los resultados de h(X) para los valores de theta de una iteración tenemos que aplicar la función de descenso de gradiente con este valor y generar los nuevos Thetas, para ello simplemente restamos el vector Y al vector de observaciones de h(X), volvemos a hacer np.dot del resultado y la matriz X para hacer la suma de la multiplicación de la diferencia y cada elemento de cada columna de X, que nos devuelve otro *array* de tamaño 3 (cantidad de columnas de la matriz X) el cual solo nos queda multiplicar por (alpha/m) y restarlo a las thetas para generar los nuevos valores de esa iteración.

El proceso se realiza en tantas iteraciones como consideremos hasta que converja, nosotros hemos probado con 1500, 3000 y 4000.

Gráficas Parte 2

La primera gráfica obtenida es la de la función de coste. Como se puede observar en la Figura5, la función de coste disminuye considerablemente en las primeras iteraciones, En la Figura5 podemos observar el descenso de la función de coste para 1500 iteraciones con alpha=0.01, que efectivamente tiene la forma que esperamos

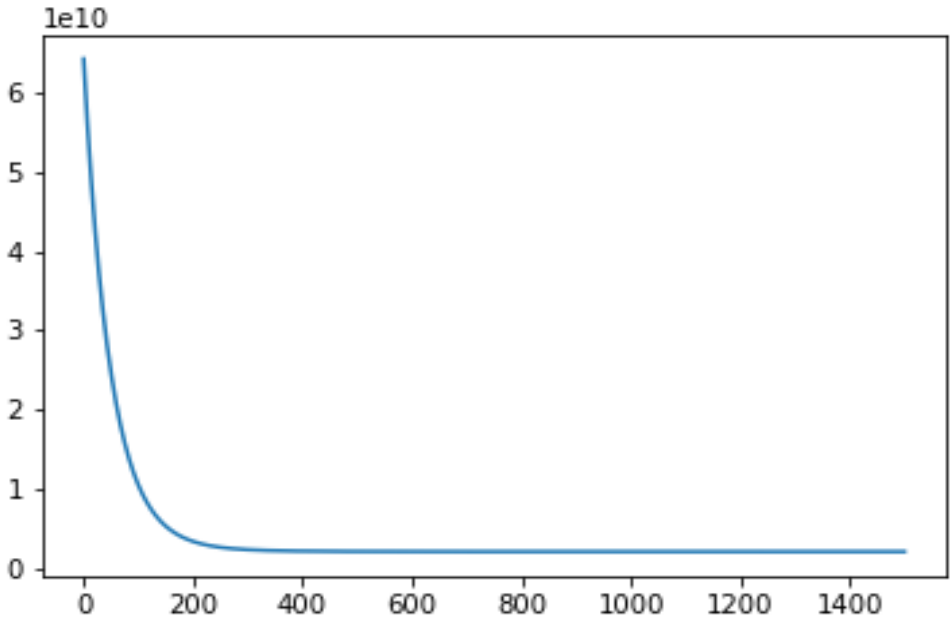

Figura 5

Ecuación Normal

Hemos programado la versión de la regresión lineal con múltiples variables usando la ecuación normal, en este caso cuando leemos los datos no los tenemos que normalizar. Como en el caso anterior separamos la columna Y en un *array* y el resto de la matriz con los valores de las X en una matriz, a la que añadimos la columna de unos en la primera posición.

El código para esta versión es muy sencillo, obtenemos la transpuesta usando *np.transpose* y la multiplicamos con np.dot con la matriz normal X, hacemos la inversa del resultado usando *np.linalg.pinv* y finalmente multiplicamos la inversa por la transpuesta y el resultado por el *array* de salida Y, esto nos devuelve el *array* de Thetas (de tamaño 3).

Resultados Obtenidos

Para comprobar que los dos métodos se han realizado correctamente hemos probado a ejecutar una predicción con las Thetas obtenidas, en el caso del descenso de gradiente hemos tenido que normalizar la entrada para que la predicción sea correcta para las Thetas obtenidas.

Predicción:
"Una casa con una superficie de 1.650 pies cuadrados y 3 habitaciones predecir el precio"

Descenso de gradiente:

1500 iteraciones, alpha 0.01
Thetas= [340412.56301439 109370.05670466 -6500.61509507]
Y (Precio)=293.098,466

4000 iteraciones, alpha 0.005
Thetas=[340412.65890716 109439.14203274 -6569.70041726]
Y (Precio)=293.083,367

5000 iteraciones, alpha 0.005
Thetas= [340412.65957003 109446.84000912 -6577.39839364]
Y (Precio)=293.081,674

Ecuación normal:

Y (Precio)=293.081,464

Podemos observar que existe cierta diferencia entre las predicciones realizadas con ambos métodos, aunque la diferencia se hace menor cuando consideramos más iteraciones del bucle para el descenso de gradiente y una *alpha* más pequeña. Con esto podemos decir que ambos métodos devuelven resultados equivalentes.

Código

Parte 1

```
#Imports
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

```
#Función que carga los datos
def load_csv(filename):
   valores = read_csv (filename, header=None).to_numpy()
   return valores.astype(float)
```

```
#Hipótesis
def h(x, theta):
   return theta[0] + theta[1] * x
```

```
#Función de Coste
def cost_function(X, Y, theta):
   observed = h(X, theta)
   return func_coste(Y, observed)
```

```
#Función Auxiliar de coste
def func_coste(expected, observed):
   #Utilizamos operaciones vectoriales para reducir el tiempo de
ejecución
   dif = (1/(2*len(expected)))*np.sum(np.subtract(expected,
observed)**2)
   return dif
```

```
#Función Principal
def lineal_regresion_one_variable():
   data = load_csv('ex1data1.csv')
```

```
    #Primera Columna
    X = data[:,0]
    #Segunda Columna
    Y = data[:,1]

    # Numero de ejemplos de entrenamiento
    m = len(X)
    #Tamanho inicial de los saltos
    alpha = 0.01
    #Inicialización de los valores de theta0 y theta1
    theta0 = theta1 = 0

    coste = list()
    for i in range(1500):
        #Hipótesis
        observed = h(X, [theta0, theta1])
        #Calculo de las Thetas
        theta0 = theta0 - (alpha/m) * np.sum(observed - Y)
        theta1 = theta1 - (alpha/m) * np.sum((observed - Y) * X)
        #Guardamos el coste obtenido por la función en una lista
        coste.append(cost_function(X, Y, [theta0, theta1]))

    #Imprimimos gráfica con la función de la recta obtenida
    plt.plot(X, Y, "x")
    min_x = min(X)
    max_x = max(X)
    min_y = theta0 + theta1 * min_x
    max_y = theta0 + theta1 * max_x
    plt.plot([min_x, max_x], [min_y, max_y], color='red',
linewidth=2)

    plt.savefig("part1_linear_regresion.png")
    plt.show

    return [theta0, theta1, coste]
```

```
#Llamada a la función principal
l = lineal_regresion_one_variable()
```

```
def make_data(t0_range, t1_range, X, Y, step=0.1):
    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)
     # Theta0 y Theta1 tienen las misma dimensiones, de forma que
     # cogiendo un elemento de cada uno se generan las coordenadas
x,y
     # de todos los puntos de la rejilla
    Coste = np.empty_like(Theta0)
    for ix, iy in np.ndindex(Theta0.shape):
        Coste[ix, iy] = cost_function(X, Y, [Theta0[ix, iy],
Theta1[ix, iy]])

    return [Theta0, Theta1, Coste]
```

```
def show_contour(data, thetas):
    plt.contour(data[0],data[1],data[2],np.logspace(-2,3,20))
    plt.plot(thetas[0], thetas[1],'rx');
    plt.xlabel('$\\theta_{0}$'); plt.ylabel("$\\theta_{1}$")
    plt.savefig("part1_contour.png")
```

```
def show_mesh(data):
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(data[0], data[1], data[2], cmap=cm.jet,
linewidth=0, antialiased=False)

    plt.show()
    fig.savefig("part1_mesh.png")
```

```
def print_graphs():
    data = load_csv('ex1data1.csv')
    X = data[:, 0]
    Y = data[:, 1]
    grid_data = make_data([-10, 10], [-1, 4], X, Y)
```

```
    dataPoint = lineal_regresion_one_variable()
    show_contour(grid_data, dataPoint)
    show_mesh(grid_data)
```

Parte 2.1

```
#Imports
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

```
#Función que carga los datos
def load_csv(filename):
    valores = read_csv (filename, header=None).to_numpy()
    return valores.astype(float)
```

```
#Función de la Hipótesis
def h_vec(x, theta):
    return np.dot(x, theta)
```

```
# Función de coste
def coste(X, Y, Theta):
    H = np.dot(X, Theta)
    return (1/(2 * len(X)))*np.dot(np.transpose(H-Y),(H-Y))
```

```
#Función que normaliza unos datos de entrada
#Devuelve los datos normalizados junto con la media y la desviación
estándar
def norm(X):
    X_norm = np.copy(X)
    means = []
    std = []
    for i in range(n):
        means.append(np.mean(X[:,i]))
        std.append(np.std(X[:,i]))
```

```
        X_norm[:,i] = (X[:,i]-np.mean(X[:,i]))/np.std(X[:,i])

    return X_norm, means, std
```

```
#Función de descenso de gradiente
def descenso_gradiente(X, Y, alpha):

    #Obtención de los valores m y n
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    #Inicialización de las thetas a 0
    thetas = np.zeros(n)

    #Realización de 1500 iteraciones
    costes = list()
    for i in range(1500):

        #Vector de diferencias entre la función con la Thetas de esta
iteración y los valores reales Y
        Aux = (h_vec(X, thetas) - Y)

        sum_x = np.dot(Aux, X)
        thetas -= (alpha / m) * sum_x

        #Guardamos el valor de la función de coste de cada iteración
        costes.append(coste(X, Y, thetas))

    plt.plot(costes)
    plt.xlabel('Iteraciones');
plt.ylabel("J($\\theta_{0}$,$\\theta_{1}$)")
    plt.savefig("part2_costes.png")
    plt.show

    return thetas, costes
```

```
def linear_regresion_multiple_variables():

    #Carga de los datos
```

```
    datos = load_csv('ex1data2.csv')

    #Obtención de todas las columnas de la tabla menos la última
columna
    X = datos[:, :-1]
    np.shape(X)          # (97, 1)
    #Obtención de la última columna de la tabla
    Y = datos[:, -1]
    np.shape(Y)          # (97,)

    #Obtención de los valores m y n
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    #Normalizamos los datos de entrada
    Means = list()
    Std = list()
    for i in range(n):
        Means.append(np.mean(X[:,i]))
        Std.append(np.std(X[:,i]))
        X[:,i] = (X[:,i]-np.mean(X[:,i]))/np.std(X[:,i])


    # añadimos una columna de 1's a la X
    X = np.hstack([np.ones([m, 1]), X])

    #Llamada a la función de descenso de gradiente
    alpha = 0.005
    Thetas, costes = descenso_gradiente(X, Y, alpha)
    print(Thetas)

    #ejemplo
    ejemplo = h_vec([1,(1650-Means[0])/Std[0] ,(3-Means[1])/Std[1]],
Thetas)

    return [Thetas[0], Thetas[1], coste], ejemplo
```

```
linear_regresion_multiple_variables()
```

Parte 2.2

```
#Importación de las librerías
import numpy as np
from numpy.core.fromnumeric import transpose
from numpy.core.numeric import empty_like
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
```

```
#Función auxiliar para cargar los datos del fichero que pasamos como
parámetro
def load_csv(filename):
    valores = read_csv(filename, header=None).to_numpy()
    return valores.astype(float)
```

```
def func_transpose(X, Y):
    #Realizamos la traspuesta Xt
    transpuesta_x = np.transpose(X)
    #Psudo inversa de una matriz (XtX)^-1
    inverse_matrix = np.linalg.pinv(np.dot(transpuesta_x, X))
    #(XtX)^-1 Xt
    second_mul = np.dot(inverse_matrix, transpuesta_x)
    #(XtX)^-1 Xt Y
    y_mul = np.dot(second_mul, Y)

    return y_mul
```

```
#Función Principal
def linear_regresion_2_2(alpha = 0.01):
    data = load_csv('ex1data2.csv')


    X = data[:, :-1]
    Y = data[:, -1]

    m = np.shape(X)[0]
```

```
    n = np.shape(X)[1]

    #Añadimos una columna de 1's
    X = np.hstack([np.ones([m, 1]), X])

    # Aplicamos la función de la ecuación normal
    Theta = func_transpose(X,Y)

    return Theta
```

```
#Llamada a la función principal
linear_regresion_2_2()
```

,