Práctica 1 – Regresión lineal

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

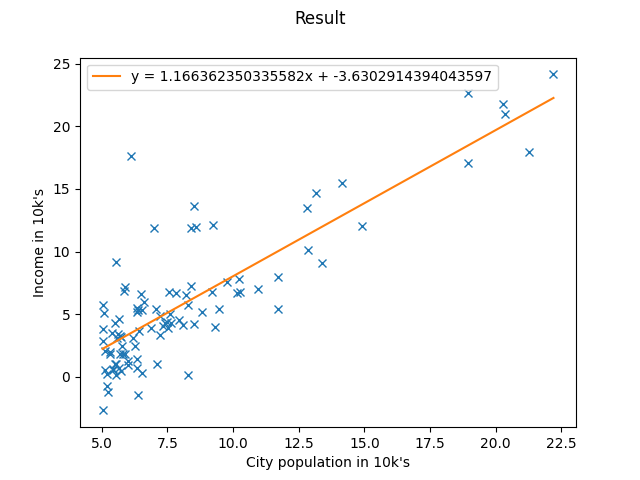
* Objetivo de la práctica

El objetivo de la práctica es aprender a aplicar la regresión lineal a un conjunto de datos, ya sea a un modelo con una variable o con varias.

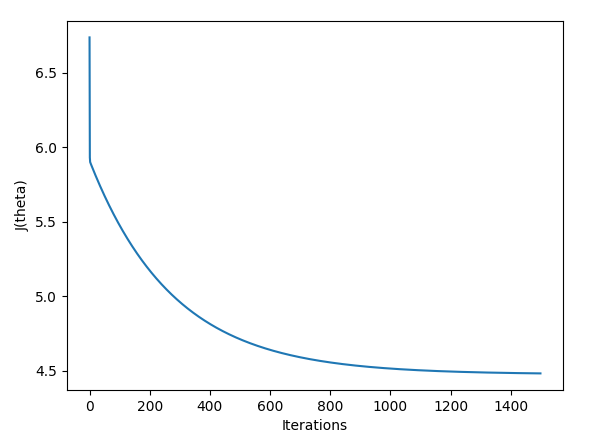
Se trata de aplicar el método de la regresión lineal a dos archivos csv. El primero de ellos representa los datos sobre beneficios de una compañía (la segunda columna) en base a la población (la primera columna). Tendremos que aplicar el método del descenso de gradiente para encontrar los parámetros Theta que definen la recta que se ajuste a los datos. Posteriormente, se aplicará la regresión lineal a unos datos con más variables: el precio de la casa, el tamaño de la casa y el número de habitaciones. También se aplicará el descenso de gradiente.

* Código de la práctica: parte 1 (una variable)
* import numpy as np  
  import matplotlib.pyplot as plt  
  from pandas.io.parsers import read\_csv  
  from mpl\_toolkits.mplot3d import Axes3D  
  from matplotlib import cm  
  from matplotlib.ticker import LinearLocator, FormatStrFormatter  
    
    
  def getMat(file\_name):  
   return read\_csv(file\_name, header=None).to\_numpy().astype(float)  
    
    
  # Prediction of the value y. In this case using linear regression.  
  def hypothesis(X, theta\_0, theta\_1):  
   return X \* theta\_1 + theta\_0  
     
    
  # Cost function J(theta), measures how good are our guesses  
  def cost(X, Y, theta\_0, theta\_1):  
   return (1 / (2 \* len(X))) \* (((hypothesis(X, theta\_0, theta\_1) - Y)\*\*2).sum())  
    
    
  def gradientDescent(X, Y, iterations, alpha):  
   theta\_0 = theta\_1 = 0  
   costs = []  
    
   for i in range(iterations):  
   costs.append(cost(X, Y, theta\_0, theta\_1))  
   H = hypothesis(X, theta\_0, theta\_1)  
   temp\_0 = theta\_0 - (alpha / len(X)) \* (H - Y).sum()  
   temp\_1 = theta\_1 - (alpha / len(X)) \* ((H - Y) \* X).sum()  
   theta\_0 = temp\_0  
   theta\_1 = temp\_1  
    
   plt.suptitle('Cost')  
   plt.xlabel('Iterations')  
   plt.ylabel('J(theta)')  
   plt.plot(np.arange(0, iterations), costs)  
   plt.savefig('costs.png')  
   plt.show()  
   return theta\_0, theta\_1  
    
    
  def plot\_surfaces(theta\_0\_, theta\_1\_, theta\_0\_range, theta\_1\_range, X, Y):  
   step = .1  
   theta\_0 = np.arange(theta\_0\_range[0], theta\_0\_range[1], step)  
   theta\_1 = np.arange(theta\_1\_range[0], theta\_1\_range[1], step)  
   theta\_0, theta\_1 = np.meshgrid(theta\_0, theta\_1)  
   Cost = np.empty\_like(theta\_0)  
   for i\_x, i\_y in np.ndindex(theta\_0.shape):  
   Cost[i\_x, i\_y] = cost(X, Y, theta\_0[i\_x, i\_y], theta\_1[i\_x, i\_y])  
     
   fig = plt.figure()  
   ax = Axes3D(fig)  
   ax.plot\_surface(theta\_0, theta\_1, Cost, cmap=cm.coolwarm)  
   plt.show()  
     
   plt.contour(theta\_0, theta\_1, Cost, np.logspace(-2, 3, 20), colors='red')  
   plt.scatter(theta\_0\_, theta\_1\_)  
   plt.show()  
    
    
  def main():  
   data = getMat('./ex1data1.csv')  
   X = data[:, 0]  
   Y = data[:, 1]  
   alpha = 0.01  
   iterations = 1500  
    
   theta\_0, theta\_1 = gradientDescent(X, Y, iterations, alpha)  
   x = np.linspace(min(X), max(X), 100)  
   y = theta\_0 + theta\_1 \* x  
     
   plt.suptitle('Result')  
   plt.xlabel('City population in 10k\'s')  
   plt.ylabel('Income in 10k\'s')  
   plt.plot(X, Y, 'x')  
   plt.plot(x, y, label=('y = ' + str(theta\_1) + 'x + ' + str(theta\_0)))  
   plt.legend()  
   plt.savefig('result.png')  
   plt.show()  
    
   plot\_surfaces(theta\_0, theta\_1, [-10, 10], [-1, 4], X, Y)  
     
    
  main()
* Resultados de ejecución: parte 1 (una variable)

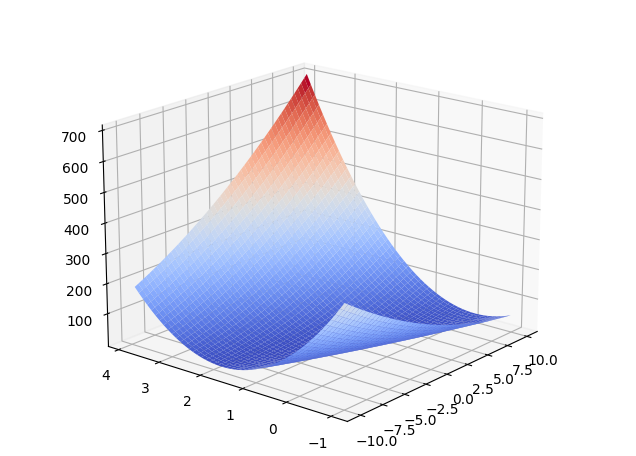
Se generan varias gráficas. En primer lugar, hemos obtenido la gráfica que los datos de entrada junto a la línea de predicción que mejor se ajusta con esta hipótesis.



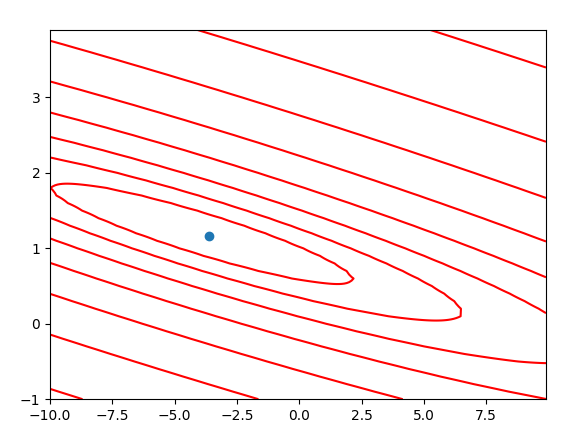
Además, hemos generado gráficas con respecto al coste. Podemos ver como el coste se va estabilizando rápidamente con Alpha = 0,01, y a la par que van aumentando las iteraciones disminuye lentamente.



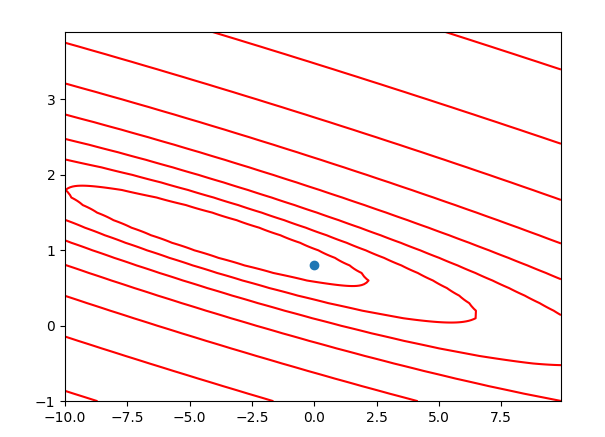
Se ha calculado también el coste dentro de los intervalos θ0 ∈ [−10, 10] y θ1 ∈ [−1, 4].



En la segunda gráfica, hemos pintado el mínimo de coste obtenido por el descenso de gradiente.



Con un valor Alpha menor, podemos comprobar que no encontramos un mejor mínimo, si no que obtenemos uno peor.



* Código de la práctica: parte 2 (varias variables)
* def getMat(file\_name):  
   return read\_csv(file\_name, header=None).to\_numpy().astype(float)  
    
    
  def gradient(X, Y, T, alpha):  
   new\_T = T  
   m = np.shape(X)[0]  
   n = np.shape(X)[1]  
   H = np.dot(X, T)  
   aux = (H - Y)  
   for i in range(n):  
   aux\_i = aux \* X[:, i]  
   new\_T[i] -= (alpha / m) \* aux\_i.sum()  
   return new\_T  
    
    
  def cost(X, Y, T):  
   XTY = np.matmul(X, T) - Y  
   return 1 / (2 \* np.shape(X)[0]) \* (np.matmul(np.transpose(XTY), XTY))  
    
  def normalize(X):  
   ranges = [ 0 ]  
   averages = [ 1 ]  
   XNorm = np.copy(X)  
   for i in range(1, np.shape(X)[1]):  
   col = XNorm[:, i]  
   ran = np.max(col) - np.min(col)  
   avg = np.average(col)  
   col -= avg  
   col /= ran  
   ranges.append(ran)  
   averages.append(avg)  
   averages = np.array(averages)  
   ranges = np.array(ranges)  
   return XNorm, averages, ranges  
    
  def normalize2(X):  
   averages = np.mean(X, axis=0)  
   ranges = np.std(X, axis=0)  
   return X/(averages - ranges), averages, ranges  
  def main():  
    
   data = getMat('./ex1data2.csv')  
   X = data[:, :-1]  
   X = np.hstack([np.ones([len(X), 1]), X])  
   Y = data[:, -1]  
   XNorm,averages,ranges = normalize(X)  
    
   costs = []  
   alpha = 0.01  
   iterations = 1500  
   T = np.zeros(np.shape(X)[1])  
   for i in range(iterations):  
   T = gradient(XNorm, Y, T, alpha)  
   costs.append(cost(XNorm, Y, T))  
    
   plt.suptitle('Cost')  
   plt.xlabel('Iterations')  
   plt.ylabel('J(theta)')  
   plt.plot(np.arange(0, iterations), costs)  
   plt.savefig('costs.png')  
   plt.show()  
    
   print(T)  
   for i in range(len(X)):  
   print(np.dot(np.transpose(T), X[i]))  
    
  main()
* Conclusiones

Se concluye que el uso de vectores numpy es muy útil para tareas con arrays, por lo que en tareas que se utilicen arrays con muchos datos será conveniente utilizarlos en vez de los nativos de Python.