Práctica 1: Regresión lineal

Rafael Herrera Troca
Rubén Ruperto Díaz

Aprendizaje Automático y Big Data
Doble Grado en Ingeniería Informática y Matemáticas

4 de septiembre de 2020

1. Introducción

En esta práctica hemos implementado el método de descenso de gradiente para hacer regresión lineal con una y varias variables. Este algoritmo trata de calcular una función que aproxime de la mejor forma posible un cierto parámetro de un conjunto de datos a partir del resto de parámetros. De esta forma, una vez se tenga dicha función, se podrá utilizar para predecir el valor de este parámetro usando los demás. Por ejemplo, en los datos con los que hemos trabajado en la práctica, tratamos de predecir el precio de una casa en función del número de habitaciones que tiene y su superficie. Para calcular esta función, el algoritmo va variando una serie de parámetros θ de los que depende la función. Esta variación no es arbitraria, sino que se hace en la dirección del gradiente de una función de coste que indica la diferencia entre el resultado dado con el valor de θ actual y el esperado. Así, se van modificando θ mientras se intenta minimizar esta función de coste, y el valor que la hace mínima es el que se toma para construir la función de predicción.

Hemos programado el código en el entorno *Spyder* con Python 3.7.

2. Procedimiento

Para la realización de la práctica, comenzamos por definir una serie de funciones que luego utilizaremos cuando sea necesario.

La primera función, y la más importante, es *desc_grad*, que calcula el valor de θ correspondiente a un conjunto de datos dado mediante el método de descenso de gradiente. Además de los datos, se le puede dar el valor de α, el de θ^0 y el número de iteraciones. En caso de no hacerlo tomará $\alpha = 0{,}01$, $\theta_i^0 = 0\,\forall i$ y 1500 iteraciones. Para calcular θ, vamos actualizando su valor de acuerdo a la fórmula:

$$\theta_j^k = \theta_j^{k-1} - \frac{\alpha}{N}\sum_{i=0}^{N}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Para hacer esto aprovechamos la potencia que ofrece Python y la librería Numpy para hacer operaciones de forma matricial, de forma que vamos calculando todas las θ_j a la vez como sigue. Primero, hacemos $h_\theta(x^{(i)}) = \theta^T X, \forall\, 0 \leq i \leq N$, donde X es la matriz que tiene por primera fila todo unos y, en las siguientes filas, las de x. De esto obtenemos la matriz fila que tiene $h_\theta(x^{(i)})$ en la columna i. Después, le restamos la matriz fila y para a continuación, usando *broadcast*, multiplicar la matriz fila resultante por X y obtener la matriz que tiene, en la fila j, los términos del sumatorio de la fórmula para esa j. Así, usando la función *np.sum*, obtenemos la matriz columna que en cada fila tiene el resultado del sumatorio para la j correspondiente y basta multiplicar dicha matriz por $\frac{\alpha}{N}$ y restarla a θ^{k-1} para finalmente conseguir θ^k. Como comprobamos en la práctica anterior, esto es mucho más rápido que ir calculando cada θ_j de forma iterativa. Esta función devuelve además la lista de todas las θ^k por

las que ha ido pasando y el valor de la función de coste en cada una de ellas, que utilizaremos para representar las gráficas explicativas que se exponen en apartados posteriores.

Para poder obtener el valor de la función de coste para una determinada θ, hemos definido la función *cost*. En ella, se aplica la siguiente fórmula:

$$J(\theta) = \frac{1}{2N} \sum_{i=0}^{N} (h_\theta(x^{(i)}) - y^{(i)})^2$$

La forma de hacer las operaciones es muy similar a la de la función *desc_ grad*, aprovechando el cálculo matricial para obtener todos los sumandos a la vez. Hay que destacar que, pese a que se puede obtener la función de coste con cálculo vectorial directamente sin usar un sumatorio con otra fórmula, hemos preferido usar esta porque permite calcular el coste tanto para una única θ como para un array de varias usando *broadcast*, algo de lo que hacemos uso a lo largo de la práctica.

Las otras funciones que hemos definido son *ec_ norm*, que calcula el valor de θ por el método de la ecuación normal aplicando la fórmula correspondiente, y *normalizar*, que normaliza un conjunto de datos dado restando su media y dividiendo entre su desviación típica, y devuelve, además de los datos normalizados, estos dos valores.

3. Resultados

3.1. Parte 1: Regresión con una variable

En esta primera parte aplicaremos regresión lineal a un conjunto de datos sobre los beneficios obtenidos por una empresa de distribución de alimentos en función del tamaño de las ciudades donde opera. El objetivo es obtener la recta que mejor se aproxima a los datos de entrenamiento.

La aplicación del descenso de gradiente, partiendo inicialmente de $\theta^0 = [0{,}9,\ 3{,}9]$ desemboca tras 1500 iteraciones, con $\alpha = 0{,}01$, en $\theta = [-3{,}6,\ 1{,}16]$. Por tanto, la recta resultante es $y = 1{,}16x - 3{,}6$.

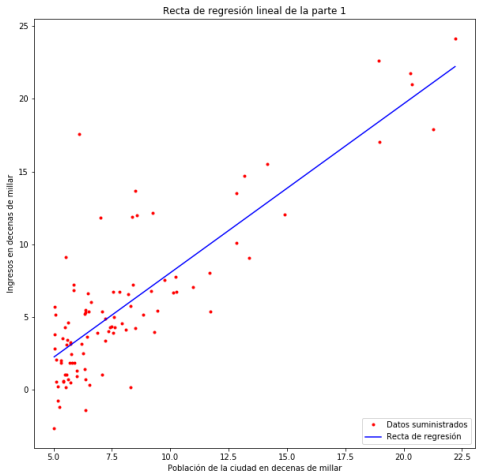

Figura 1: Recta de regresión para la primera parte

3.1.1. Comportamiento del descenso de gradiente con una variable para distintos valores de α

Para observar de qué manera afecta al descenso de gradiente variar los valores de α, decidimos superponer el recorrido seguido por θ durante el descenso a la gráfica de la superficie que forma la función de coste $J(\theta_0, \theta_1)$. Para visualizar mejor el efecto, elegimos como θ inicial [0,9, 3,9] en lugar de [0, 0], ya que se genera un recorrido más largo.

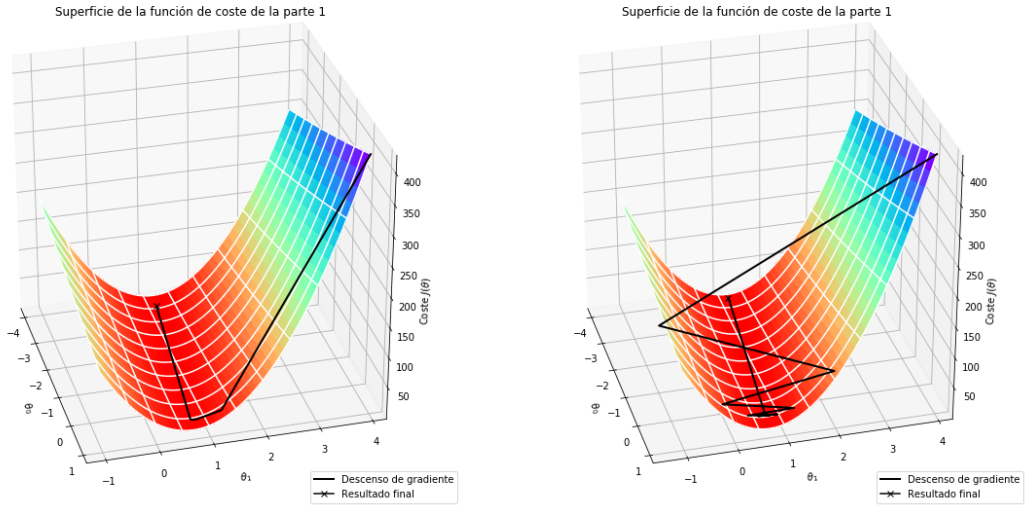

Figura 2: A la izquierda, $\alpha = 0{,}01$, a la derecha $\alpha = 0{,}02$

La forma de la gráfica de $J(\theta_0, \theta_1)$ es similar a un cuenco alargado. Observamos que para $\alpha = 0{,}01$ el descenso es muy rápido cuando nos situamos en la cara del cuenco. Al llegar a la base del cuenco, seguimos descendiendo hacia un punto central mínimo, pero a mucha menos velocidad debido a la menor pendiente que presenta la superficie.

Por otro lado, para $\alpha = 0{,}02$ los saltos iniciales son mucho más grandes que en el caso anterior, como era de esperar al aumentar α. Tanto es así, que el punto asociado al segundo θ salta a la otra cara del cuenco, pero afortunadamente a una altura menor. A partir de ahí, salta de una cara a otra, cada vez a menor altura, lo que le permite acabar convergiendo hacia la base del cuenco como ocurrió para $\alpha = 0{,}01$. Si probamos un valor más alto, como $\alpha = 0{,}03$, ya no se logra la convergencia, puesto que el salto es tan grande que pasa a la otra cara a mayor altura que antes, por lo que el valor de $J(\theta_0, \theta_1)$ crece sin parar.

3.1.2. Perpendicularidad del gradiente y las curvas de nivel

Al dibujar la gráfica de las curvas de nivel de la función de coste $J(\theta)$ pensamos que podría ser ilustrativo añadir también el recorrido que ha ido siguiendo θ mientras avanzaba el descenso de gradiente, al igual que hicimos sobre la gráfica de la

superficie. Esperábamos que este recorrido fuera perpendicular a las curvas de nivel, pues hay un resultado matemático que lo demuestra, sin embargo, nos descolocó ver que en la gráfica no se cumplía este hecho.

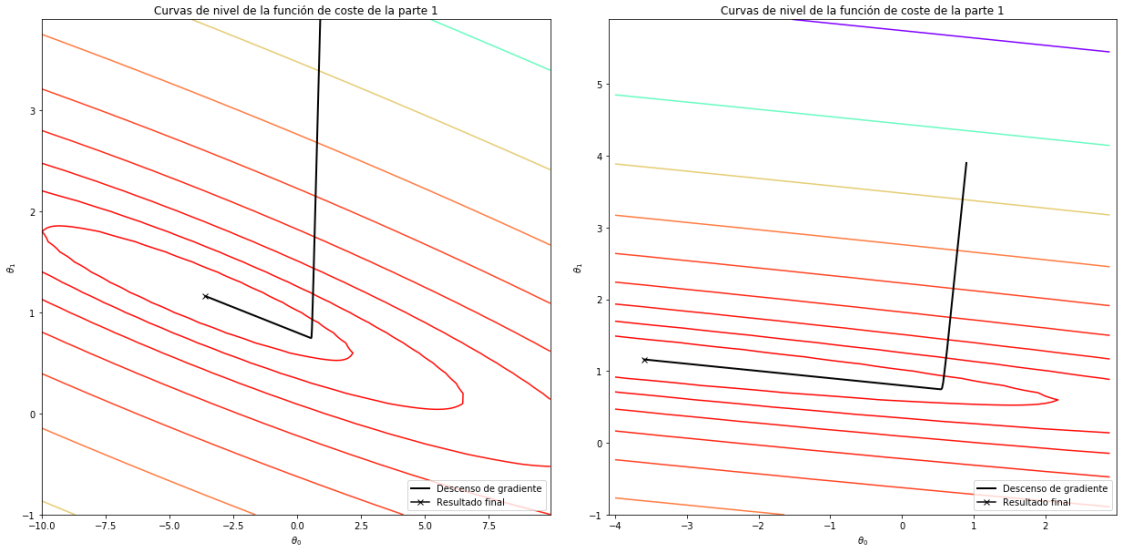

Figura 3: Aunque las dos imágenes muestran los mismos datos, las escalas de los ejes alteran visualmente los ángulos entre los elementos que aparecen

Tras una búsqueda en internet sobre la cuestión, encontramos que tener los ejes de la gráfica a distinta escala puede distorsionar la perpendicularidad. En efecto, comprobamos que eso pasaba en nuestra gráfica (figura 3, izquierda) y que al corregirlo queda patente en ella que el gradiente es perpendicular a las curvas de nivel (figura 3, derecha).

3.2. Parte 2: Regresión con varias variables

En este caso, el conjunto de datos consiste en precios y características de casas vendidas en Portland, Oregon. Queremos obtener una función que permita predecir el precio de una casa nueva en función del resto de atributos, que son la superficie de la casa en pies cuadrados y el número de habitaciones.

En primer lugar, aplicamos el método de descenso de gradiente vectorizado para minimizar la función de coste $J(\theta)$. Partiendo de $\theta^0 = [0, 0, 0]$, con $\alpha = 0{,}01$ y 1500 iteraciones, obtenemos como resultado $\theta = [340412{,}56,\ 109370{,}06,\ -6500{,}62]$. Nótese que estos valores están ajustados con atributos normalizados, para acelerar la convergencia en el descenso. Evaluando la función $h_\theta(x) = \theta^T x$ para una casa de 1650 pies cuadrados y 3 habitaciones, obtenemos una predicción del precio de 293098,47 dólares.

Además, hemos utilizado el método de la ecuación normal, que calcula el valor óptimo de θ mediante la expresión $\theta = (X^T X)^{-1} X^T \vec{y}$. Tras el cálculo resulta $\theta = [89597{,}91,\ 139{,}21,\ -8738{,}02]$. La diferencia respecto al valor θ obtenido mediante descenso de gradiente se explica porque en este caso los datos de entrenamiento no han sido normalizados. La predicción del precio de la casa de 1650 pies cuadrados y 3 habitaciones en este caso es de 293081,46 dólares, prácticamente igual que la obtenida por el otro método.

3.2.1. Comportamiento del descenso del gradiente multivariable para distintos valores de α

Hemos estudiado la rapidez con la que converge el descenso de gradiente con los datos de las casas de Oregon en función de algunos valores de α. Para ello, hemos guardado los valores de $J(\theta)$ a lo largo del proceso y los hemos representado en una gráfica, comparando los datos obtenidos con $\alpha \in \{0{,}3,\ 0{,}1,\ 0{,}03,\ 0{,}01,\ 0{,}003,\ 0{,}001\}$.

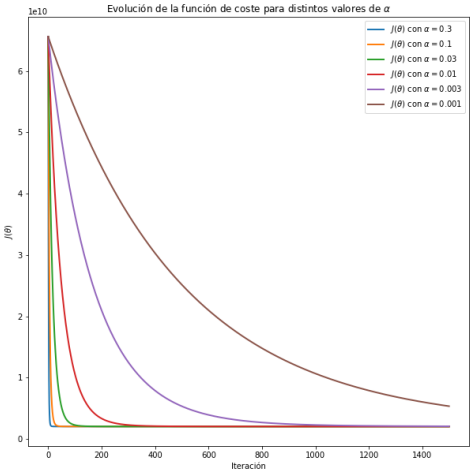

Figura 4: Comparativa del descenso de gradiente multivariable para varios α

Observamos que para valores más grandes de α se alcanza el mínimo en un menor número de iteraciones. Tan sólo para $\alpha = 0{,}001$ no se alcanza el mínimo, ya que a esa velocidad de descenso no son suficientes las 1500 iteraciones que han bastado para el resto. Sin embargo, aumentando el número de iteraciones se alcanzaría la convergencia.

4. Conclusiones

Esta práctica nos ha permitido comprender con gran nivel de detalle el funcionamiento del descenso de gradiente como un método eficaz para minimizar funciones de coste. Gracias a los gráficos realizados con la librería *Matplotlib* hemos podido visualizar el comportamiento del método durante su ejecución, a través de la evolución de θ y el valor de la función $J(\theta)$.

Es destacable la potencia que ofrece la librería *Numpy* para vectorizar cálculos, permitiendo escribir el algoritmo del descenso de gradiente en un código muy compacto e incrementando la eficiencia durante su ejecución. Sin embargo, a pesar de

la ventajas que ofrece, también tiene el inconveniente de que a veces aumenta la confusión sobre qué cálculos se están haciendo realmente en cada instrucción.

5. Anexo: Código

A continuación se ofrece el código escrito para resolver los ejercicios propuestos en la práctica.

```
'''
Práctica 1

Rubén Ruperto Díaz y Rafael Herrera Troca
'''

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def desc_grad(x, y, alpha=0.01, theta=None, it=1500):
    X = np.vstack((np.ones(len(y)), x.T))
    y = np.array(y)
    if (theta == None):
        theta = np.zeros(np.shape(X)[0])
    thetas = np.array([theta])
    for i in range(it):
        h = np.dot(theta, X)
        sumandos = (h - y) * X
        theta = theta - (alpha / len(y)) * np.sum(sumandos,1)
        thetas = np.append(thetas, [theta], axis=0)
    return theta, thetas, cost(thetas, X, y)

def ec_norm(x, y):
    X = np.hstack((np.array([np.ones(len(y))]).T, x))
    y = np.array(y)
    theta = np.dot(np.dot(np.linalg.pinv(np.dot(X.T,X)),X.T),y)
    return theta

def cost(theta, X, y):
    h = np.dot(theta, X)
    sumandos = (h - y)**2
    return 1 / (2 * len(y)) * np.sum(sumandos, 1)

def normalizar(x):
    mu = np.mean(x, 0)
    sigma = np.std(x, 0)
    x2 = (x - mu) / sigma
    return x2, mu, sigma


# Parte 1: Regresión con una variable

```

```
46
47 # Leemos los datos que usaremos para calcular la recta de regresión
48 data = pd.read_csv("ex1data1.csv", names=['población', 'beneficios'
       ])
49
50 # Utilizamos nuestra función desc_grad para calcular los valores de
       theta y
51 # además se obtienen los valores que ha ido tomando theta durante
       el proceso
52 # y el coste para cada uno
53 x = np.array(data['población'].tolist()).T
54 y = data['beneficios'].tolist()
55 theta, thetas, costes = desc_grad(x, y, theta=[0.9,3.9])
56
57 # Representamos los datos y la recta de regresión obtenida
58 part = np.linspace(min(data['población']),max(data['población'])
       ,1000)
59 plt.figure(figsize=(10,10))
60 plt.plot(data['población'], data['beneficios'], 'r.', label="Datos
       suministrados")
61 plt.plot(part, theta[0]+theta[1]*part, 'b', label="Recta de
       regresión")
62 plt.title("Recta de regresión lineal de la parte 1")
63 plt.xlabel("Población de la ciudad en decenas de millar")
64 plt.ylabel("Ingresos en decenas de millar")
65 plt.legend(loc="lower right")
66 plt.savefig("p1recta.png")
67 plt.show()
68
69 # Representamos la superficie de la función de coste según el valor
       de
70 # theta y los valores que ha ido recorriendo nuestro descenso de
       gradiente
71 Theta0, Theta1 = np.meshgrid(np.arange(-4,1,0.1), np.arange
       (-1,4,0.1))
72 Costes = cost(np.transpose(np.array([np.ravel(Theta0),np.ravel(
       Theta1)])), np.array([np.ones(len(y)),x]), np.array(y))
73 Costes = np.reshape(Costes, np.shape(Theta0))
74 fig = plt.figure(figsize=(10,10))
75 ax = fig.gca(projection='3d')
76 ax.plot_surface(Theta0, Theta1, Costes, cmap=cm.rainbow_r,
       linewidth=0, antialiased=False)
77 ax.plot_wireframe(Theta0, Theta1, Costes, rstride=5, cstride=5,
       color='white')
78 ax.plot(thetas[:,0], thetas[:,1], costes, color='black', linewidth
       =2, zorder=4, label="Descenso de gradiente")
79 ax.plot([theta[0]], [theta[1]], [costes[-1]], color='black', marker
       ='x', zorder=5, label="Resultado final")
80 ax.view_init(None,-15)
81 plt.title("Superficie de la función de coste de la parte 1")
82 ax.set_xlabel(r"$\theta_0$")
83 ax.set_ylabel(r"$\theta_1$")
84 ax.set_zlabel(r"Coste $J(\theta)$")
85 plt.legend(loc="lower right")
86 plt.savefig("p1sup.png")
```

```
plt.show()

# Representamos las curvas de nivel de la función de coste según el valor
# de theta y los valores que ha ido recorriendo nuestro descenso de gradiente
Theta0, Theta1 = np.meshgrid(np.arange(-10,10,0.1), np.arange(-1,4,0.1))
Costes = cost(np.transpose(np.array([np.ravel(Theta0),np.ravel(Theta1)])), np.array([np.ones(len(y)),x]), np.array(y))
Costes = np.reshape(Costes, np.shape(Theta0))
fig = plt.figure(figsize=(10,10))
plt.contour(Theta0, Theta1, Costes, np.logspace(-2,3, 20), cmap=cm.rainbow_r)
plt.plot(thetas[:,0], thetas[:,1], color='black', linewidth=2, label="Descenso de gradiente")
plt.plot(theta[0], theta[1], color='black', marker='x', label="Resultado final")
plt.title("Curvas de nivel de la función de coste de la parte 1")
plt.xlabel(r"$\theta_0$")
plt.ylabel(r"$\theta_1$")
plt.axis('equal')
plt.legend(loc="lower right")
plt.savefig("p1niv.png")
plt.show()


# Parte 2: Regresión con varias variables


# Leemos los datos que usaremos
data = pd.read_csv("ex1data2.csv", names=['superficie', 'habitaciones', 'precio'])

# Normalizamos los datos de entrada del algoritmo
x = np.array([data['superficie'].tolist(), data['habitaciones'].tolist()]).T
normx, mu, sigma = normalizar(x)

# Utilizamos nuestra función desc_grad para calcular hacer la regresión lineal
# variando la tasa de aprendizaje. Además, vamos almacenando los valores que
# han ido tomando theta y la función de coste durante el proceso para cada
# posible valor de alfa
# Además, representamos en una gráfica los valores que va tomando la función
# de coste a medida que avanza el descenso de gradiente para cada valor de la
# tasa de aprendizaje
y = data['precio'].tolist()
plt.figure(figsize=(10,10))
for alfa in [0.3, 0.1, 0.03, 0.01, 0.003, 0.001]:
    theta, thetas, costes = desc_grad(normx,y,alpha=alfa)
```

```
    plt.plot(range(len(costes)), costes, linewidth=2, label=r"$J(\
    theta)$ con $\alpha=$" + str(alfa))
plt.title(r"Evolución de la función de coste para distintos valores
    de $\alpha$")
plt.xlabel("Iteración")
plt.ylabel(r"$J(\theta)$")
plt.legend(loc="upper right")
plt.savefig("p2cost.png")
plt.show()

# Finalmente comprobamos que el resultado obtenido con el descenso
    de
# gradiente es correcto comparándolo con el que se obtiene a partir
    de la
# ecuación normal
thetaGrad = desc_grad(normx, y)[0]
thetaNorm = ec_norm(x, y)

ex = np.array([1650, 3])
normEx = (ex-mu)/sigma
ex = np.append([1], ex)
normEx = np.append([1], normEx)

print("Usando el descenso de gradiente, la predicción es y =", np.
    dot(thetaGrad, normEx))
print("Usando la ecuación normal, la predicción es y =", np.dot(
    thetaNorm, ex))
```