Universidad Complutense de Madrid

IAAC - PRÁCTICA 2

Yaco Alejandro Santiago Pérez

Asignatura: INTELIGENCIA ARTIFICIAL APLICADA A INTERNET DE LAS COSAS

P2: Regresión logística

Master IOT

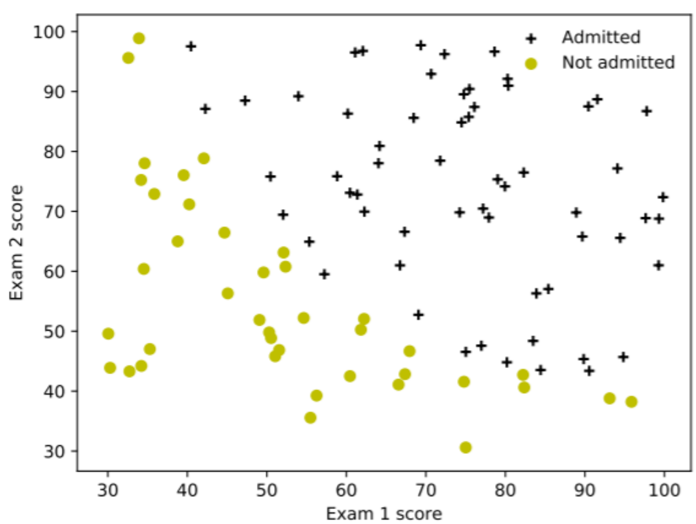
9 de marzo de 2020

Índice general

Capítulo 1

Introducción

Esta práctica trata de emplear la ***Regresión logística*** sobre los datos que representan las notas obtenidas por una serie de candidatos en los dos exámenes de admisión de una universidad junto con la información sobre si fueron admitidos o no.



Capítulo 2

Objetivos

En esta práctica, la cual se divide en dos partes, el objetivo es construir **modelos por regresión logística**.

- Calcular la regresión logística
- Calcular la regresión logística regularizada

Para ello, deberemos alcanzar los siguientes objetivos más concretos:

- Calcular el coste
- Calcular el *gradiente*
- Calcular la *theta óptima*
- *Evaluar* los resultados obtenidos
- Generar las *gráficas* con la *frontera* que divide los puntos.

Capítulo 3

Parte 1: Regresión logística

3.1. Funciones

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

- **sigmoid(x):** Calcula el valor de la función **sigmoide**:

$$g(z) = \frac{1}{1+e^{-z}}$$

- **pinta_frontera_recta(X, Y, theta):** Función que pinta la frontera que divide los puntos en *Admitidos* y *No admitidos*, en base a la *theta óptima* obtenida.

- **cost(theta, X, Y):** Función que calcula el coste para una Theta concreta. Implementa la siguiente fórmula:

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

- **gradient(theta, XX, Y):** Función que calcula el gradiente mediante la siguiente fórmula:

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

- **pinta_puntos(X,Y):** Función que sitúa en el plano los puntos y los colorea en función de si son admitidos o no.

- **evaluaPorcentaje(X,Y,Theta):** Función que calcula el porcentaje de ejemplos de entrenamiento que se clasifican correctamente utilizando el vector *Theta* para calcular el valor de la función **sigmoide** sobre cada ejemplo de entrenamiento. Interpreta que si el resultado es mayor o igual a 0,5 entonces el alumno será admitido (1) y si es menor no lo será (0).

- **main():**Función que hace todas las llamadas pertinentes.
  Carga los datos, inicializa la theta a 0s.
  calcula las thethas y el coste de manera óptima con la llamada a *opt.fmin_tnc*.
  A continuación pinta la recta y evalúa las predicciones correctas.

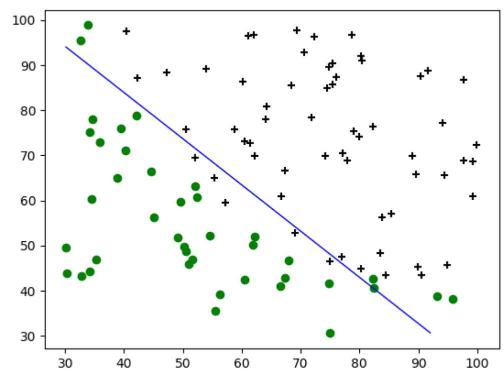
3.2. Ejecución

En la consola se ven los valores obtenidos en la ejecución:

```
  NIT   NF   F                       GTG
    0    1  6.931471805599452E-01   2.71082898E+02
    1    3  6.318123602631195E-01   7.89087138E-01
    2    5  5.892425226259743E-01   7.39226552E+01
    3    7  4.227824087032675E-01   1.85265830E+01
    4    9  4.072926957270926E-01   1.68671132E+01
    5   11  3.818854900460816E-01   1.07735087E+01
    6   13  3.786234863950825E-01   2.31584932E+01
tnc: stepmx = 1000
    7   16  2.389268224905230E-01   3.00821934E+00
    8   18  2.047203887730008E-01   1.52227476E-01
    9   20  2.046713896742690E-01   6.62494850E-02
   10   22  2.035303163816719E-01   9.30780205E-04
tnc: fscale = 32.7775
   11   24  2.035293522731225E-01   8.07222080E-06
   12   26  2.035251114966092E-01   1.80210494E-04
   13   28  2.034984108349350E-01   5.02860966E-04
   14   30  2.034978381620867E-01   9.91430416E-06
   15   32  2.034977907129648E-01   3.77634440E-06
   16   34  2.034977388203336E-01   1.94627679E-05
   17   36  2.034977015894746E-01   2.34303181E-13
tnc: |pg| = 1.47677e-08 -> local minimum
   17   36  2.034977015894746E-01   2.34303181E-13
tnc: Local minima reach (|pg| ~= 0)
Result :(array([-25.16131863,   0.20623159,   0.20147149]), 36, ↩
   0)
Hay un 60.0 % de aciertos
```

Hay un 60 % de aciertos

Se muestra y guarda la **gráfica** con la linea de frontera entre puntos:



3.3. Código

A continuación presento el código de la parte 1:

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

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

def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s

def pinta_frontera_recta(X, Y, theta):
    pinta_puntos(X,Y)
    x1_min, x1_max = X[:,1].min(), X[:,1].max()
    x2_min, x2_max = X[:, 2].min(), X[:, 2].max()

    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
    np.linspace(x2_min, x2_max))

    h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)),
    xx1.ravel(),
    xx2.ravel()].dot(theta))
    h = h.reshape(xx1.shape)

```

```
    # el cuarto par metro es el valor de z cuya frontera se quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    plt.savefig("frontera.png")
    plt.show()

def cost(theta, X, Y):
    H = sigmoid(np.matmul(X, theta))
    cost = (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1 - H)))
    return cost

def gradient(theta, XX, Y):
    H = sigmoid( np.matmul(XX, theta) )
    grad = (1 / len(Y)) * np.matmul(XX.T, H - Y)
    return grad

def pinta_puntos(X,Y):
    plt.figure()
    mark='o'
    cc='g'
    i=0
    for i in range(2):
        pos= np.where(Y== i)
        if i==1:
            mark='+'
            cc='k'
        plt.scatter(X[pos, 1], X[pos,2], marker=mark, c=cc)

def evaluaPorcentaje(X,Y,Theta):
    cont = 0
    m = len(X)
    prediccion =1 / (1 + np.exp(-np.dot(Theta, X.T)))
    for i in range(m):
        if (prediccion.T[i] >= 0.5 and Y[i] == 1) or (prediccion.T[i] < 0.5 and Y[i] == 0):
            cont += 1
    print("Hay un "+ str((cont/m)*100) + "% de aciertos")

def main():
    datos = carga_csv('ex2data1.csv')
    X = datos[:, :-1]
    np.shape(X)
    Y = datos[:, -1]
    np.shape(Y)
```

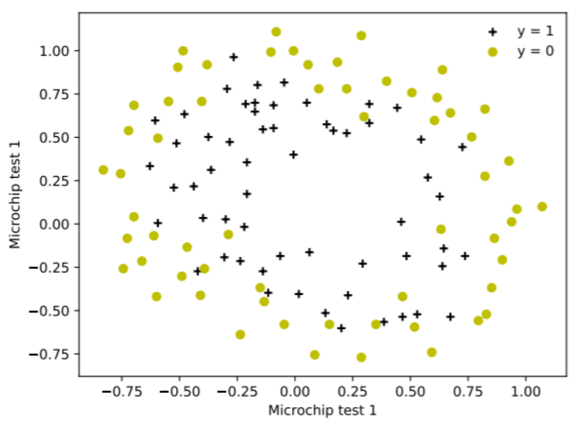
```
    m = np.shape(X)[0]
    X = np.hstack([np.ones([m, 1]), X])

    initialTheta = np.zeros(3)

    result = opt.fmin_tnc(func=cost , x0=initialTheta , fprime=
       gradient, args =(X, Y))

    print("Result␣:"+str(result))

    pinta_frontera_recta(X,Y,result[0])
    evaluaPorcentaje(X,Y,initialTheta)
main()
```

Código parte 1: p2.py

Capítulo 4

Parte 2: Regresión logística regularizada

En esta parte se usará la regresión logística regularizada para encontrar una función que pueda predecir si un microchip pasará o no el control de calidad, a partir del resultado de dos tests a los que se somete a los microchips.



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:

$$J(\theta) = -\frac{1}{m}((\,log\,(g(X\theta)))^T y + (\,log\,(1 - g(X\theta)))^T (1 - y)) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

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

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} X^T (g(X\theta) - y) + \frac{\lambda}{m}\theta_j$$

- **costeMinimo(XX, Y, lam):** Inicializa las thetas a 0 y realiza la llamada a *opt.fmin_tnc* para un *lamda* concreto. Para obtener el valor óptimo de *theta* para la versión regularizada de la función de coste.

- **plot_decisionboundary(X, Y, theta, poly,lam):** Pinta la delimitación polinómica que separa los puntos.

- **main():** El *main* ha sufrido sustanciosas modificaciones.

  Ahora se calcula para diferentes valores de **lamda** *(1,0.1, 0.3, 0.01, 0.03, 0.001, 0.003, 0.000003, 50, 100, 500)* con el fin de mostrar los diferentes resultados.
  Para cada uno de estos valores se realiza la llamada a *costeMinimo* y con las *thetas* mínimas para dicha *lamda* se pintará la delimitación en una gráfica.

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:

```
[LAMDA: 1]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
    1    6  5.360727445969196E-01   9.47467765E-04
tnc: fscale = 32.4876
    2   11  5.291073407934827E-01   1.47916266E-05
    3   14  5.290104042369796E-01   3.98120393E-07
tnc: fscale = 1584.87
    4   19  5.290029439251899E-01   6.89672366E-09
    5   22  5.290028141673242E-01   3.25682617E-09
    6   27  5.290027426717190E-01   1.38854119E-09
tnc: |fn-fn-1] = 1.27072e-08 -> convergence
    7   32  5.290027299645007E-01   1.17067354E-11
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 0.1]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
    1    6  4.460802252725170E-01   2.43734953E-03
tnc: fscale = 20.2554
```

```
    2   12  4.002455356516611E-01   2.12258361E-04
    3   15  3.959634583728573E-01   2.59598663E-05
    4   21  3.946455674765940E-01   2.51875971E-07
tnc: fscale = 1992.54
    5   28  3.945971489165538E-01   1.82972314E-08
    6   31  3.945950455035792E-01   2.66262220E-08
    7   38  3.945941649439164E-01   6.12750815E-10
tnc: fscale = 40397.8
    8   46  3.945941390164711E-01   1.32120523E-12
tnc: |pg| = 2.84529e-11 -> local minimum
    8   46  3.945941390164711E-01   1.32120523E-12
tnc: Local minima reach (|pg| ~= 0)
[LAMDA: 0.3]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
    1    6  4.768698284655525E-01   1.90974260E-03
tnc: fscale = 22.883
    2   12  4.521102159673544E-01   9.62030682E-05
    3   15  4.510638266940439E-01   3.56249508E-06
tnc: fscale = 529.813
    4   21  4.509200095639659E-01   6.44888994E-08
    5   25  4.509188638519904E-01   2.76269162E-09
tnc: fscale = 19025.4
    6   31  4.509187918179109E-01   7.34149385E-12
tnc: |fn-fn-1] = 2.78971e-10 -> convergence
    7   35  4.509187915389401E-01   4.82730376E-12
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 0.01]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
tnc: stepmx = 1000
    1    7  3.808092050775161E-01   2.78317611E-03
    2   11  3.621590995867802E-01   1.39837712E-04
tnc: fscale = 84.5645
    3   20  3.380549296545360E-01   1.35117252E-05
    4   24  3.350484681480979E-01   1.58373286E-05
    5   33  3.329935455222823E-01   7.09159208E-07
    6   42  3.327070097785043E-01   6.20000614E-08
tnc: fscale = 4016.09
    7   51  3.326656423020788E-01   3.67504259E-07
    8   57  3.326570010020000E-01   7.81151090E-09
    9   68  3.326527895862872E-01   7.23870265E-10
   10   77  3.326525945677236E-01   1.33739268E-10
tnc: fscale = 86471
   11   86  3.326525495756134E-01   8.31487796E-11
```

```
tnc: |fn-fn-1] = 3.76748e-09 -> convergence
   12   90  3.326525458081364E-01   2.33901143E-11
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 0.03]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
tnc: stepmx = 1000
    1    7  3.885758397799475E-01   2.71157582E-03
    2   11  3.709304142654307E-01   1.06970689E-04
tnc: fscale = 96.6869
    3   20  3.557962755954324E-01   7.06421431E-06
    4   23  3.553318206011966E-01   7.24754738E-06
    5   27  3.549685715428941E-01   2.84875634E-06
    6   33  3.547662139100415E-01   3.14484053E-07
    7   42  3.547254133742747E-01   1.19655924E-08
tnc: fscale = 9141.82
    8   50  3.547209965219245E-01   1.80852616E-09
    9   59  3.547204230530154E-01   4.95940899E-11
   10   64  3.547203994936236E-01   1.56740488E-10
tnc: |fn-fn-1] = 7.82068e-09 -> convergence
   11   73  3.547203916729397E-01   3.20705017E-12
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 0.001]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
tnc: stepmx = 1000
    1    7  3.773182883330984E-01   2.81740382E-03
    2   11  3.580793324112773E-01   1.59152947E-04
tnc: fscale = 79.267
    3   20  3.278068095289328E-01   2.27413929E-05
    4   23  3.235357235334811E-01   2.54513114E-05
    5   34  3.134674827109585E-01   3.91161326E-05
    6   44  3.083671223959456E-01   5.98188751E-06
    7   47  3.078876541622895E-01   5.47992688E-06
    8   57  3.061912274935260E-01   8.86081442E-07
    9   66  3.057162742615221E-01   9.42629639E-08
tnc: fscale = 3257.09
   10   72  3.056535593755805E-01   1.28723848E-07
   11   81  3.054482526725112E-01   4.36903951E-08
   12   84  3.054394083822150E-01   2.48859953E-07
   13   96  3.053148036393047E-01   2.42998221E-07
   14  101  3.052975499135698E-01   3.46627652E-08
   15  115  3.052393419021551E-01   2.11077599E-08
   16  121  3.052297255037367E-01   9.20752746E-09
   17  127  3.052202826434248E-01   9.90121353E-09
```

```
   18  132  3.052183574341756E-01   3.45034497E-09
   19  143  3.052134119757501E-01   2.07683826E-08
   20  150  3.052127587412990E-01   2.58824942E-09
   21  164  3.052114979607330E-01   9.76559360E-10
   22  176  3.052106829430526E-01   2.44570469E-10
   23  187  3.052104092809213E-01   8.13648605E-10
   24  202  3.052100208634540E-01   3.56773088E-10
   25  205  3.052099944416152E-01   7.28525659E-11
tnc: fscale = 117160
   26  214  3.052099411046195E-01   1.31291294E-11
   27  220  3.052099213619388E-01   1.42479300E-10
tnc: |fn-fn-1] = 1.29314e-08 -> convergence
   28  226  3.052099084305813E-01   7.66883136E-12
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 0.003]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
tnc: stepmx = 1000
    1    7  3.780946756643514E-01   2.80997872E-03
    2   11  3.589938630869162E-01   1.54591615E-04
tnc: fscale = 80.428
    3   20  3.302338208646606E-01   2.06588489E-05
    4   23  3.262908845301375E-01   4.57773804E-05
    5   33  3.193361205229668E-01   1.26384584E-05
    6   43  3.171917274062098E-01   1.36566882E-06
    7   46  3.170893050015301E-01   1.27005044E-06
    8   57  3.166755010820262E-01   1.53611038E-07
tnc: fscale = 2551.46
    9   66  3.166220360352694E-01   4.30932063E-08
   10   78  3.165657268445338E-01   5.65025458E-09
   11   81  3.165648955327059E-01   9.17762974E-09
   12   93  3.165566910612760E-01   4.32008774E-09
   13   98  3.165561380900948E-01   2.57455375E-09
   14  107  3.165556673301859E-01   5.97543279E-11
tnc: fscale = 129365
   15  116  3.165555368853910E-01   3.21290099E-10
   16  120  3.165555001615299E-01   1.66763298E-10
   17  130  3.165554726767813E-01   8.74165089E-11
tnc: |fn-fn-1] = 8.01497e-09 -> convergence
   18  140  3.165554646618129E-01   6.86651153E-12
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 3e-06]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
tnc: stepmx = 1000
```

```
    1    7  3.769324091931708E-01   2.82155957E-03
    2   11  3.576218282801458E-01   1.61492730E-04
tnc: fscale = 78.6907
    3   20  3.265788747498314E-01   2.39425764E-05
    4   23  3.220134031508985E-01   2.01225107E-05
    5   34  3.099257316943960E-01   3.36801963E-05
    6   50  2.985641445729089E-01   1.29283040E-04
    7   56  2.959774783363324E-01   2.97455406E-06
    8   68  2.899463466396988E-01   7.18710443E-06
    9   82  2.827091152940611E-01   3.54805884E-06
   10   87  2.821028189842305E-01   2.43942346E-06
   11   99  2.801600204212333E-01   2.35148582E-06
   12  114  2.765891245753863E-01   2.04019518E-05
   13  129  2.741698585234981E-01   7.02485743E-07
   14  144  2.732366164010770E-01   5.87967492E-06
   15  147  2.731626840510445E-01   5.61749733E-07
   16  152  2.729976540084674E-01   1.39748599E-07
tnc: fscale = 2675.02
   17  161  2.728699995829900E-01   1.11016429E-07
   18  175  2.718795892330613E-01   5.11828600E-08
   19  190  2.712743259801015E-01   1.64626914E-06
   20  202  2.709120346761067E-01   6.96231498E-07
   21  217  2.700484657194910E-01   2.03762904E-06
   22  229  2.681603589711120E-01   3.73007915E-07
   23  243  2.674500728970605E-01   1.40226169E-07
   24  255  2.672170862029215E-01   6.12496856E-08
   25  267  2.670829114925242E-01   2.69668297E-08
   26  280  2.669953788023428E-01   1.16453767E-07
   26  280  2.669953788023428E-01   1.16453767E-07
tnc: Maximum number of function evaluations reached
[LAMDA: 50]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
    1    4  6.809073717564785E-01   1.14566048E-04
tnc: fscale = 93.4269
    2    7  6.807252630700570E-01   2.95402549E-06
    3   10  6.807224367461087E-01   2.87595610E-07
tnc: fscale = 1864.7
    4   13  6.807221041554352E-01   3.49362391E-09
tnc: |fn-fn-1] = 5.07834e-09 -> convergence
    5   20  6.807220990770997E-01   4.73535234E-10
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 100]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
```

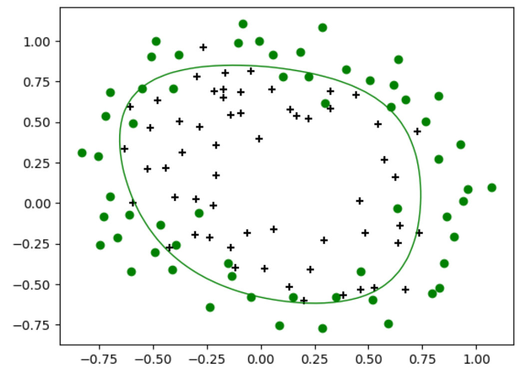
```
    1    4  6.865362017685083E-01   3.38580563E-05
tnc: fscale = 171.858
    2    8  6.864838347936352E-01   1.72447636E-09
tnc: fscale = 24080.8
tnc: |fn-fn-1] = 9.20933e-10 -> convergence
    3   11  6.864838338727018E-01   6.16448700E-14
tnc: Converged (|f_n-f_(n-1)| ~= 0)
[LAMDA: 500]-------------
  NIT   NF   F                       GTG
    0    1  6.931471805599454E-01   1.28006529E-02
    1    4  6.916580330400698E-01   8.99225285E-05
tnc: fscale = 105.455
    2    8  6.916270408215286E-01   1.90934582E-10
tnc: fscale = 72369.9
tnc: |fn-fn-1] = 2.23358e-11 -> convergence
    3   11  6.916270407991927E-01   2.22277750E-13
tnc: Converged (|f_n-f_(n-1)| ~= 0)
```

Como se puede observar en las siguientes gráficas, en función del valor de *lamda* se va a tener un **mayor ajuste** a los valores de entrenamiento.

Como decía el enunciado, inicializando el vector *theta* con ceros y *lamda* a 1 el coste inicial debería ser de **0,693** aproximadamente:

4.2.1. Gráficas de coste para cada Lamda

Para *lamda* **1** se obtiene la siguiente gráfica:



Ahora, voy a presentar las diferentes gráficas en función de *lamda*, para demostrar que a medida que se disminuye la precisión es mayor.

Al elegir una *lamda* muy pequeña corremos el riesgo de que se adapte perfectamente a los casos de test, pero que falle en las futuras pruebas.
Al elegir una muy grande, se corre el riesgo contrario, que se generalice tanto que se de lugar a interpretaciones incorrectas.

Las gráficas son las siguientes:

A) Lamda valor 0.1

B) Lamda valor 0.3

C) Lamda valor 0.01

D) Lamda valor 0.03

E) Lamda valor 0.001

F) Lamda valor 0.003

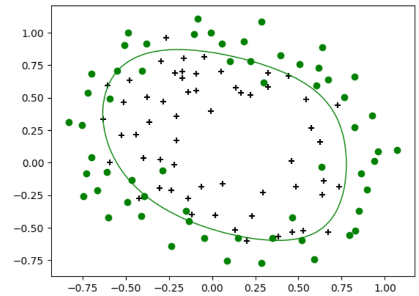
G) Lamda valor 0.000003

H) Lamda valor 50

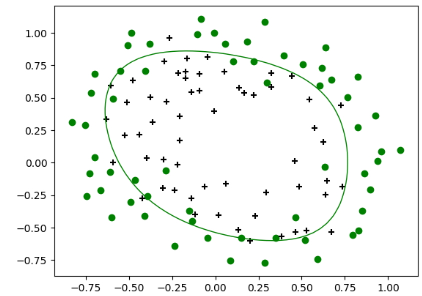
I) Lamda valor 100

J) Lamda valor 500

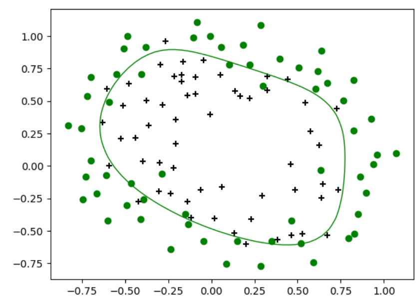
a)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

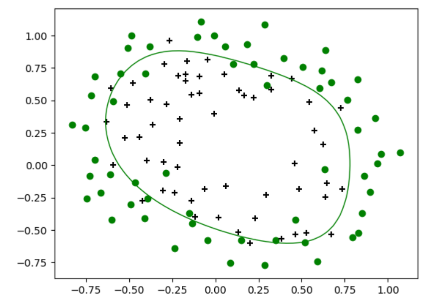
b)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

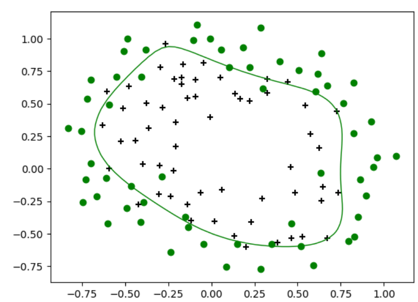
c)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

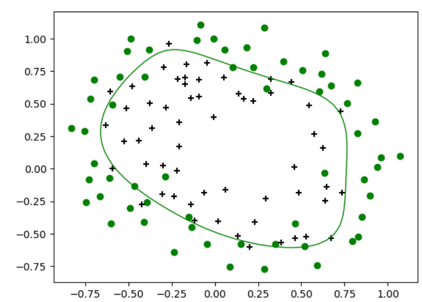
d)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

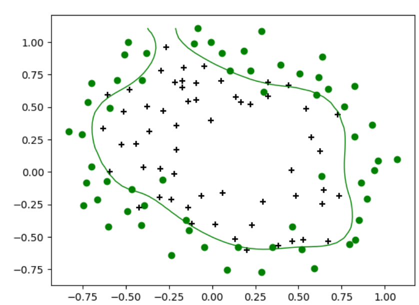
e)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

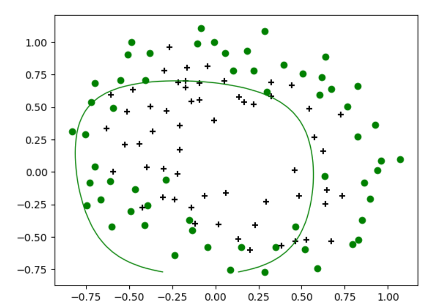
f)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

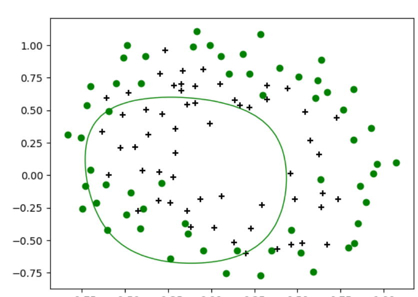
g)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

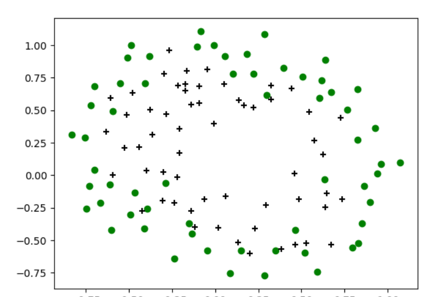
h)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

i)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

j)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

4.3. Código

```
import numpy as np
import copy
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
from sklearn.preprocessing import PolynomialFeatures


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

def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s

def cost(theta, X, Y, lam):
    m = len(X)
    H = sigmoid(np.dot(theta,X.T))
    part3= (np.sum(np.power(theta[1:], 2))*lam)/(2*m)
    part2= (np.log(1-H)).T*(1-Y)
    part1= (np.log(H)).T*Y

    return -1/m*(np.sum(part1 + part2)) + part3


def gradient(theta, XX, Y, lam):
    H = sigmoid(np.dot(XX, theta))
    thetaNoZeroReg = np.insert(theta[1:], 0, 0)
    gradient =  (np.dot(XX.T, (H - Y)) + lam * thetaNoZeroReg) / len(Y)
    return np.vstack(gradient)


def costeMinimo(XX, Y, lam):
    initialTheta = np.zeros(len(XX[0]))
    result = opt.fmin_tnc(func=cost, x0=initialTheta, fprime=gradient, args=(XX, Y, lam))
    return result[0]


def pinta_puntos(X,Y):
    plt.figure()
```

```python
    mark='o'
    cc='g'
    i=0
    for i in range(2):
        pos= np.where(Y== i)
        if i==1:
            mark='+'
            cc='k'
        plt.scatter(X[pos, 0], X[pos,1], marker=mark, c=cc)


def plot_decisionboundary(X, Y, theta, poly,lam):
    x1_min, x1_max = X[:, 1].min(), X[:, 1].max()
    x2_min, x2_max = X[:, 2].min(), X[:, 2].max()
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),np.linspace(x2_min, x2_max))
    h = sigmoid(poly.fit_transform(np.c_[xx1.ravel(), xx2.ravel()]).dot(theta))
    h = h.reshape(xx1.shape)
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='g')
    plt.savefig("boundary"+str(lam)+".png")


def main():
    datos = carga_csv('ex2data2.csv')
    X = datos[:, :-1]
    np.shape(X)
    Y = datos[:, -1]
    np.shape(Y)

    poly = PolynomialFeatures(6)
    X2 = poly.fit_transform(X)

    lams = [1,0.1, 0.3, 0.01, 0.03, 0.001, 0.003, 0.000003, 50, 100, 500]

    for lam in lams:
        pinta_puntos(X,Y)
        print("[LAMDA: "+str(lam)+"]-------------")
        thetaMin = costeMinimo(X2,Y, lam)
        plot_decisionboundary(X2, Y, thetaMin, poly,lam)

main()
```

Código parte 2: p2.2.py