Aprendizaje Automático y Big Data, Práctica 1

Romain Contrain y Guillermo Martín Sánchez

Febrero 2020

1 Objetivo

El objetivo de esta práctica es implementar el algoritmo de regresión lineal. Se trata de un algoritmo de aprendizaje supervisado de regresión que busca encontrar la ecuación de la recta que se ajuste más a ciertos puntos. En concreto, buscamos el vector θ tal que la recta $h_\theta = \theta^T x$ minimice la función coste $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

2 Método

Hemos usado el método de descenso de gradiente para calcular el mínimo de dicha función. En general este método se aproxima a un mínimo local, sin embargo, por la naturaleza parabólica de la función de coste, esta tiene un único mínimo local que es, a su vez, un mínimo global. En este caso tenemos que

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

Por tanto el procedimiento de actualización de θ que usamos es:

$$\begin{cases}\theta_j^0 = 0 \quad \forall j \\ \theta_j^{i+1} = \theta_j^i - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_{\theta^i}(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \forall j\end{cases}$$

hasta que un número de iteraciones de 1500 o la norma del gradiente sea menor que cierto $\epsilon = 0.001$

Primero analizamos el caso de una variable cuyos datos tienen como atributo lo población de una ciudad en $10.000s$ y la variable a predecir es cuántos beneficios ganaría la compañía en esa ciudad, en $\$10.000s$. Luego estudiamos el caso multivariable cuyos datos tienen como atributos el número de pies cuadrados y de habitaciones que tiene una casa y la variable a predecir es el precio por el que se vende.

En el segundo caso hemos hecho uso de normalización para facilitar la convergencia del método. Para ello, para cada atributo j, hemos hecho

$$\hat{X}_j = \frac{X_j - \mu_j}{\sigma_j} \tag{1}$$

3 Resultados

3.1 Una variable

Como podemos ver en la Figura 1, la función de coste tiene forma parabólica. En concreto, se ve mejor en la Figura 2 que representa las líneas de contorno, que esto es así y que, además, el resultado del método de descenso de gradiente $\theta_{min} \approx (-3.63, 1.17)$ (marcado con una cruz roja) efectivamente se encuentra cerca del mínimo. En la Figura 3, vemos los puntos que intentamos aproximar por una recta en azul y en rojo, la recta que hemos encontrado como óptima ($h_{\theta_{min}}$)

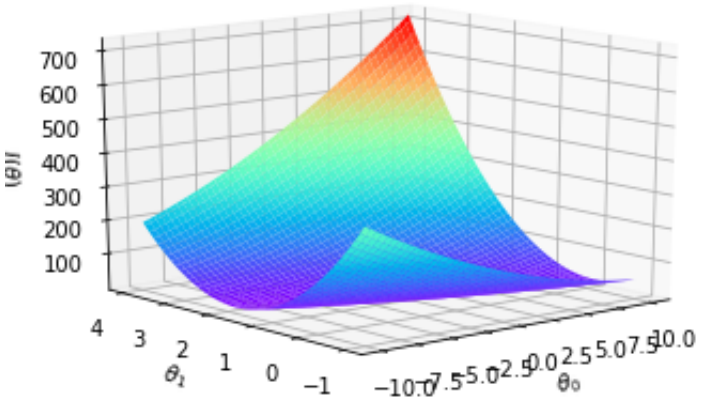

Figure 1: Función de coste $J(\theta)$ para distintos valores de θ_0 y θ_1

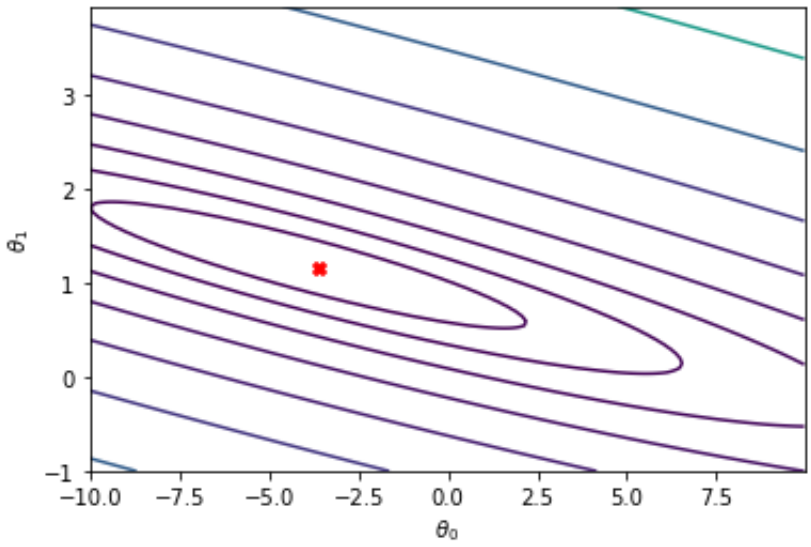

Figure 2: Función de coste $J(\theta)$ para distintos valores de θ_0 y θ_1. La cruz representa el mínimo calculado con el descenso de gradiente

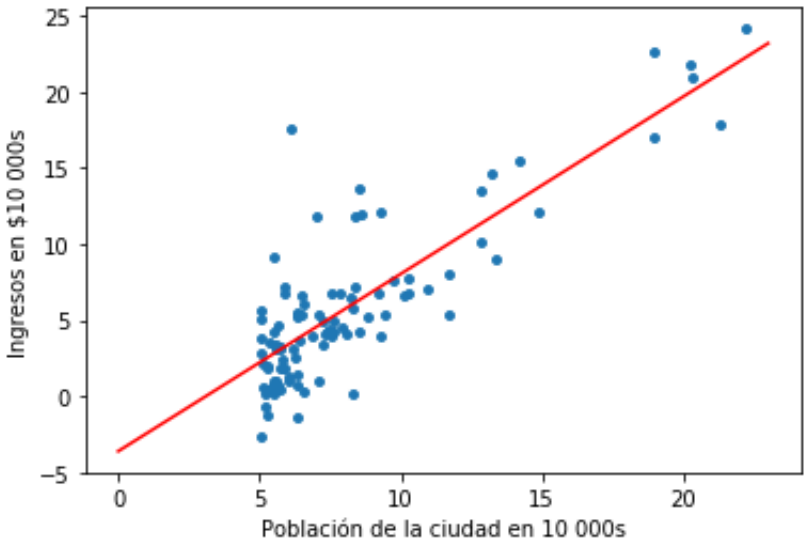

Figure 3: Puntos del dataset en azul y recta que los aproxima en rojo

3.2 Multivariable

En este caso hemos probado distintos valores de $\alpha = \{0.3, 0.1, 0.03, 0.01, 0.003, 0.001\}$ para ver cómo se comporta la función de coste en el descenso de gradiente. Como se esperaba, y se ve reflejado en la Figura 4, cuanto mayor sea α más rápido converge el método. Hay que tener en cuenta que esto sólo ocurre para αs suficientemente pequeños: cuando es muy grande la función de coste deja de ser estrictamente decreciente y el método puede no converger.

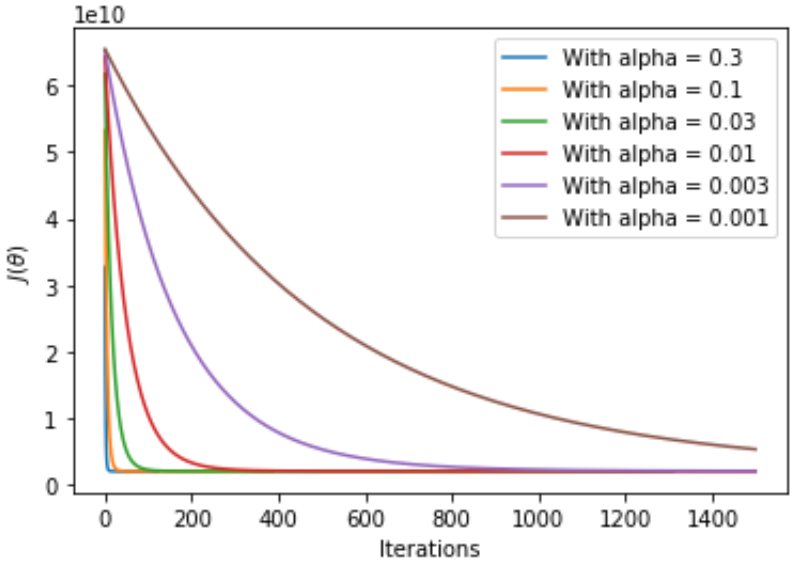

Figure 4: Valor del coste $J(\theta)$ en cada iteración para distintos parámetros de aprendizaje α

Para $\alpha = 0.03$ obtenemos $\theta_{min} \approx (340412.66, 109447.79, -6578.35)$.

Por otro lado hemos implementado el método de la ecuación normal que nos da $\overline{\theta}_{min}$ mediante una fórmula exacta en vez de con un método numérico:

$$\overline{\theta}_{min} = (X^T X)^{-1} X^T Y \tag{2}$$

Después hemos querido obtener una estimación del precio de una casa con $1650m^2$ y 3 habitaciones. Para ello hemos normalizado estos valores con los $\mu_{1,2}$ y $\sigma_{1,2}$ obtenidos durante el proceso de normalización de la Eq 1 para obtener $\hat{x}$ y hemos calculado $h_{\theta_{min}}(\hat{x})$ usando el resultado del descenso de gradiente θ_{min}. Además, lo hemos comparado el valor obtenido con una mejor estimación: $h_{\overline{\theta}_{min}}(x)$ usando el resultado de la ecuación normal $\overline{\theta}_{min}$ (Eq 2). En el caso de descenso de gradiente hemos obtenido $y = 293081.46468417434$ y en el caso de la ecuación normal $y = 293081.4643349892$

Como se puede ver, ambos resultados son muy similares con un error relativo $\approx 1.19 \cdot 10^{-9}$. Al ser precio, de hecho, ambos serían interpretados como $y = \$293,081.46$ y por tanto nos darían el mismo exacto valor predicho.

4 Conclusiones

La regresión lineal es una técnica de regresión de aprendizaje supervisado sencilla de implementar y que, para estos datos, da buenos resultados. Aunque existe la ecuación normal para calcular de forma exacta un $\overline{\theta}_{min}$, en general para matrices grandes la inversa puede ser muy costosa de calcular o incluso no existir. Haber implementado este método nos ha servido como introducción a muchos conceptos importantes y generales del aprendizaje automático como la función coste, el descenso de gradiente, la normalización, el coeficiente de aprendizaje, etc.

5 Código

```
# -*- coding: utf-8 -*-
"""
Coursework 1: Linear regression
"""

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

# Load data
def carga_csv(file_name):
    valores=read_csv(file_name,header=None).values
    return valores.astype(float)

# Normal equation for exact computation of theta
def ecuacion_normal(X, Y):
    X_t = np.transpose(X)
    Aux = np.dot(X_t,X)
    X_plus = np.linalg.pinv(Aux).dot(X_t)
    return X_plus.dot(Y)

# Normalize dataset
def normalizar(X):
    mu = np.mean(X,  axis=0)
    sigma = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
```

```
    return X_norm, mu, sigma

# Compute the gradient of the cost function
def gradiente(X, Y, theta):
    res = np.zeros((np.shape(X)[1],1))
    m = np.shape(X)[0]
    H = np.dot(X,theta)
    res = 1/m * np.dot(np.transpose(X), H-Y)
    return res

# Compute the cost function
def coste(X, Y, theta):
    H = np.dot(X, theta)
    Aux = (H - Y)**2
    return Aux.sum() / (2*len(X))

# Compute gradient descent for a maximum of maxIter iterations or until
# the norm of the gradient is lower than eps
def descenso_gradiente(X, Y, alpha, maxIter, eps):
    theta = np.zeros((np.shape(X)[1],1))
    grad = gradiente(X, Y, theta)
    it = 0
    costes = []
    while np.linalg.norm(grad) > eps and it < maxIter:
        theta = theta - alpha*grad
        grad = gradiente(X, Y, theta)
        costes.append(coste(X, Y, theta))
        it += 1
    return theta, costes

#%%
"""
One variable case
"""
datos = carga_csv('ex1data1.csv')
X = datos[:,:-1]
m = np.shape(X)[0]
Y = datos[:,-1]


# Add the column of 1s
X=np.hstack([np.ones([m,1]),X])
Y = Y[:,np.newaxis]

alpha = 0.01
maxIter = 1500
```

```
eps = 1e-3
Thetas, costes = descenso_gradiente(X, Y, alpha, maxIter, eps)

# Plot data an line
x = np.linspace(0,23,23)
plt.scatter(X[:,1],Y,s=15)
plt.plot(x, Thetas[0] + Thetas[1]*x, c='r')
plt.xlabel('Población de la ciudad en 10 000s')
plt.ylabel('Ingresos en $10 000s')

#%%
"""
Plot cost function
"""
# Plot function in 3D
fig = plt.figure()
ax = fig.gca(projection = '3d')

Th0 = np.arange(-10,10,0.05)
Th1 = np.arange(-1,4,0.05)
Th0m,Th1m = np.meshgrid(Th0,Th1)
cost = np.zeros((len(Th1),len(Th0)))
for i in range(len(Th1)) :
    for j in range(len(Th0)):
        cost[i][j] = coste(X,Y, np.array([[Th0[j]],[Th1[i]]]))
curve = ax.plot_surface(Th0m,Th1m,cost, cmap = cm.rainbow)
ax.view_init(elev = 15, azim=230)
plt.xlabel(r'$\theta_{0}$')
plt.ylabel(r'$\theta_{1}$')
ax.set_zlabel(r'$J(\theta)$',rotation=0)

# Plot function as contours
plt.figure()
plt.contour(Th0m,Th1m,cost, np.logspace(-2,3,20))
plt.scatter(Thetas[0],Thetas[1], marker='X',c='r')
plt.xlabel(r'$\theta_{0}$')
plt.ylabel(r'$\theta_{1}$')

#%%
"""
For multivariate case, check different learning rates alpha
"""
datos = carga_csv('ex1data2.csv')
X_old = datos[:,:-1]
m = np.shape(X_old)[0]
Y_old=datos[:,-1]
```

```
# Normalize data
X, mu_x, sigma_x = normalizar(X_old)

# Add the column of 1s
X_old = np.hstack([np.ones([m,1]),X_old])
X = np.hstack([np.ones([m,1]),X])
Y = Y_old[:,np.newaxis]

alpha_arr = [0.3,0.1,0.03,0.01,0.003,0.001]
maxIter = 1500
eps = 1e-3
plt.figure()
for alpha in alpha_arr:
    Thetas, costes = descenso_gradiente(X, Y, alpha, maxIter, eps)
    plt.plot(costes, label = ("With alpha = " + str(alpha)))
plt.legend()
plt.xlabel('Iterations')
plt.ylabel(r'$J(\theta)$')
#%%
"""
Multivariate case
"""
datos = carga_csv('ex1data2.csv')
X_old = datos[:,:-1]
m = np.shape(X_old)[0]
Y_old=datos[:,-1]

# Normalize data
X, mu_x, sigma_x = normalizar(X_old)

# Add the column of 1s
X_old = np.hstack([np.ones([m,1]),X_old])
X = np.hstack([np.ones([m,1]),X])
Y = Y_old[:,np.newaxis]

alpha = 0.3
maxIter = 1500
eps = 1e-3
Thetas, costes = descenso_gradiente(X, Y, alpha, maxIter, eps)

# Compute with the normal equation and compare
th = ecuacion_normal(X_old,Y)
test = np.array([1650,3])
test_n = (test - mu_x)/sigma_x
test = np.hstack([1,test])
```

```
test_n = np.hstack([1,test_n])
price = np.dot(test,th)[0]
price_n = np.dot(test_n,Thetas)[0]
print("The value computed with the normal equation for x = [1650,3] is y = ", price)
print("The value computed with gradient descent for x = [1650,3] is y = ", price_n)
print("The relative distance between the two values is ", np.abs((price - price_n)/ price))
```