Universidad Complutense de Madrid

IAAC - PRÁCTICA 1

Yaco Alejandro Santiago Pérez

Asignatura: INTELIGENCIA ARTIFICIAL APLICADA A INTERNET DE LAS COSAS

P1: Regresión lineal

Master IOT

24 de febrero de 2020

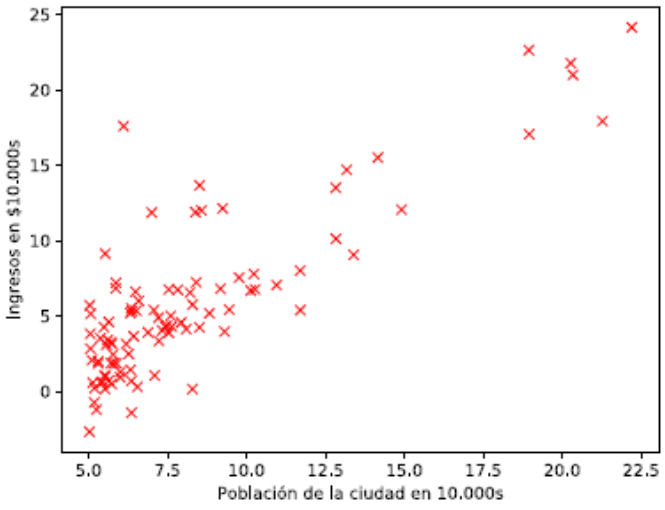
Índice general

Capítulo 1

Introducción

Esta práctica consiste en la utilización de la ***Regresión lineal*** sobre los datos que representan datos sobre los beneficios de una compañía de distribución de comida en distintas ciudades, en base a su población.

Para ello, vamos a utilizar el **método de descenso de gradiente** y encontrar los parámetros de *theta* que definen la recta que se ajusta a los datos de entrenamiento.



Capítulo 2

Objetivos

A lo largo de esta práctica, la cual se divide en dos partes, y se pretende alcanzar, a grandes rasgos, los siguientes objetivos:

- Calcular la regresión lineal con una variable
- Calcular la regresión lineal con varias variables

Para ello, deberemos alcanzar los siguientes objetivos más concretos:

- Calcular el coste
- Calcular los valores de la *hipótesis*
- Calcular los valores de *theta* para definir la recta de regresión
- Generar la *gráfica de contorno*

Capítulo 3

Parte 1: Regresión lineal con una variable

3.1. Funciones

Las funciones empleadas, en orden de llamada, son las siguientes:

- **pinta(puntosX, puntosY):** Es la función que pinta todos los puntos en la gráfica. En mi caso, en ella he definido una parábola.
- **coste(X, Y, Theta):** Función que calcula el coste para una Theta concreta. Implementa la siguiente fórmula:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- **gradiente(X, Y, Theta, alpha):** Función que calcula el gradiente mediante la siguiente fórmula:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- **descenso_gradiente(X, Y, theta, alpha):** Hace las llamadas a los dos métodos previos con el fin de calcular la **Theta** y el **Coste** de esta.
- **make_data(t0_range, t1_range, X, Y,t1,t2):**Función que genera las matrices X,Y,Z para generar un plot en 3D.
- **main():**Función que hace todas las llamadas pertinentes. *Carga los datos*, realiza *1500* iteraciones para obtener la **theta** óptima y su **coste**, *pintar la* ***recta*** *definida por las thetas y lo exporta la gráfica (3.1)*, y por último *genera las gráficas* de **Contorno** (3.2) y **Surface** (3.3) en *3D*.

3.2. Ejecución

En esta sección voy a presentar una ejecución del código.

En la consola se va a ver como se imprimen los valores obtenidos en las **1500** iteraciones. De esta manera:

```
Coste: 4.483458098818883 - theta: [-3.62885054  1.1662176 ]
[-3.62885054  1.1662176 ]
1497
Coste: 4.483434734017543 - theta: [-3.6293317   1.16626593]
[-3.6293317   1.16626593]
1498
Coste: 4.483411453374869 - theta: [-3.62981201  1.16631419]
[-3.62981201  1.16631419]
1499
Coste: 4.483388256587726 - theta: [-3.63029144  1.16636235]
[-3.63029144  1.16636235]
```

Al finalizar la ejecución se mostrarán y guardarán las siguientes **gráficas**:

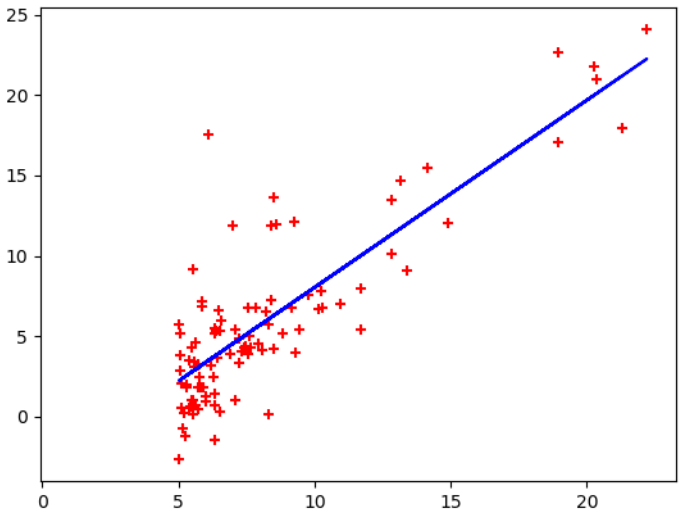


Figure 3.1: *Gráfica de la recta obtenida*

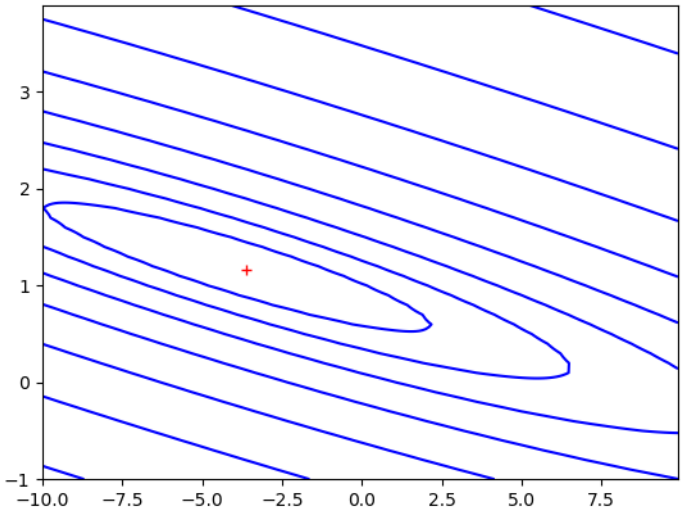


Figure 3.2: *Gráfica de contorno*

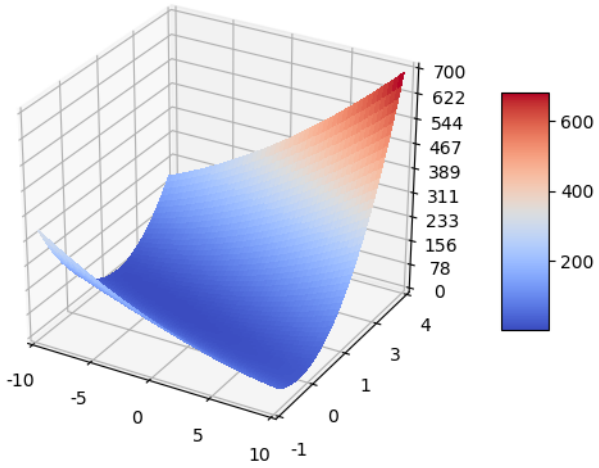


Figure 3.3: *Gráfica de superficie*

3.3. Código

A continuación presento el código de la parte 1:

```
1  import numpy as np
2  import copy
3  from pandas.io.parsers import read_csv
4  import matplotlib.pyplot as plt
5
6  from mpl_toolkits.mplot3d import Axes3D
7  from matplotlib import cm
8  from matplotlib.ticker import LinearLocator, FormatStrFormatter
9
10 fig = plt.figure()
11
12 def carga_csv(file_name):
13     valores = read_csv(file_name, header=None).values
14     # suponemos que siempre trabajaremos con float
15     return valores.astype(float)
16
17 def pinta(puntosX, puntosY):
18     plt.scatter(puntosX, puntosY, marker='+', color = "red")
19
20 def coste(X, Y, Theta):
21     H = np.dot(X, Theta)
22     Aux = (H - Y) ** 2
23     return Aux.sum() / (2 * len(X))
24
25 def gradiente(X, Y, Theta, alpha):
26     NuevaTheta = Theta
27     m = np.shape(X)[0]
28     n = np.shape(X)[1]
29     H = np.dot(X, Theta)
30     Aux = (H - Y)
31     for i in range(n):
32         Aux_i = Aux * X[:, i]
33         NuevaTheta[i] -= (alpha / m) * Aux_i.sum()
34     return NuevaTheta
35
36 def descenso_gradiente(X, Y, theta, alpha):
37     theta = gradiente(X,Y,theta,alpha)
38     costes = coste(X,Y,theta)
39     print("Coste:␣"+str(costes)+"␣-␣theta:␣"+str(theta))
40     return theta,costes
41
42
```

```
43
44 def make_data(t0_range, t1_range, X, Y,t1,t2):
45     """Genera las matrices X,Y,Z para generar un plot en 3D
46     """
47     step = 0.1
48     Theta0 = np.arange(t0_range[0], t0_range[1], step)
49     Theta1 = np.arange(t1_range[0], t1_range[1], step)
50     Theta0, Theta1 = np.meshgrid(Theta0, Theta1)
51     # Theta0 y Theta1 tienen las misma dimensiones, de forma que
52     # cogiendo un elemento de cada uno se generan las coordenadas↩
           x,y
53     # de todos los puntos de la rejilla
54     Coste = np.empty_like(Theta0)
55     for ix, iy in np.ndindex(Theta0.shape):
56         Coste[ix, iy] = coste(X, Y, [Theta0[ix, iy], Theta1[ix, ↩
               iy]])
57
58     return (Theta0, Theta1, Coste)
59
60 def main():
61     iteraciones = 1500
62
63     datos = carga_csv('ex1data1.csv')
64     X = datos[:, :-1]
65     np.shape(X)          # (97, 1)
66     Y = datos[:, -1]
67     np.shape(Y)          # (97,)
68     m = np.shape(X)[0]
69     pinta(X,Y)
70     # aniadimos una columna de 1's a la X
71     X = np.hstack([np.ones([m, 1]), X])
72
73     alpha = 0.01
74     theta = np.zeros(2)
75     i=0
76     for i in range(iteraciones):
77         print(i)
78         theta, costes =  descenso_gradiente(X, Y, theta, alpha)
79         print(theta)
80
81     plt.plot(X,theta[0] + theta[1]*X,color='blue') #recta ↩
           definida por las thetas
82     plt.savefig('p1-puntosYrecta.png')
83     plt.show()
84
```

```
85      Theta0, Theta1, Coste = make_data([-10,10], [-1,4], X, Y,↩
            theta[0],theta[1])
86      #Contorno
87      plt.figure()
88      plt.contour(Theta0, Theta1, Coste, np.logspace(-2, 3, 20), ↩
            colors='blue')
89      plt.plot(theta[0],theta[1], marker='+',color = "red")
90      plt.savefig('p1-contorno.png')
91      plt.show()
92
93      # Plot the surface.
94      fig = plt.figure()
95      ax = fig.gca(projection='3d')
96      surf = ax.plot_surface(Theta0, Theta1, Coste, linewidth=0, ↩
            antialiased=False,cmap=cm.coolwarm)
97      # Customize the z axis.
98      ax.set_zlim(0,700)
99      ax.zaxis.set_major_locator(LinearLocator(10))
100     ax.zaxis.set_major_formatter(FormatStrFormatter('%.0f'))
101
102     ax.set_xlim(-10,10)
103     ax.xaxis.set_major_locator(LinearLocator(5))
104     ax.xaxis.set_major_formatter(FormatStrFormatter('%.0f'))
105     ax.yaxis.set_major_locator(LinearLocator(5))
106     ax.yaxis.set_major_formatter(FormatStrFormatter('%.0f'))
107
108   # Add a color bar which maps values to colors.
109     fig.colorbar(surf, shrink=0.5, aspect=5)
110
111     plt.savefig('p1-3d.png')
112     plt.show()
113
114
115 main()
```

Código parte 1: p1.py

Capítulo 4

Parte 2: Regresión con varias variables

En esta parte se van a utilizar varias variables, es decir, más columnas para cada uno de los valores.

4.1. Funciones

Las funciones añadidas o modificadas para la parte 2 son las siguientes:

- **coste(X, Y, Theta):** Esta función ha sufrido modificaciones debido a que la fórmula para calcular el coste varía:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

- **gradiente(X, Y, Theta, alpha):** Esta función también ha sufrido cambios debido a que la fórmula ha cambiado:

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

- **normalizar(X):** Normaliza los datos (unidades en el caso del número de habitaciones y miles en el caso de la superficie) para acelerar la convergencia al aplicar el método de descenso de gradiente.

- **calcularNormal(X,Y):** Función que calcula la Theta normalizada. El objetivo de usar esta función es contrastar el resultado obtenido por *descenso de gradiente*.

- **main():** El *main* ha sufrido sustanciosas modificaciones.
  Ahora se calcula para diferentes valores de **alpha** *(0.1, 0.3, 0.01, 0.03, 0.001, 0.003)*.
  Para cada uno de estos valores se realizan las *1500* iteraciones en el descenso de gradiente, y se genera una **gráfica de los costes**.

Se almacenan las *thetas* y los *costes* de todas estas 6 ejecuciones y finalmente nos quedamos con aquella que menor coste tenga.

Por último, se calcula el valor que debería tener *una casa con una superficie de 1.650 pies cuadrados y 3 habitaciones*, mediante el método del **descenso de gradiente** y mediante la formula.

4.2. Ejecución

En la consola se va a ver como se imprimen los valores obtenidos en las ejecuciones para las distintas **aphas**. De esta manera:

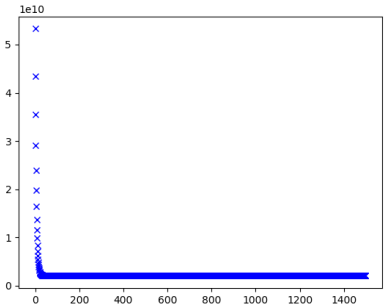
```
Alpha: 0.1 Coste: [[2.04328005e+09]] - theta: [[340412.65957447]
 [109447.79646964]
 [ -6578.35485416]]
Alpha: 0.3 Coste: [[2.04328005e+09]] - theta: [[340412.65957447]
 [109447.79646964]
 [ -6578.35485416]]
Alpha: 0.01 Coste: [[2.04328259e+09]] - theta: [[340412.56301468]
 [109371.67271736]
 [ -6502.39924935]]
Alpha: 0.03 Coste: [[2.04328005e+09]] - theta: [[340412.65957447]
 [109447.79636264]
 [ -6578.35474788]]
Alpha: 0.001 Coste: [[5.35695487e+09]] - theta: ↩
   [[264513.53515905]
 [ 74531.85304568]
 [ 18434.35121852]]
Alpha: 0.003 Coste: [[2.07833559e+09]] - theta: ↩
   [[336656.51850815]
 [101423.07405843]
 [  1350.13471268]]
[FINAL] Alpha: 0.1 Coste: 2043280050.6028287 - Theta: ↩
   [[340412.65957447]
 [109447.79646964]
 [ -6578.35485416]]
Una casa con una superficie de 1.650 pies cuadrados y 3 ↩
   habitaciones costara:
[DESCENSO] 293081.46 euros.
[FORMULA]  293081.46 euros.
```

Como se puede observar en la ejecución el alpha con el menor coste es **0.1**, como se puede ver también en las gráficas (4.2.1).

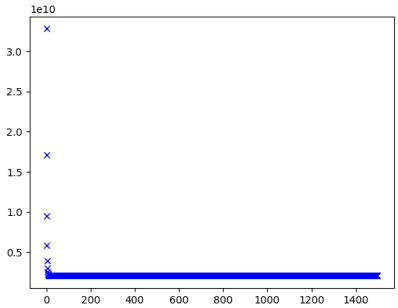
En la comprobación, comparado con la aplicación de la *theta* obtenida por la formula, el método de **descenso de gradiente** funciona **correctamente**.

4.2.1. Gráficas de coste para cada Alpha

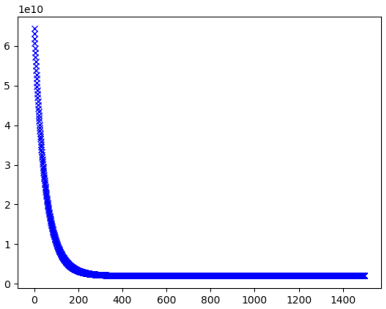
a)



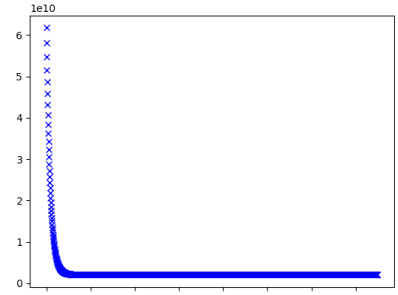
b)



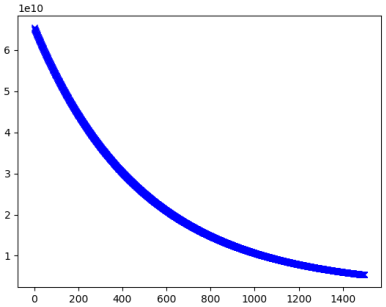
c)



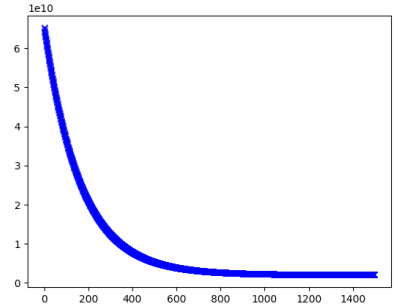
d)



e)



f)



a) Alpha 0.1 b) Alpha 0.3 c) Alpha 0.01 d) Alpha 0.03 e) Alpha 0.001 f) Alpha 0.003

4.3. Código

```
import numpy as np
import copy
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

fig = plt.figure()

def carga_csv(file_name):
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)

def coste(X, Y, Theta):
    H = np.dot(X, Theta)
    Aux = (H - Y)
    dot = np.dot(Aux.T, Aux)
    return dot / (2 * len(X))


def gradiente(X, Y, Theta, alpha):
    NuevaTheta = Theta
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    sumat = [0.] * n
    for i in range(n):
        for j in range(m):
                sumat[i] += (np.dot(X[j],Theta) - Y[j]) * X[j, i]
                    # Calcula el sumatorio con esta ecuaci n
        NuevaTheta[i] -= (alpha / m) * sumat[i]
    return NuevaTheta


def descenso_gradiente(X, Y, theta, alpha):
    theta = gradiente(X,Y,theta,alpha)
    costes = coste(X,Y,theta)
    return theta,costes


```

```
43 def normalizar(X):
44     mu = np.mean(X, axis=0)
45     sigma = np.std(X, axis=0)
46     normalX = (X - mu) / sigma
47     return (normalX, mu, sigma)
48
49
50 def calcularNormal(X,Y):
51     X = np.hstack([np.ones((len(X), 1)), X])
52     dot = np.linalg.inv(np.dot(X.T, X))
53     return np.dot(np.dot(dot, X.T), Y)
54
55
56 def main():
57     iteraciones = 1500
58     alpha = [0.1, 0.3, 0.01, 0.03, 0.001, 0.003]
59
60     datos = carga_csv('ex1data2.csv')
61     X = datos[:, 0:-1]
62     Y = datos[:, -1:]
63
64     Xn, mu, sigma = normalizar(X)
65     m = len(X)
66
67     fig = plt.figure()
68     ax = fig.gca()
69
70     costes = []
71     thetas = []
72     i=0
73     # a adimos una columna de 1's a la X
74     Xn = np.hstack([np.ones([m, 1]), Xn])
75
76     for a in alpha:
77         theta = np.array(np.ones((len(Xn[0])))).reshape(len(Xn↩
               [0]), 1)
78         fig = plt.figure()
79         ax = fig.gca()
80         for i in range(iteraciones):
81             theta, coste =  descenso_gradiente(Xn, Y, theta, a)
82             ax.plot(i, coste, 'bx')
83         thetas.append(theta)
84         costes.append(coste)
85         plt.savefig('coste-'+str(a)+'.png')
```

```
86          print("Alpha: "+str(a)+ " Coste: "+str(coste)+" - theta: "+str(theta))
87          fig.clf()
88
89      thetaFinal= thetas[np.argmin(costes)]
90      costeFinal= np.min(costes)
91      aFinal= alpha[np.argmin(costes)]
92      print("[FINAL] Alpha: "+str(aFinal)+ " Coste: "+str(costeFinal)+" - Theta: "+str(thetaFinal))
93
94      xn = np.array([1, (1650 - mu[0]) / sigma[0], (3 - mu[1]) / sigma[1]])  # Genera un neuvo dato normalizado
95      yd = np.dot(xn.T, thetaFinal)
96      ThetaFormula = calcularNormal(X, Y)
97      x = np.array([1, 1650, 3])
98      yf = np.dot(x.T, ThetaFormula)
99
100     print("Una casa con una superficie de 1.650 pies cuadrados y 3 habitaciones costar : ")
101     print("[DESCENSO] "+str(round(yd[0],2))+" euros.")
102     print("[FORMULA]  "+str(round(yf[0],2))+" euros.")
103
104 ################
105 main()
```

Código parte 2: p1.2.py