Javier Cordero Calvo y Juan Ruiz Jiménez

Aprendizaje Automático y Minería de Datos – Práctica 2

Parte 1:

En primer lugar, cargamos los datos mediante la función "carga_csv(file_name)", y acto seguido guardamos en las matrices correspondientes los datos leídos. También inicializamos el vector de Thetas con el valor 0 por cada componente.

```
def carga_csv(file_name):
    """carga el fichero csv especificado y lo
    devuelve en un array de numpy"""
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)

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

Y = datos[:, -1]
np.shape(Y)

thetas = np.zeros(3)
```

Tras tener todos los datos preparados, preparamos los puntos dados para pintarlos mediante la función "dibuja_puntos(X, Y)", la cual recibe las dos matrices de datos.

```
def dibuja_puntos(X, Y):
    pos = np.where(Y == 1)
    plt.scatter(X[pos, 0], X[pos, 1], marker = '+', c = 'red')

    neg = np.where(Y == 0)
    plt.scatter(X[neg, 0], X[neg, 1], marker = '.', c = 'blue')

dibuja_puntos(X, Y)
```

Creamos las matrices m, y n necesarias y establecemos el aplha a 0.01.

Tras esto, añadimos la columna de 1's necesaria a la matriz X.

```
m = np.shape(X)[0]
n = np.shape(X)[1]
alpha = 0.01

# añadimos una columna de 1's a la X
X = np.hstack([np.ones([m, 1]), X])
```

Ahora nos disponemos a calcular la Theta óptima para nuestro problema mediante la función "calcOptTheta(thetas)", la cual recibe el vector de tetas previamente creado. También generamos una matriz nX la cual contiene las X de los datos leídos sin la columna de 1's, la cual nos servirá más adelante para pintar la recta óptima.

```
def calcOptTheta(theta):
    result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X,
Y))
    return result[0]

nX = datos[:, :-1]
T = calcOptTheta(thetas)
```

La función "calcOptTheta()" depende de otras dos funciones, la función "cost()" que nos devuelve el coste óptimo y la función "gradient()" que es la que implementa el gradiente. Así mismo, la función "cost()" depende de la función "sigmoid()" que devuelve el sigmoide de x.

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

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
```

Tras tener ahora guardada en T la theta óptima, llamamos a la función "pinta_frontera_recta(X, Y, theta)", que dibujará la recta óptima con la theta proporcionada.

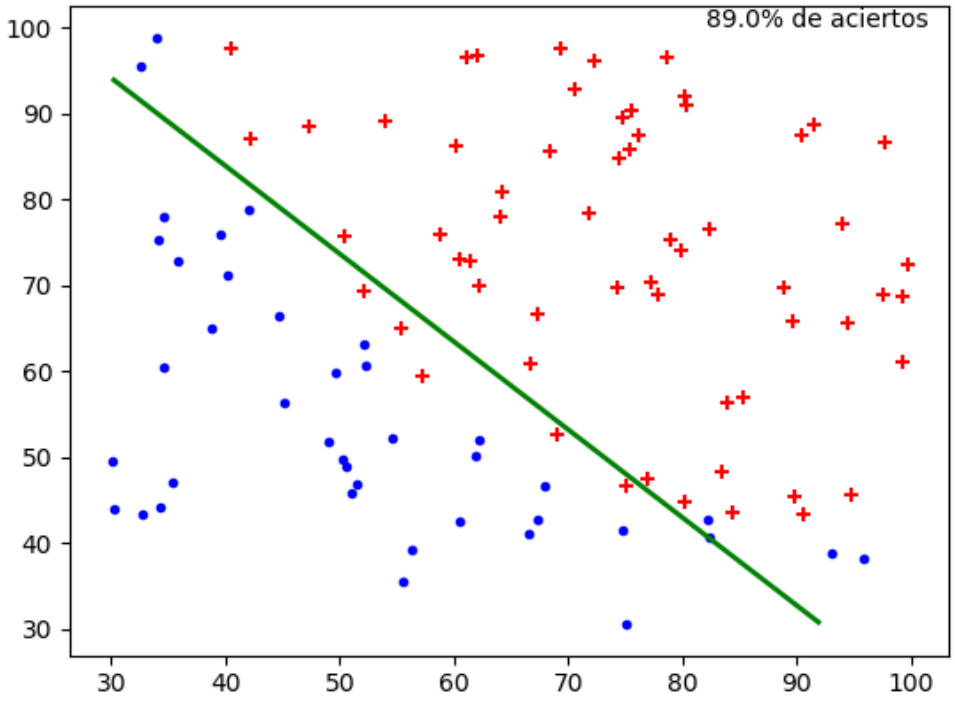
```
def pinta_frontera_recta(X, Y, theta):
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()

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

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

    # el cuarto parámetro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=2, colors='green')

pinta_frontera_recta(nX, Y, T)
```

Finalmente, mediante la función "calcAciertos(X, Y, t)" comprobamos el porcentaje de éxito que ha tenido nuestra solución y dibujamos el resultado obtenido.

```
def calcAciertos(X, Y, t):
    prediccion = 0
    cont = 0
    aciertos = 0
    totales = len(Y)

    for i in X:
        if sigmoid(np.dot(i, t)) >= 0.5:
            prediccion = 1
        else:
            prediccion = 0

        if Y[cont] == prediccion:
            aciertos += 1

        cont += 1

    porcentaje = aciertos / totales * 100
    plt.text(82,100, str(porcentaje) + "% de aciertos")
```

```
calcAciertos(X, Y, T)

plt.show()
```

Tras todo este proceso, el resultado obtenido es:



Este resultado por lo tanto nos divide nuestros datos en dos grupos con una tasa del 89.0% de aciertos.

Parte 2:

Para esta parte, partimos de la carga de datos y la declaración de las matrices del apartado anterior, así como de la forma de pintar los puntos en la grafica.

```
datos = carga_csv('ex2data2.csv')

X = datos[:, :-1]
np.shape(X)

Y = datos[:, -1]
np.shape(Y)

pinta_puntos(X, Y)
```

Tras tener las matrices guardadas y los puntos preparados para pintar en el futuro, vamos a ampliar la matriz X hasta la potencia de 6, de tal manera que ahora X contará con 28 elementos por la función:

```
poly = PF(6)
Xpoly = poly.fit_transform(X)
landa = 1
```

(también damos un valor inicial a lambda (“landa” en el código) de 1)

Ahora calculamos el valor de Theta óptimo igual que en el apartado 1, con la diferencia de que ahora tenemos que tener en cuenta la lambda previamente establecida. Por lo tanto, el método “calcOptTheta()” cambia y en los argumentos “args” también tenemos que incluir dicho valor lambda.

Pese a que las funciones “cost()” y “sigmoid()” siguen siendo iguales, la función “gradient()” cambia para poder tener en cuenta dicha lambda.

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

    aux=np.r_[[0],theta[1:]]

    result = grad+(landa*aux/m)
    return result

def calcOptTheta():
    result = opt.fmin_tnc(func=cost, x0=np.zeros(Xpoly.shape[1]), fprime=
gradient, args=(Xpoly, Y, landa))
    return result[0]
```

Tras haber realizado esos cálculos, tenemos guardado en T la Theta óptima, así que llamamos a la función “pinta_frontera_curva()” con X, Y, theta y lambda.

```
def pinta_frontera_curva(X, Y, theta, landa):
    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_mi
n, x2_max))
    h = sigmoid(poly.fit_transform(np.c_[xx1.ravel(), xx2.ravel()]).dot(t
heta))
    h = h.reshape(xx1.shape)
    plt.contour(xx1, xx2, h, [0.5], linewidths=2, colors='green')
```

Ya temenos preparade la función curva para pintar al igual que los puntos, pero antes de ello debemos calcular la tasa de aciertos con el método "calcAciertos()" al que ahora le pasamos la nueva X calculada "Xpoly".

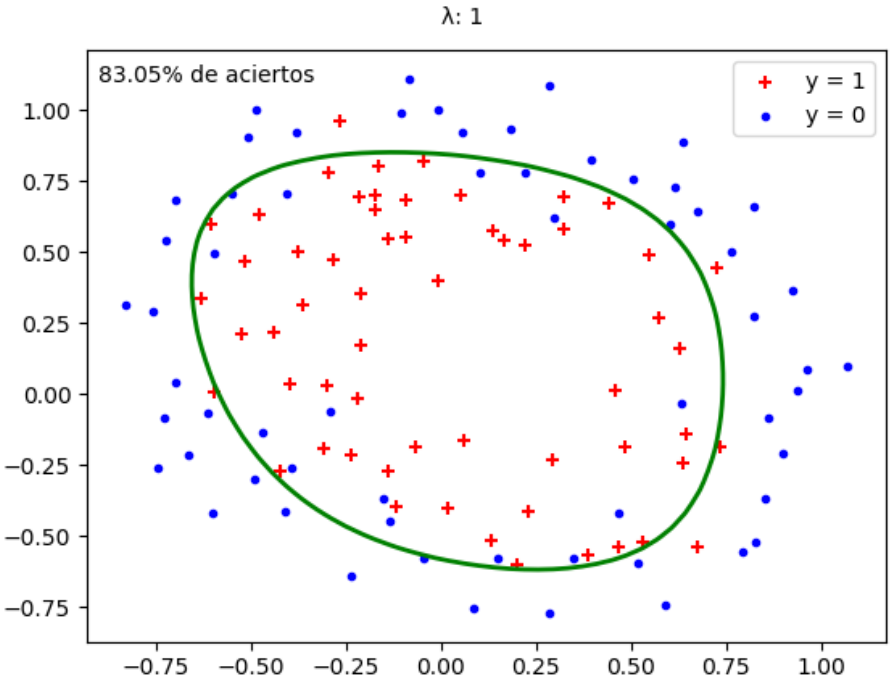
La preparamos también para mostrarla por pantalla, indicando que coja exclusivamente los dos primeros decimales.

```
def calcAciertos(X, Y, t):
    prediccion = 0
    cont = 0
    aciertos = 0
    totales = len(Y)

    for i in X:
        if sigmoid(np.dot(i, t)) >= 0.5:
            prediccion = 1
        else:
            prediccion = 0

        if Y[cont] == prediccion:
            aciertos += 1

        cont += 1

    porcentaje = aciertos / totales * 100

    plt.text(-
0.9,1.1, str("{0:.2f}".format(porcentaje)) + "% de aciertos")
```

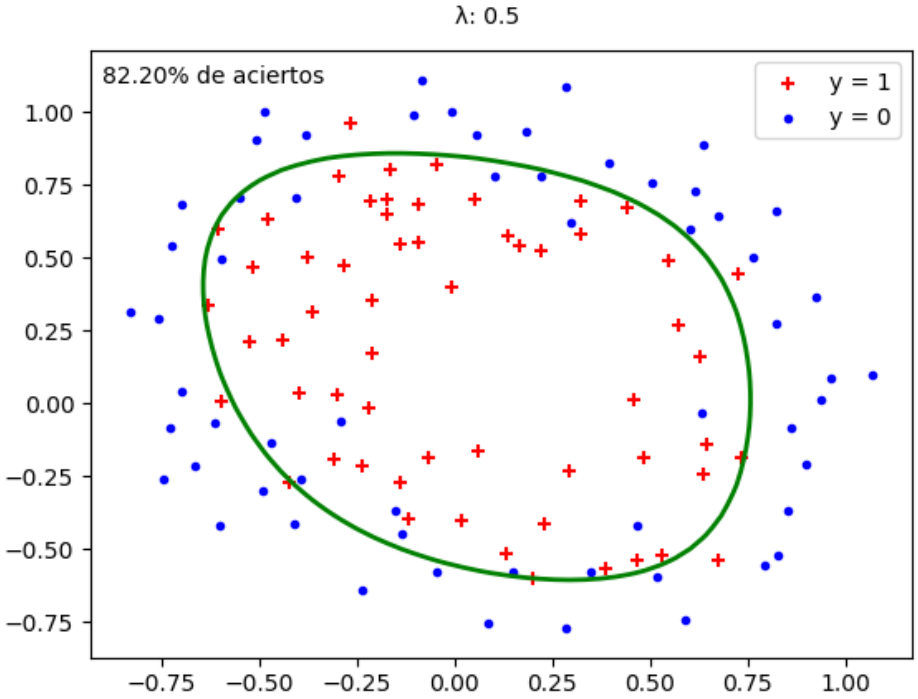
Finalmente, dibujamos el resultado final:

```
plt.legend()
plt.show()
```

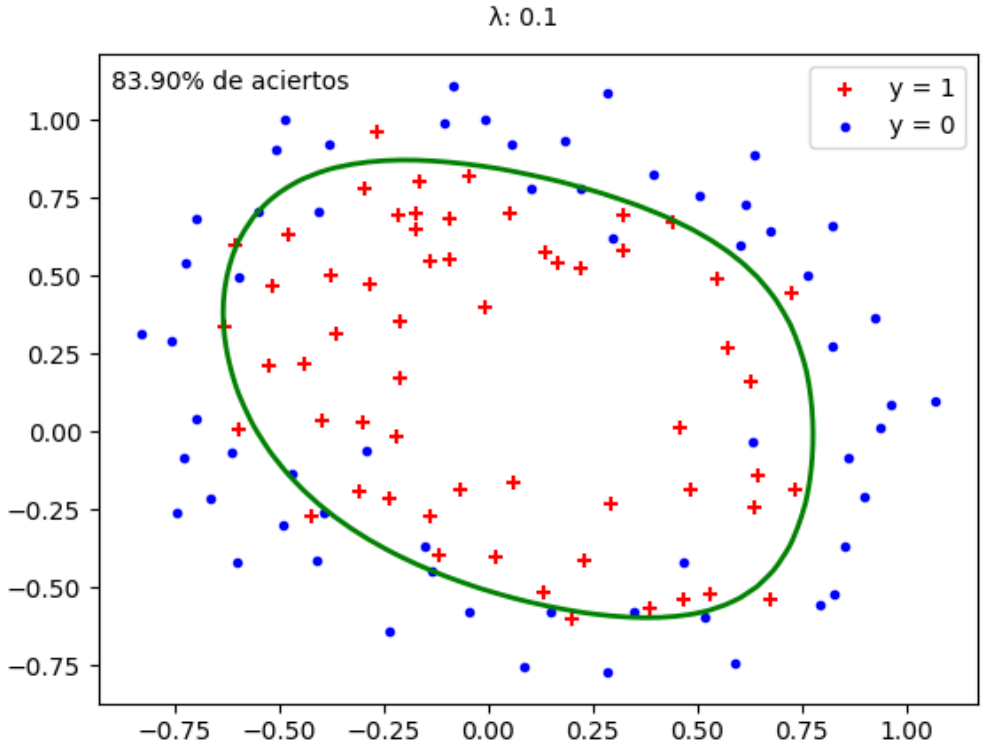
El resultado obtenido implica que con lambda = 1 obtenemos un 83.05% de acierto como se muestra en la siguiente imagen:



Por lo tanto, si modificamos lambda, tanto nuestra grafica como nuestra tasa de aciertos cambia. Para lambda = 0.5:



Finalmente, para lambda = 0.1



Y lambda = 0: (No tener en cuenta lambda)