Practica2AA

April 11, 2021

1 Práctica 2 : Regresión Logística

Guillermo García Patiño Lenza y Mario Quiñones Pérez

[73]:
```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D
import scipy.optimize as opt
```

1.1 Parte 1:

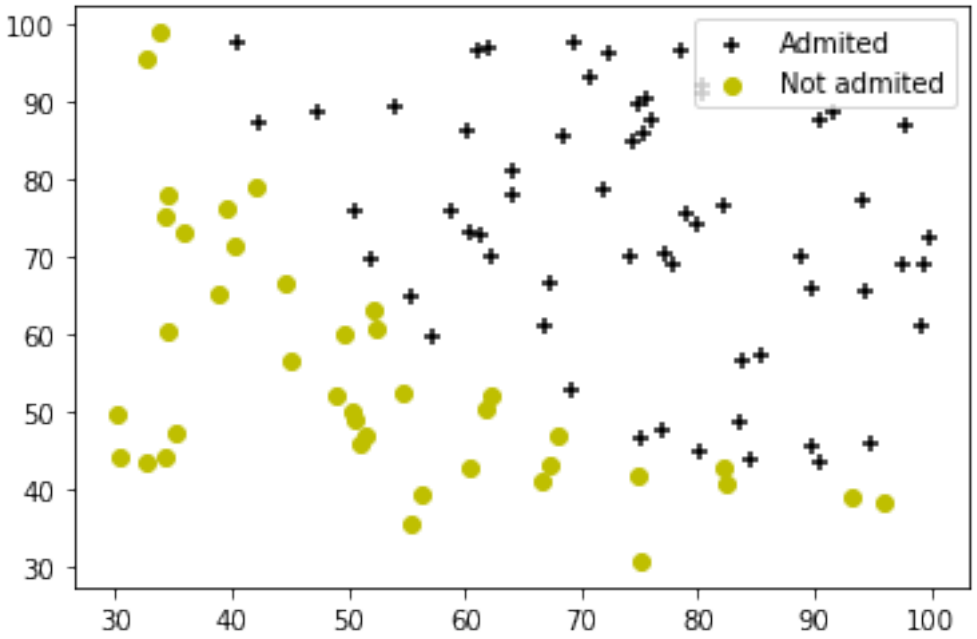
1.1.1 Carga y visualización de los datos

En esta primera parte cargaremos los datos que se encuentran en el fichero "ex2data1.csv" y los visualizaremos diferenciando enttre los elementos de la grafica que representan que un alumno ha sido admitido (y = 1) o no (y = 0)

[74]:
```
def cargaDatos(fichero):
    datos = read_csv(fichero, header = None).to_numpy()
    return datos.astype(float)
```

[6]:
```
def crearGrafica(datos, labels = ['y = 1', 'y = 0']):
    X = datos[:,:-1]
    Y = datos[:,-1]

    c1 = np.where(Y == 1)
    c2 = np.where(Y == 0)

    plt.scatter(X[c1,0], X[c1,1] , marker = '+', c = 'k', label = labels[0])
    plt.scatter(X[c2,0], X[c2,1], marker = 'o', c = 'y', label = labels[1])
    plt.legend(loc = 'upper right')
    plt.show()
```

[7]:
```
crearGrafica(cargaDatos("ex2data1.csv"), ['Admited', 'Not admited'])
```



1.1.2 Funcion Sigmoide

En esta parte crearemos la función sigmoide que se podra aplicar indistintamente a cualquier número de datos y devolvera una cantidad de datos de la misma forma en la que entraron

[8]:
```
def sigmoide(Z):
    sigmoide = 1 / (1 + np.exp(-Z))
    return sigmoide
```

1.1.3 Cálculo de la función de coste y gradiente

Utilizaremos el metodo de descenso de gradiente para calcualr los valores Theta que minimicen el coste ,el cual nos informara de cuantos de los atributos estan bien interpretados segun la recta, cuanto menor el coste mejor representará la recta a los datos dados

[9]:
```
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)
```

[10]:
```
def coste(Theta, X, Y):
    G = sigmoide(np.dot(X, Theta))
    sum1 = np.dot(Y, np.log(G))
    sum2 = np.dot((1-Y), np.log(1 - G))
    return (-1 / X.shape[0]) * (sum1 + sum2)
```

[11]:
```
def gradiente(Theta, X, Y):
    m = X.shape[0]
    G = sigmoide( np.matmul(X,Theta) )
    gradiente  = (1 / len(Y)) * np.matmul(X.T, G - Y)
    return gradiente
```

[12]:
```
def prueba():
    datos = cargaDatos("ex2data1.csv")
    X = datos[:,:-1]
    Y = datos[:,-1]
    n = len(Y)

    X2,mu,sigma = normalizar(X)
    X2 = np.hstack([np.ones([n,1]), X2])

    c = coste(np.zeros(X2.shape[1]), X2, Y)
    gr = gradiente(np.zeros(X2.shape[1]),X2,Y)
    print(gr)
    print(c)
```

[13]:
```
prueba()
```

```
[-0.1        -0.28122914 -0.25098615]
0.6931471805599453
```

1.1.4 Cálculo del valor óptimo de los parámetros

En este apartado utilizaremos la función scipy.optimize.fmin_tnc de SciPy para obtener el valor de los parámetros Theta que minimizan la función de coste para la regresión logística implementada en el apartado anterior. Dicha funcion utilizando las funciones de coste a minimizar, la funcion del gradiente para su calculo y los valores X e Y como argumentos para el calculo de costes y gradiente, devolvera los Thetas que minimizan dicho coste como resultado

[69]:
```
def optimiza(datos = cargaDatos("ex2data1.csv")):
    X = datos[:,:-1]
    Y = datos[:,-1]
    n = len(Y)

    X_n, mu, sigma = normalizar(X)
    X_n = np.hstack([np.ones([n,1]), X_n])
    T = np.zeros(X.shape[1]+1)

    result = opt.fmin_tnc(func = coste, x0 = T, fprime = gradiente, args =␣
 ↪(X_n, Y))

    c_f = coste(result[0], X_n, Y )
    print(c_f)
    print(result[0])
```

```
    return result[0]
```

[70]:
```
def optimizaparagrafica(datos = cargaDatos("ex2data1.csv")):
    X = datos[:,:-1]
    Y = datos[:,-1]
    n = len(Y)

    X = np.hstack([np.ones([n,1]), X])
    m = np.shape(X)[1]
    T = np.zeros([m,1])

    result = opt.fmin_tnc(func=coste, x0=T, fprime = gradiente,args=(X, Y))

    c_f = coste(result[0], X, Y)
    print(c_f)

    theta_opt = result[0]
    return theta_opt
```

[15]:
```
optimiza()
```

```
0.20349771564653243
[1.71787865 3.99150584 3.72363972]
```

[15]:
```
array([1.71787865, 3.99150584, 3.72363972])
```

Pintaremos la recta frontera con dichos thetas optimizados gracias al apartado anteriór y la añadiremos a la grafica con todos los elementos en el archivo original de datos para poder tener una demostración visual de que Theta minimiza dicho coste y separa correctamente ambos tipos de datos

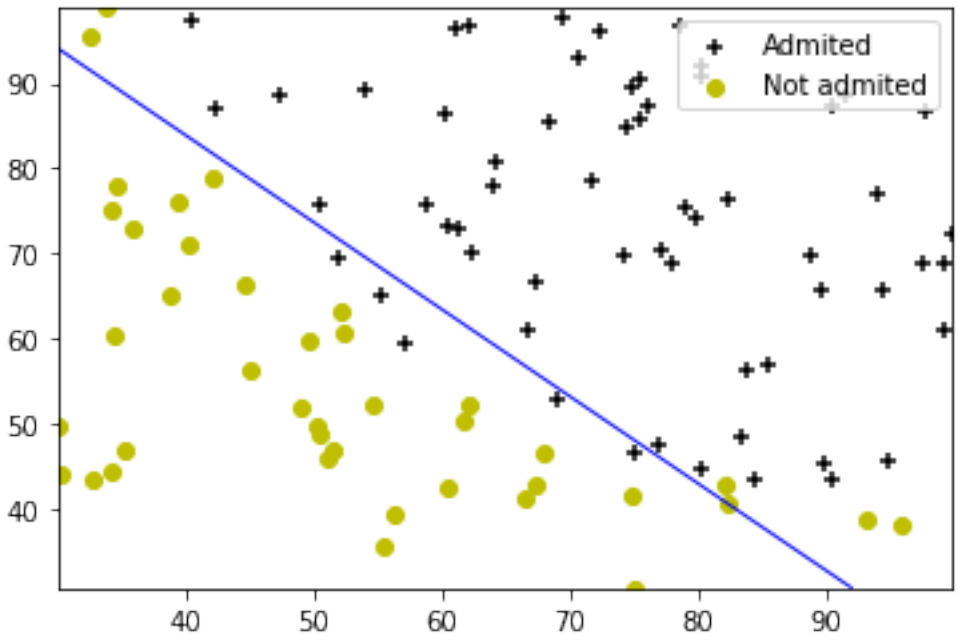
[16]:
```
def frontera(X, Y, theta, plt):
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()

    x1, x2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min,␣
 ↪x2_max))

    h = sigmoide(np.c_[np.ones((x1.ravel().shape[0], 1)), x1.ravel(), x2.
 ↪ravel()].dot(theta))
    h = h.reshape(x1.shape)

    plt.contour(x1, x2, h, [0.5], linewidths=1, colors='b')
```

[17]:
```
def crearGraficaConFrontera(datos, Theta, labels = ['y = 1', 'y = 0']):
    X = datos[:,:-1]
    Y = datos[:,-1]
```

```
    c1 = np.where(Y == 1)
    c2 = np.where(Y == 0)

    plt.scatter(X[c1,0], X[c1,1] , marker = '+', c = 'k', label = labels[0])
    plt.scatter(X[c2,0], X[c2,1], marker = 'o', c = 'y', label = labels[1])
    plt.legend(loc = 'upper right')

    frontera(X, Y, Theta, plt)

    plt.show()
```

[18]:
```
crearGraficaConFrontera(cargaDatos("ex2data1.csv"), optimizaparagrafica(),␣
↪['Admited', 'Not admited'])
```

```
0.20349770158947458
```



1.1.5 Evaluación de la regresión

En este apartado implementamos una función que calcula el porcentaje de ejemplos de entrenamiento que se clasifican correctamente utilizando el vector Theta obtenido en el apartado anterior para calcular el valor de la función sigmoide sobre cada ejemplo de entrenamiento, interpretando que si el resultado es mayor que 0.5 el valor resultante es 1 y si no es 0 llegamos a la conclusión de que se predice correctamente un 48% de las veces el valor correcto.

[19]:
```
def evalua(datos, parte_entrenamiento):
    f = round(len(datos) * parte_entrenamiento / 100)
    datos_ent = datos[:f]
    datos_eva = datos[f:]

    theta = optimiza(datos_ent)

    dat_ev_x = datos_eva[:,:-1]
    dat_ev_y = datos_eva[:,-1]

    dat_ev_x_n, mu, sigma = normalizar(dat_ev_x)
    dat_ev_x_n = np.hstack([np.ones([len(datos_eva),1]), dat_ev_x_n])

    res_eva = np.dot(dat_ev_x_n, theta)
    res_eva_m = []
    for e in map(lambda x : 1 if x >= 0.5 else 0, res_eva):
        res_eva_m.append(e)

    iguales = filter(lambda x : dat_ev_y[res_eva_m.index(x)] == x , res_eva_m)

    i = 0;
    for e in iguales:
        i = i+1

    print("Se ha predicho correctamente el resultado un {}% de las veces".
↪format(i/len(datos_eva) * 100))

    return (i/len(datos_eva)*100)
```

[20]:
```
evalua(cargaDatos("ex2data1.csv"), 75)
```

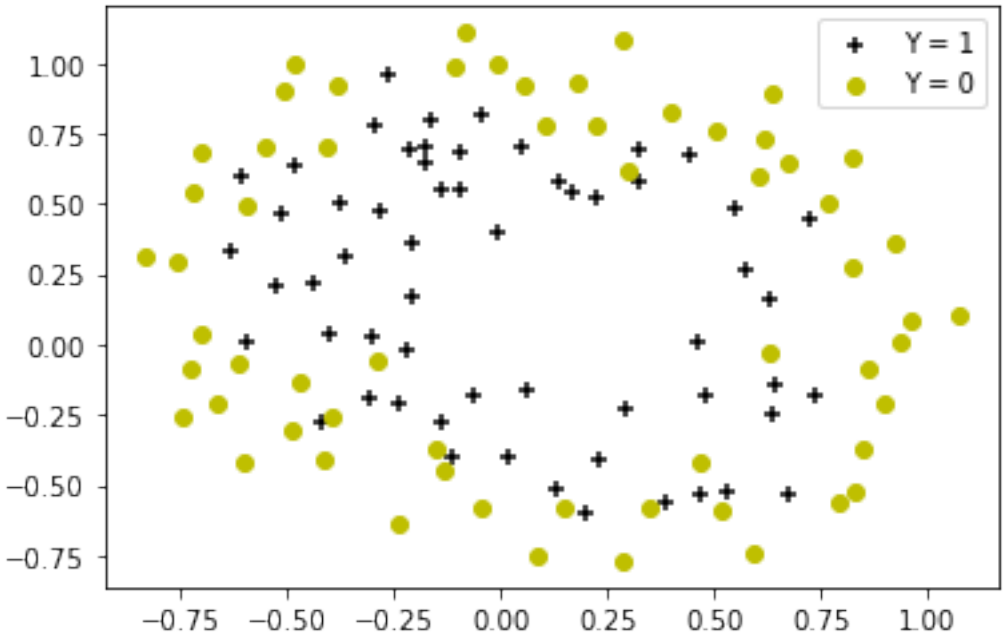
```
0.19456570964464878
[0.63167769 4.16277648 3.60606694]
Se ha predicho correctamente el resultado un 48.0% de las veces
```

[20]:
```
48.0
```

1.2 Parte 2:

1.2.1 Carga y visualización de datos:

[21]:
```
crearGrafica(cargaDatos("ex2data2.csv"), ['Y = 1', 'Y = 0'])
```



1.2.2 Mapeo de los atributos:

Ya que no todos los datos pueden ser separados con una sola recta se necesita otra manera de poder utilizar solo dos datos para poder crear curvas y otras figuras polinomicas, para eso utilizaremos la función PolynomialFeatures(grado).fit_transform(matriz_de_datos) que sirve para convertir una serie de datos en una convinacion de los mismos para conseguir mas variables.

[22]:
```
from sklearn.preprocessing import PolynomialFeatures

def prepara_datos(datos, add):
    X = datos[:,:-1]
    Y = datos[:,-1]

    X_n, mu, sigma = normalizar(X)

    p = PolynomialFeatures(add)
    X2 = p.fit_transform(X_n)

    return (X2,Y,mu,sigma)
```

1.2.3 Cálculo del coste y el gradiente regularizados

Utilizamos la forma regularizada de la versión de regresión logistica de las funciones de coste y de gradiente para poder calcular dichos valores con el polinomio creado que nos dara 28 atributos

[23]:
```
def coste_reg(Theta,X,Y, Lambda):
    c = coste(Theta,X,Y)
    m = X.shape[0]
    e = 0

    for t in range(1,len(Theta)):
        e += Theta[t]**2

    return c + (Lambda/(2*m))*e
```

[24]:
```
def gradiente_reg(Theta,X,Y,Lambda):
    m = X.shape[0]
    gr = gradiente(Theta,X,Y)
    theta2 = (Lambda/m)*Theta
    return (gr + theta2)
```

Hacemos una prueba para el calculo con Theta inicializad a creos y Lambda a 1 y nos da un coste de 0.693 aproxiamdamente lo que es bastante bajo y demustra que funciono el metodo de descenso de gradiente logitico regularizado

[25]:
```
def prueba2():
    datos = cargaDatos("ex2data2.csv")
    X, Y, mu, sigma = prepara_datos(datos, 6)

    Theta = np.zeros(X.shape[1])

    c = coste_reg(Theta,X,Y,1)
    gr = gradiente_reg(Theta,X,Y,1)

    print(gr)
    print(c)
```

[26]:
```
prueba2()
```

```
[ 0.00847458  0.03705198 -0.00284799  0.19753354  0.03181133  0.14157707
  0.08474569 -0.02197759  0.01912472  0.01978675  0.60552761 -0.01178425
  0.17516318 -0.00748334  0.41389202  0.32662613 -0.08620202  0.03911234
 -0.02213139  0.05549666  0.01571     1.78985126 -0.11770386  0.34722535
 -0.07226827  0.2933989  -0.11751531  1.15773589]
0.6931471805599453
```

1.2.4 Cálculo de los valores óptimos:

Creamos una función que nos prepare las funciones para que estas utilicen un Lambda dado y utilizamos denuevo la función scipy.optimize.fmin_tnc para minimizar el coste de dichos valores Theta

[27]:
```
def preparaFunciones(Lambda):
    c = lambda Theta,X,Y : coste_reg(Theta,X,Y,Lambda)
    gr = lambda Theta,X,Y : gradiente_reg(Theta,X,Y,Lambda)

    return (c,gr)
```

[48]:
```
def optimiza_reg(datos,Lambda):
    X, Y, mu, sigma = prepara_datos(datos,6)
    c, gr = preparaFunciones(Lambda)

    T = np.zeros(X.shape[1])

    result = opt.fmin_tnc(func = c, x0 = T, fprime = gr, args = (X, Y))
    c_f = coste(result[0], X, Y)
    print("coste:", c_f)
    return result[0]
```

[68]:
```
datos = cargaDatos("ex2data2.csv")
optimiza_reg(datos, 1)
```

```
coste: 0.31064314472446225
```

[68]:
```
array([ 1.7903008 , -0.20810284,  0.35766677,  0.34424822, -0.36085601,
        0.20176854,  0.16024196,  0.28060681,  0.06676788, -0.30579355,
        0.30662686, -0.00676247, -0.57664467, -0.51383133, -0.19082302,
       -0.05641764, -0.07118331,  0.82605945, -0.34853422, -0.44094965,
       -0.05126368, -0.44397464, -0.1982422 , -0.46715723,  0.01152511,
       -0.87767704, -0.6908056 , -0.33307882])
```

Creamos la grafica que representa los datos dados en el fichero “ex2data2.csv” y añadimos la funcion polinomica dada por Theta que representa la frontera de separación entre los chips que han sido o no aceptados y comprobamos que dicha frontera separa de manera correcta ambos tipos de datos

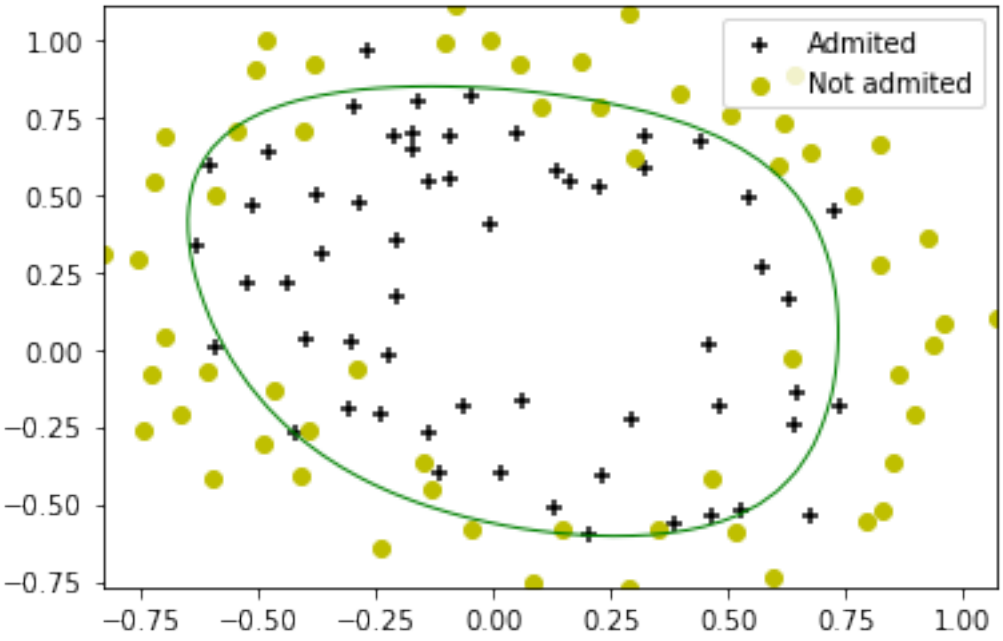
[62]:
```
def optimiza_reg_paragrafica(datos,Lambda):
    X_aux = datos[:,:-1]
    Y = datos[:,-1]

    p = PolynomialFeatures(6)
    X = p.fit_transform(X_aux)

    c, gr = preparaFunciones(Lambda)

    T = np.zeros(X.shape[1])

    result = opt.fmin_tnc(func = c, x0 = T, fprime = gr, args = (X, Y))
    c_f = coste(result[0], X, Y)
    print("coste:", c_f)
    return result[0]
```

[63]:
```
def crearGraficaConFrontera2(datos, Theta, labels = ['y = 1', 'y = 0']):
    X = datos[:,:-1]
    Y = datos[:,-1]

    c1 = np.where(Y == 1)
    c2 = np.where(Y == 0)

    plt.scatter(X[c1,0], X[c1,1] , marker = '+', c = 'k', label = labels[0])
    plt.scatter(X[c2,0], X[c2,1], marker = 'o', c = 'y', label = labels[1])
    plt.legend(loc = 'upper right')

    frontera2(X, Y, Theta, PolynomialFeatures(6), plt)

    plt.show()
```

[64]:
```
def frontera2(X, Y, theta, poly, plt):
    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(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')
```

[67]:
```
datos = cargaDatos("ex2data2.csv")
crearGraficaConFrontera2(datos, optimiza_reg_paragrafica(datos, 1), ['Admited',␣
 ↪'Not admited'])
```

```
coste: 0.46822079610635636
```



1.2.5 Efectos de la regularizacion:

En este apartado creamos una funcion que represente graficamente los efectos obtenidos gracias a la regularización y podemos observar en la grafica resultante que para lambdas muy parecidos (10,11,12..) los valores devueltos no cambian pero aun asi caunto mayor es el valor lambda mayor es la precisión con la que se calcula la parte de entrenamiento y la función se vuelve mas reutilizable

[71]:
```
def evalua_reg(datos, parte_entrenamiento, Lambda):
    f = round(len(datos) * parte_entrenamiento / 100)
    datos_ent = datos[:f]
    datos_eva = datos[f:]

    theta = optimiza_reg(datos_ent,Lambda)

    dat_ev_x_n , dat_ev_y, mu, sigma = prepara_datos(datos_eva,6)

    res_eva = np.dot(dat_ev_x_n, theta)
    res_eva_m = []
    for e in map(lambda x : 1 if x >= 0.5 else 0, res_eva):
        res_eva_m.append(e)

    iguales = filter(lambda x : dat_ev_y[res_eva_m.index(x)] == x , res_eva_m)

    i = 0;
    for e in iguales:
        i = i+1
```

```
    return (i/len(datos_eva)*100)
```

[46]:
```
def grafica_ev():
    datos = cargaDatos("ex2data2.csv")
    X = []
    Y = []
    for i in range(1,100):
        X.append(i)
        Y.append(evalua_reg(datos,75,i))

    plt.figure()
    plt.plot(X,Y)
    plt.show()
```

[47]:
```
grafica_ev()
```