Práctica 2 – Regresión logística

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

* Objetivo de la práctica

En esta práctica, se aplicará la regresión logística a dos conjuntos de datos, y se tratará de predecir si ‘aceptamos o no’.

El primer conjunto de datos representa las notas obtenidas en dos exámenes por una serie de candidatos, junto a un booleano que indica si fueron aceptados o no. Deberemos construir un modelo por regresión logística que estime si un estudiante será admitido o no según sus notas. En este conjunto de datos se ve claramente que se puede realizar una división con una línea recta, es decir, son linealmente separables.

El segundo conjunto de datos representa dos pruebas aplicadas a una serie de microchips, junto al resultado booleano que indica si pasan la prueba de calidad o no. En este conjunto de datos la cosa se complica, ya que no son linealmente separables. Por ello, la región de división no vendrá definida por una simple recta, si no que será una región más compleja.

* Código de la práctica: parte 1 (datos linealmente separables)

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

from pandas.io.parsers import read\_csv

import scipy.optimize as opt

def load\_data(file\_name):

    return read\_csv(file\_name, header=None).values

def print\_data(X, Y):

    pos = np.where(Y == 1)

    # Dibuja los ejemplos positivos

    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')

    pos = np.where(Y == 0)

    plt.scatter(X[pos, 0], X[pos, 1], marker='o', c='b')

    plt.savefig("data.pdf")

    plt.legend(

        handles=[

            mpatches.Patch(color='black', label='Accepted'),

            mpatches.Patch(color='blue', label='Refused')

        ])

def sigmoide(Z):

    return 1 / (1 + np.exp(-Z))

def coste(theta, X, Y):

    m = np.shape(X)[0]

    H = sigmoide(np.matmul(X, np.transpose(theta)))

    l1 = np.transpose(np.log(H))

    l2 = np.transpose(np.log(1 - H))

    return (-1 / m) \* ((np.matmul(l1, Y)) + (np.matmul(l2, (1 - Y))))

def gradiente(theta, X, Y):

    H = sigmoide(np.dot(X, np.transpose(theta)))

    grad = (1 / np.shape(X)[0]) \* np.matmul(np.transpose(X), H - Y)

    return grad

def pinta\_frontera\_recta(X, Y, theta):

    x1\_min, x1\_max = X[:, 0].min(), X[:, 0].max()

    x2\_min, x2\_max = X[:, 1].min(), X[:, 1].max()

    xx1, xx2 = np.meshgrid(np.linspace(x1\_min, x1\_max),

    np.linspace(x2\_min, x2\_max))

    h = sigmoide(np.c\_[np.ones((xx1.ravel().shape[0], 1)), xx1.ravel(), xx2.ravel()].dot(theta))

    h = h.reshape(xx1.shape)

    plt.contour(xx1, xx2, h, [0.5], linewidths=2, colors='green')

def porcentaje\_aciertos(X, Y, theta):

    aciertos = 0

    j = 0

    for i in X:

        pred = sigmoide(np.dot(i, theta))

        if pred >= 0.5 and Y[j] == 1:

            aciertos += 1

        elif pred < 0.5 and Y[j] == 0:

            aciertos += 1

        j += 1

    return aciertos / len(Y) \* 100

if \_\_name\_\_ == '\_\_main\_\_':

    datos = load\_data('./ex2data1.csv')

    X = datos[:, :-1]

    Y = datos[:, -1]

    print\_data(X, Y)

    X = np.hstack([np.ones([np.shape(X)[0], 1]), X])

    m = np.shape(X)[0]

    n = np.shape(X)[1]

    theta = np.zeros(n)

    print('1.3. Calculo: ', str(coste(theta, X, Y)), str(gradiente(theta, X, Y)))

    nX = datos[:, :-1]

    result = opt.fmin\_tnc(func=coste, x0=theta, fprime=gradiente, args=(X, Y))

    print('Resultado opt.fmin\_tnc: ' + str(result))

    theta\_opt = result[0]

    print('Coste final: ' + str(coste(theta\_opt, X, Y)))

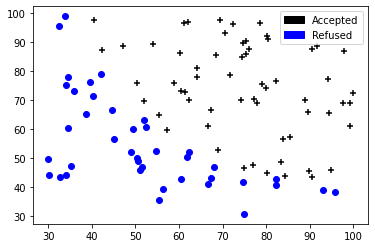
    pinta\_frontera\_recta(nX, Y, theta\_opt)

    print("Prediccion con un porcentaje de aciertos de:", porcentaje\_aciertos(X, Y, theta\_opt))

    plt.show()

* Resultados de ejecución: parte 1 (datos linealmente separables)

Para comprobar que los datos son linealmente separables, se ha generado la siguiente gráfica sobre el primer conjunto de datos.



Si iniciamos a 0 los elementos theta, se puede comprobar que obtenemos el valor de la función de coste en torno a 0,693 y un gradiente de [ -0.1 -12.0092 -11.2628].

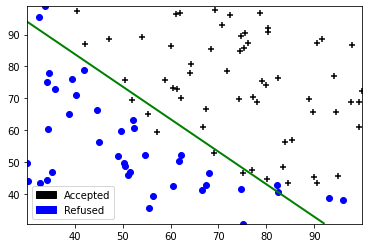


Para obtener el valor de los parámetros theta que minimizan la función de coste, hacemos uso de la función scipy.optimize.fmin\_tnc de SciPy.

Obtenemos los siguiente valores de coste óptimo y valores del gradiente:



Gracias a esto, podemos pintar la frontera de decisión (descomentando la llamada a pinta\_frontera\_recta en el main):



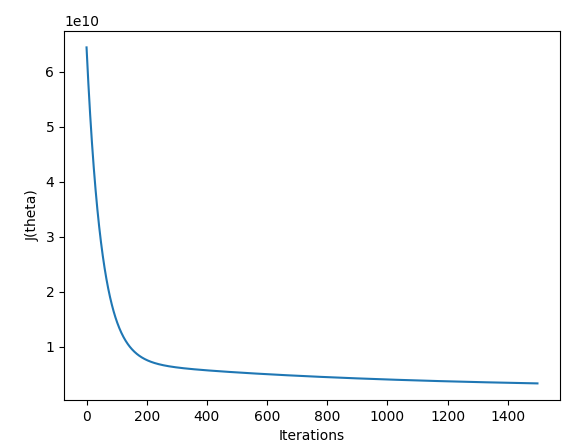
Por último, hemos implementado una función que interpreta los resultados obtenidos de nuestro modelo. Si el resultado de la función sigmonide, utilizando el vector theta calculado, es mayor a 0,5, el alumno se considera admitido, mientras que si es menor, será rechazado. Si este resultado coincide con el resultado real, consideraremos un acierto de nuestro modelo, y obtenemos así un porcentaje de aciertos:



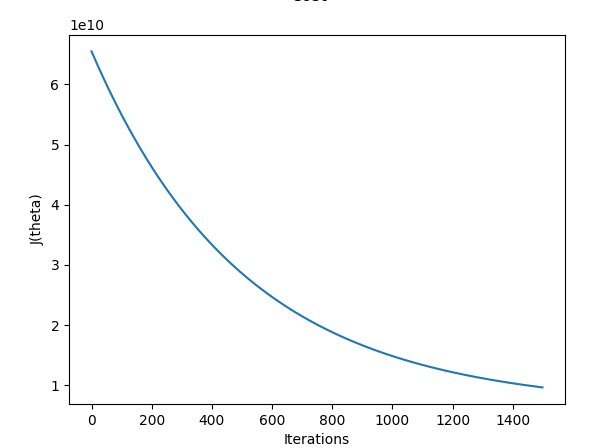
* Código de la práctica: parte 2 (varias variables)
* def getMat(file\_name):  
   return read\_csv(file\_name, header=None).to\_numpy().astype(float)  
    
    
  def gradient(X, Y, T, alpha):  
   new\_T = T  
   m = np.shape(X)[0]  
   n = np.shape(X)[1]  
   H = np.dot(X, T)  
   aux = (H - Y)  
   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()
* Resultados de ejecución: parte 2 (varias variables)

Se ha generado la gráfica del coste teniendo en cuenta la tasa de aprendizaje y 1500 iteraciones.

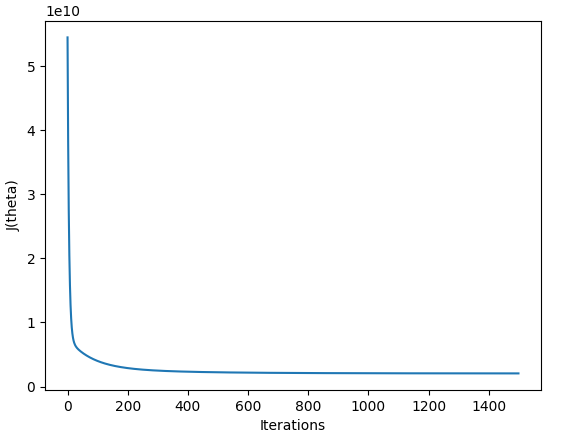
Esta gráfica muestra el coste J(theta) con una tasa de aprendizaje de 0,01.

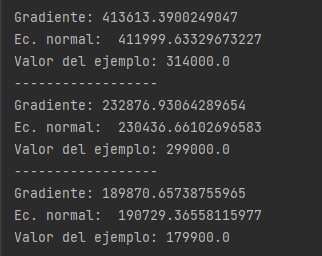


Si reducimos esa tasa de aprendizaje (por ejemplo, a 0,001), podemos ver que el coste no llega a converger en nuestras 1500 iteraciones, ya que da saltos muy pequeños



Si aumentamos la tasa (por ejemplo, a 0,1), obtenemos convergencia muy rápido, pero los valores se distancian, en ocasiones, bastante de los valores reales de los ejemplos de entrenamiento.





Estos valores mostrados, se han obtenido para todos los ejemplos del csv. Se comprueba que valor predicen nuestros dos algoritmos (tanto el implementado con el descenso gradiente, como el implementado mediante la ecuación normal). A continuación, se muestran los resultados para todos los valores del csv con 1500 iteraciones y alpha= 0,01.

Predicciones del CSV usando grandiente y la ecuacion normal:

Gradiente: 343568.70614447986

Ec. normal: 356283.11033889925

Valor del ejemplo: 399900.0

------------------

Gradiente: 311159.49006306095

Ec. normal: 286120.93063401605

Valor del ejemplo: 329900.0

------------------

Gradiente: 362602.6901922973

Ec. normal: 397489.4698481164

Valor del ejemplo: 369000.0

------------------

Gradiente: 278837.3668245447

Ec. normal: 269244.1857271005

Valor del ejemplo: 232000.0

------------------

Gradiente: 421675.2774980165

Ec. normal: 472277.8551463641

Valor del ejemplo: 539900.0

------------------

Gradiente: 356406.71733404783

Ec. normal: 330979.02101847425

Valor del ejemplo: 299900.0

------------------

Gradiente: 306915.4260523989

Ec. normal: 276933.0261488528

Valor del ejemplo: 314900.0

------------------

Gradiente: 300034.8980351136

Ec. normal: 262037.4840289668

Valor del ejemplo: 198999.0

------------------

Gradiente: 297012.6100275209

Ec. normal: 255494.58235013846

Valor del ejemplo: 212000.0

------------------

Gradiente: 304343.2660459371

Ec. normal: 271364.5991881477

Valor del ejemplo: 242500.0

------------------

Gradiente: 353513.0373267783

Ec. normal: 324714.54068768106

Valor del ejemplo: 239999.0

------------------

Gradiente: 336881.0901276791

Ec. normal: 341805.2002410662

Valor del ejemplo: 347000.0

------------------

Gradiente: 329807.6501099091

Ec. normal: 326492.0260991274

Valor del ejemplo: 329999.0

------------------

Gradiente: 537206.7769455726

Ec. normal: 669293.2122320869

Valor del ejemplo: 699900.0

------------------

Gradiente: 289810.5620094278

Ec. normal: 239902.98686016432

Valor del ejemplo: 259900.0

------------------

Gradiente: 376662.47738493467

Ec. normal: 374830.38333402626

Valor del ejemplo: 449900.0

------------------

Gradiente: 272664.1828090363

Ec. normal: 255879.9610214085

Valor del ejemplo: 299900.0

------------------

Gradiente: 287752.8340042584

Ec. normal: 235448.24529160035

Valor del ejemplo: 199900.0

------------------

Gradiente: 396532.4134348522

Ec. normal: 417846.4816054725

Valor del ejemplo: 499998.0

------------------

Gradiente: 423668.7015030244

Ec. normal: 476593.3860409105

Valor del ejemplo: 599000.0

------------------

Gradiente: 321898.258090039

Ec. normal: 309369.1131949595

Valor del ejemplo: 252900.0

------------------

Gradiente: 309188.8549007941

Ec. normal: 334951.62386341975

Valor del ejemplo: 255000.0

------------------

Gradiente: 311416.7060637071

Ec. normal: 286677.7733300865

Valor del ejemplo: 242900.0

------------------

Gradiente: 354927.7253303323

Ec. normal: 327777.1755160688

Valor del ejemplo: 259900.0

------------------

Gradiente: 458415.65043300006

Ec. normal: 604913.3741343784

Valor del ejemplo: 573900.0

------------------

Gradiente: 279007.4899822882

Ec. normal: 216515.5936252033

Valor del ejemplo: 249900.0

------------------

Gradiente: 302028.3220401215

Ec. normal: 266353.01492351317

Valor del ejemplo: 464500.0

------------------

Gradiente: 370704.99421265203

Ec. normal: 415030.01477433724

Valor del ejemplo: 469000.0

------------------

Gradiente: 349741.8901599882

Ec. normal: 369647.33504459134

Valor del ejemplo: 475000.0

------------------

Gradiente: 377842.73823058355

Ec. normal: 430482.39959029364

Valor del ejemplo: 299900.0

------------------

Gradiente: 306037.9588928784

Ec. normal: 328130.3008365561

Valor del ejemplo: 349900.0

------------------

Gradiente: 231596.71554854987

Ec. normal: 220070.56444809595

Valor del ejemplo: 169900.0

------------------

Gradiente: 359943.43734293286

Ec. normal: 338635.60808944365

Valor del ejemplo: 314900.0

------------------

Gradiente: 409994.7383113563

Ec. normal: 500087.73659910634

Valor del ejemplo: 579900.0

------------------

Gradiente: 345217.82130593894

Ec. normal: 306756.3637394074

Valor del ejemplo: 285900.0

------------------

Gradiente: 300677.938036729

Ec. normal: 263429.5907691431

Valor del ejemplo: 249900.0

------------------

Gradiente: 287945.74600474304

Ec. normal: 235865.87731365324

Valor del ejemplo: 229900.0

------------------

Gradiente: 365859.405357795

Ec. normal: 351442.9900990652

Valor del ejemplo: 345000.0

------------------

Gradiente: 499804.6376942942

Ec. normal: 641418.8240777791

Valor del ejemplo: 549000.0

------------------

Gradiente: 367788.5253626414

Ec. normal: 355619.31031959393

Valor del ejemplo: 287000.0

------------------

Gradiente: 294784.75886460795

Ec. normal: 303768.43288347166

Valor del ejemplo: 368500.0

------------------

Gradiente: 352185.4421661269

Ec. normal: 374937.3406572611

Valor del ejemplo: 329900.0

------------------

Gradiente: 393831.6454280673

Ec. normal: 411999.63329673227

Valor del ejemplo: 314000.0

------------------

Gradiente: 285437.8899984427

Ec. normal: 230436.66102696583

Valor del ejemplo: 299000.0

------------------

Gradiente: 242569.91073343306

Ec. normal: 190729.36558115977

Valor del ejemplo: 179900.0

------------------

Gradiente: 347854.2853125623

Ec. normal: 312464.00137413

Valor del ejemplo: 299900.0

------------------

Gradiente: 285630.8019989274

Ec. normal: 230854.2930490187

Valor del ejemplo: 239500.0

------------------

Para finalizar, se ha hecho la comparación entre el gradiente y la ecuación normal con el ejemplo concreto de 1500 pies cuadrados y 3 habitaciones.

Casa con una superficie de 1.650 pies cuadrados y 3 habitaciones:

Ec.normal: 293081.4643348973

Gradiente: 314374.6900711382