Práctica 2

Víctor Choza Merino

Adrián Turiel Charro

En la **primera parte**, la función ***sigmoide*** es equivalente a la función hipótesis de las prácticas anteriores, es decir, aquella que trata de guiarnos hacia el conocimiento de los hechos a partir del análisis de lo observado.

Para esta práctica, en vez de calcular el descenso gradiente, usamos una función de la librería scipy.optimize, **fmin_tnc,** que recibe tres argumentos principales, la función de coste, el vector de ceros (valores iniciales del vector Thetas) y la función gradiente. Con esta función obtenemos el valor de los parámetros que minimizan la función de coste para la regresión logística.

En la función ***porcentaje_aprobado****,* para cada uno de los elementos del vector que son devueltos la función sigmoide, comprobamos si su valor supera o no 0.5, para contar así el número de aprobados. Y después calculamos el porcentaje total de aprobados.

La función de **pinta_frontera_recta** es donde representamos la solución en forma de gráfica. Primero ajustamos la rejilla con *meshgrid* y después representamos por un lado los aprobados juntos a los suspensos, separados por una línea que representamos con la función *contour.* Esta línea de separación corresponde por tanto a aquellos puntos donde la probabilidad de pertenecer a cualquiera de las dos clases es del 50%.

La principal diferencia en la **segunda parte** es que los datos a observar ya no son linealmente separables, por lo que tratamos de buscar una función que se aproxime y pueda predecir el resultado lo máximo posible.

Ahora tras obtener el conjunto de datos no hace falta añadir la columna de unos, en su lugar utilizamos la clase sklearn.preprocessing.PolynomialFeatures para extender cada ejemplo de entrenamiento con los términos polinómicos de x1 y x2 hasta la sexta potencia, completando así un total de 28 atributos para cada ejemplo. De este modo pasamos de los dos atributos con los que definimos la recta, a 28 atributos con los que se puede generar una función que delimite mejor la regresión logística.

La función de coste y del gradiente cambian ligeramente, se les añade la expresión en amarillo:
Coste:

$$J(\theta) = \left[\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradiente:

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

Añadiendo a estas funciones la expresión lambda, podemos probar diferentes ajustes y así quedarnos con el resultado más preciso o el que más nos interese.

En función del lambda que le pasemos a la función **opt.fmin_tnc,** el valor de los parámetros que minimizan la función de coste para la regresión logística irán variando. Para mayores valores de lambda, los coeficientes de theta tendrán menos libertad, por lo que resultará en una función menos precisa, aunque conseguimos minimizar la función de coste. Para valores más pequeños sin embargo, los coeficientes de theta tienen mayor libertad, con lo que se consigue mayor precisión a costa de una mayor función de coste.

PRÁCTICA 2.1 - VÍCTOR CHOZA MERINO - ADRIÁN TURIEL CHARRO

1. Regresión logística

In [1]:

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

In [2]:

```
def sigmoide(z): #g(z)
    return (1 / (1 + np.exp(-z)))
```

In [3]:

```
def hipotesis(X, Thetas):
    return sigmoide(np.matmul(X, Thetas))
```

In [4]:

```
def gradiente(Thetas, X, Y):
    H = sigmoide(np.matmul(X, Thetas))   #Hipótesis
    #Thetas -= np.matmul(H - Y,X)*(alpha/len(X))
    return np.matmul(X.T, H - Y)*(1/len(X))
```

In [5]:

```
def coste(Thetas, X, Y):
    H = sigmoide(np.matmul(X, Thetas))
    return (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1 - H)))
```

In [6]:

```
def porcentaje_aprobado(X, Y, Thetas):
    H = hipotesis(X, Thetas)
    # (H >= 0.5) -> Los valores mayores de 0.5 se ponen a 1 (True)
    # y los demás a 0 (False)

    # (np.sum((H >= 0.5)==Y) -> Número de valores cuyo valor real es 1 y
    # su valor correspondiente a la hipótesis es 1 (True)
    return (np.sum((H >= 0.5)==Y)/len(X))*100
```

In [7]:

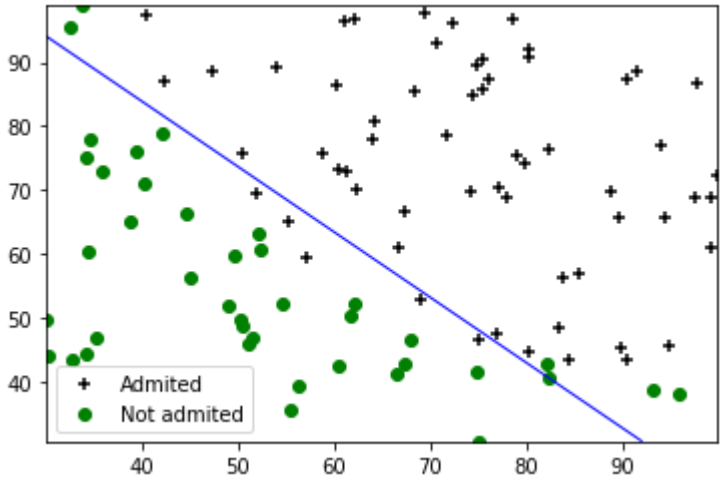
```
def pinta_frontera_recta(X, Y, Theta):
    plt.figure()
    X = X[:,1:np.shape(X)[1]]  # Todas las columnas menos la de unos
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()

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

    aux = np.c_[np.ones((xx1.ravel().shape[0], 1)), xx1.ravel(), xx2.ravel()]
    h = sigmoide(aux.dot(Theta))
    h = h.reshape(xx1.shape)

    # Obtiene un vector con los índices de los ejemplos positivos
    pos1 = np.where(Y == 1)
    # Obtiene un vector con los índices de los ejemplos negativos
    pos2 = np.where(Y == 0)

    # Dibuja los ejemplos positivos
    plt.scatter(X[pos1, 0],X[pos1, 1],marker='+',c='k',label='Admited' )
    # Dibuja los ejemplos negativos
    plt.scatter(X[pos2, 0],X[pos2, 1],marker='o',c='g',label='Not admited'  )

    plt.legend()     # parar mostrar la leyenda
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    plt.show()
    plt.savefig("frontera.pdf")
    plt.close()
```

In [8]:

```
def regresion_logistica(datos):
    valores = read_csv(datos, header=None).to_numpy()
    X = valores[:,:-1]
    Y = valores[:,-1]

    m = np.shape(X)[0]   #Filas
    n = np.shape(X)[1]   #Columnas

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

    Thetas = np.zeros(n+1)    #Thetas calculadas

    result = opt.fmin_tnc (func=coste , x0=Thetas , fprime=gradiente , args =(X, Y))
    Thetas = result [0]

    pinta_frontera_recta(X, Y, Thetas)

    return X, Y, Thetas
```

In [9]:

```
X, Y, Thetas = regresion_logistica("ex2data1.csv")
print (porcentaje_aprobado(X, Y, Thetas), "%" " de aprobados")
```




```
89.0 % de aprobados
```

PRÁCTICA 2.2 - VÍCTOR CHOZA MERINO - ADRIÁN TURIEL CHARRO

Ver como usar plot_decisionboundary

Type *Markdown* and LaTeX: α^2

2. Regresión logística regularizada

In [1]:

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

In [2]:

```
def sigmoide(z): #g(z)
    return (1 / (1 + np.exp(-z)))
```

In [3]:

```
def gradiente(Thetas, X, Y, lambdaa):
    H = sigmoide(np.matmul(X, Thetas))   #Hipótesis
    return np.matmul(X.T, H - Y)*(1/len(X)) + (lambdaa/len(X))*Thetas
```

In [4]:

```
def coste(Thetas, X, Y, lambdaa):
    H = sigmoide(np.matmul(X, Thetas))
    m = len(X)
    # Thetas[1:] xq el prim valor no nos interesa
    sumatorioTheta = sum(np.power(Thetas[1:],2))
    return (- 1 / m) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
                        np.log(1 - H))) +  (lambdaa/2*m) * sumatorioTheta
```

In [6]:

```
def plot_decisionboundary(X, Y, theta, poly):
    plt.figure()

    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)

    # Obtiene un vector con los índices de los ejemplos positivos
    pos1 = np.where(Y == 1)
    # Obtiene un vector con los índices de los ejemplos negativos
    pos2 = np.where(Y == 0)


    # Dibuja los ejemplos positivos
    plt.scatter(X[pos1, 0],X[pos1, 1],marker='+',c='k',label='Admited' )
    # Dibuja los ejemplos negativos
    plt.scatter(X[pos2, 0],X[pos2, 1],marker='o',c='g',label='Not admited'  )

    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='g')
    plt.savefig("boundary.pdf")
    plt.show()
    plt.close()
```

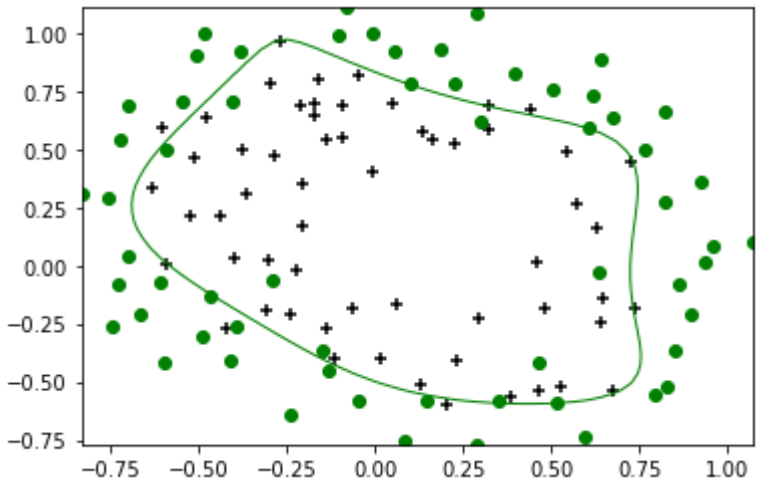
In [13]:

```
def regresion_logistica_regularizada(datos,lambdas):
    valores = read_csv(datos, header=None).to_numpy()

    X = valores[:,:-1]
    Y = valores[:,-1]

    m = np.shape(X)[0]   #Filas
    n = np.shape(X)[1]   #Columnas

    # Para PolynomialFeatures(2), con [a, b, c]
    # obtenemos [1, a, b, c, a^2, b^2, c^2, ab, bc, ca]
    # Para PolynomialFeatures(6), obtenemos 28 combinaciones
    poly = PolynomialFeatures(6)

    X_Poly = poly.fit_transform(X)
    Thetas = np.zeros(len(X_Poly[1]))

    for l in lambdas:
        print("Lambda: ", l)
        result = opt.fmin_tnc (func=coste , x0=Thetas ,
                               fprime=gradiente , args =(X_Poly, Y, l))
        Thetas = result [0]

        plot_decisionboundary(X, Y, Thetas, poly)
        print("-----------------------------------------------------")
```

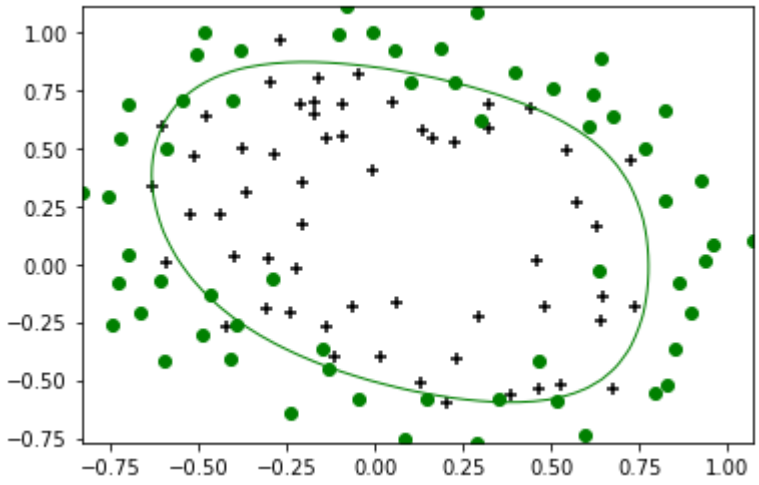
In [17]:

```
lambdas = np.array([0.0000001, 0.1, 1, 10, 100])
regresion_logistica_regularizada("ex2data2.csv",lambdas)
```

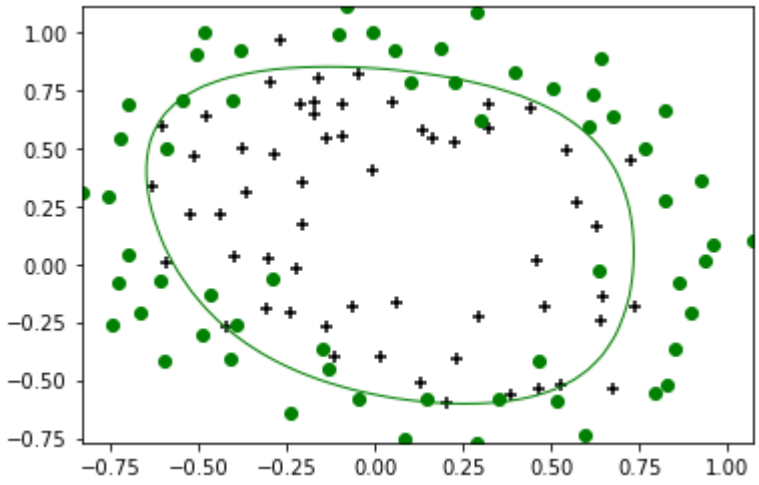
```
Lambda:  1e-07
```



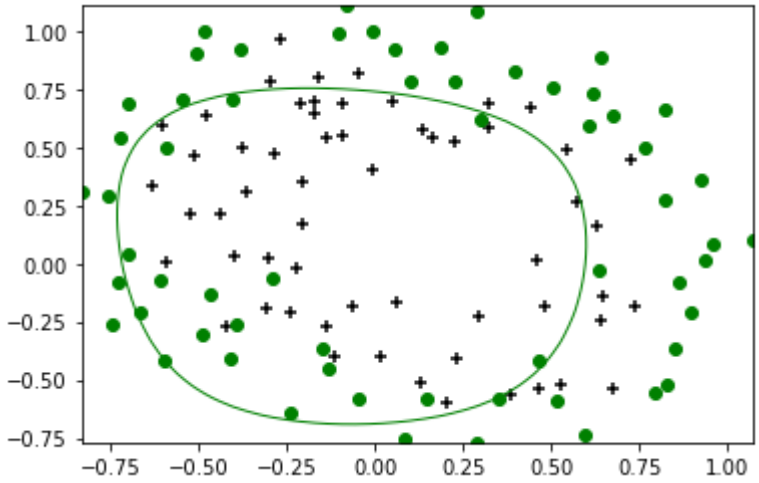

```
----------------------------------------------------------
Lambda:  0.1
```



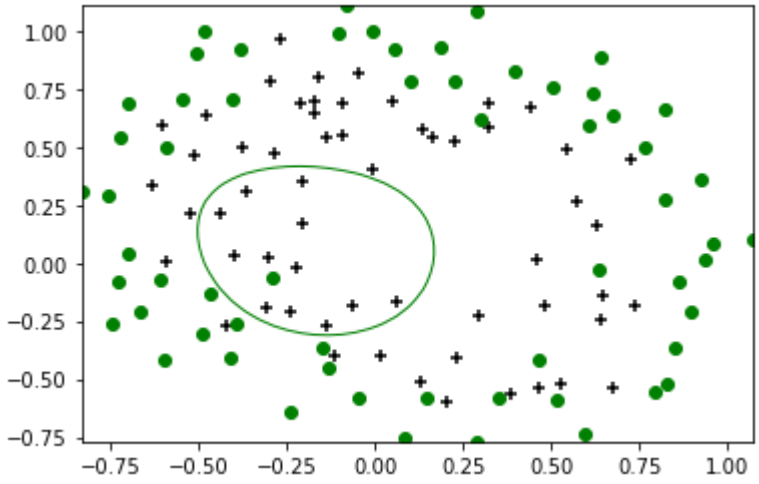

```
----------------------------------------------------------
Lambda:  1.0
```



Lambda: 10.0



Lambda: 100.0



In [ ]:

```
1
```