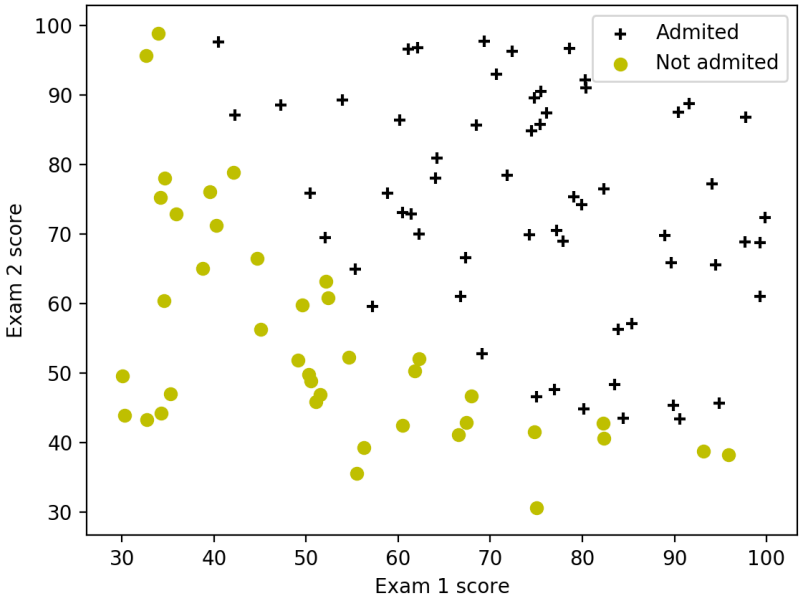
Práctica 2

Alejandro Aizel Boto
Miguel Robledo Blanco

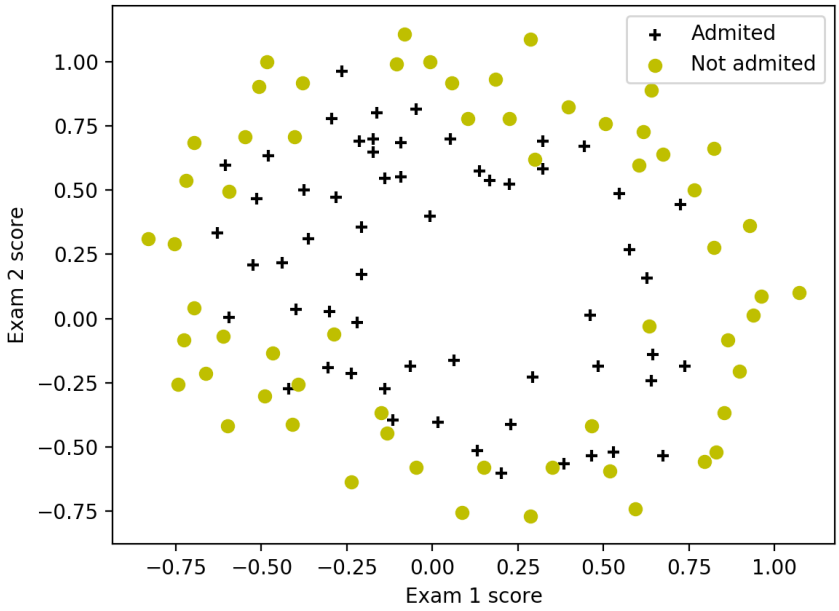
Índice

1. Descripción General

Esta práctica, al igual que la anterior consta de dos parte. La primera consiste en aplicar el método de regresión logística para estimar la probabilidad de que un alumno sea admitido en la universidad en función de su nota en dos exámenes. Para ello disponemos de la información de si fueron o no admitidos representados mediante un 1 o un 0 respectivamente.



El segundo método consiste en aplicar el método de regresión logística regularizada para encontrar una función que pueda predecir si un microchip pasará o no el control de calidad.



2. Explicación de la Solución

En este caso la práctica como hemos comentado anteriormente se divide en dos apartados, la regresión lineal con una sola variable y la regresión lineal con varias variables.

El código comienza con una llamada al método `main()` que contiene dos funciones, una por cada parte de la práctica. Vamos a comenzar explicando la primera parte.

```
def main():
    logistic_regression()
    regularized_regression()
```

La función `logistic_regression()` comienza con la llamada a la función `load_csv()` que nos va a devolver, en un array de Numpy, los datos contenidos en el archivo csv que le indiquemos por parámetro. El código de esta función lo tenemos a continuación.

```
def logistic_regression():
    Data = load_csv('Práctica 2/Recursos/ex2data1.csv')

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

    X = np.hstack([np.ones([len(X), 1]), X])
    Theta = np.zeros(len(X[0]))

    result = opt.fmin_tnc(func=cost , x0=Theta, fprime=gradient , args=(X,
                  Y))
    theta_opt = result [0]

    show_result(X[:, 1:], Y, theta_opt, 'Exam 1 score', 'Exam 2 score',
         'Admited', 'Not Admited')

    print('Correctly classified: {} %'.format(evaluation(X, Y, theta_opt)))
```

```
def load_csv(file_name):
    values = read_csv(file_name, header=None).values

    return values.astype(float)
```

Una vez tengamos los datos leídos lo que hacemos es separarlos en un array de `X` y un array de `Y`. Añadimos una columna de unos con a `X` para poder posteriormente hallar el valor de la hipótesis como el producto de dos vectores y creamos una variable `Theta` inicializada con tamaño el número de elementos de `X`.

Una vez tenemos los datos preparados vamos a utilizar la función opt.fmin_tnc para calcular la regresión. Esta función recibe varios parámetros entre ellos: la fucnión a

minimizar, `cost`; la función sobre la que se calcula el coste, `gradient`; el valor de theta inicial, `Theta`; y los argumentos necesarios para las funciones, `(X, Y)`.

A continuación se muestran las funciones `cost`:

```
def cost(Theta, X, Y):
    H = sigmoid(np.matmul(X, Theta))

    return (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
         np.log(1 - H)))
```

Que de forma vectorizada calcula:

$$J(\theta) = -\frac{1}{m}((log\,(g(X\theta)))^T y + (log\,(1 - g(X\theta)))^T (1 - y))$$

A continuación se muestra la función `gradient`:

```
def gradient(Theta, XX, Y):
    H = sigmoid(np.matmul(XX, Theta))

    return (1 / len(Y)) * np.matmul(XX.T, H - Y)
```

Que de forma vectorizada calcula:

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

A continuación llamamos a la función show_result que tiene la siguiente forma:

```
def show_result(X, Y, Theta, x_lable, y_lable, ad_legends, not_legend,
         Lambda=None, poly=None, name='graph'):
    plt.figure()

    show_data(X, Y, x_lable, y_lable, ad_legends, not_legend, name)

    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))

    if Lambda == None:
        h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)), xx1.ravel(),
                 xx2.ravel()].dot(Theta))
    else:
        plt.title('lambda = {}'.format(Lambda))
```

```
            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='b')
        plt.savefig('Práctica 2/Recursos/{}_result.png'.format(name), dpi=200)
        plt.close()
```

Esta función recibe varios parámetros:

- `X`: Datos de entrada
- `Y`: Resultado de datos de entrada
- `Theta`: Theta óptima
- `x_lable`: Nombre del eje X de la gráfica
- `y_lable`: Nombre del eje Y de la gráfica
- `ad_legends`: Leyenda para los casos admitidos
- `not_legend`: Leyenda para los casos no admitidos
- `Lambda`: Valor lambda en caso de que se necesite
- `poly`: Objeto poly en caso de que se necesite
- `name`: Nombre de la gráfica

Lo primero que hacemos es llamar a la función show_data() a la que le pasamos estos parámetros para que nos genere la gráfica de datos de entrada y nos deje el plt preparado para imprimir el resultado.

```
def show_data(X, Y, x_lable, y_lable, ad_legends, not_legend, name):
    admited = np.where(Y == 1)
    not_admited = np.where(Y == 0)

    ad = plt.scatter(X[admited, 0], X[admited, 1], marker='+', c='k')
    not_ad = plt.scatter(X[not_admited, 0], X[not_admited, 1], c='y')

    plt.xlabel(x_lable)
    plt.ylabel(y_lable)
    plt.legend((ad, not_ad), (ad_legends, not_legend))
    plt.savefig('Práctica 2/Recursos/{}.png'.format(name), dpi=200)
```

Lo que hacemos primero es separar los admitidos de los no admitidos y los imprimimos con un '+' y un '·' respectivamente. Después ponemos nombre a los ejes y a la leyenda, y los guardamos con el nombre dado con una resolución de 200.

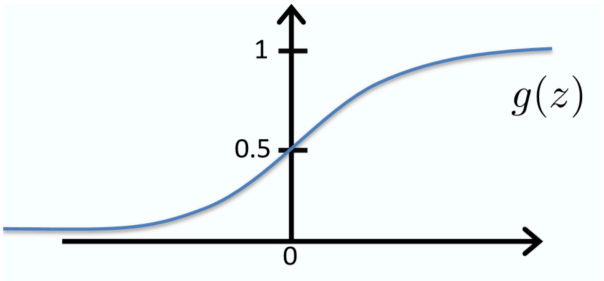
Volviendo a lo anterior tenemos que distinguir entre la regularizada y la normal. En caso de que estemos en la normal el valor de `Lambda` será `None` y podremos calcular `h` de esa forma. En caso contrario incluiremos en la gráfica el valor de `Lambda` y calcularemos `h` de la otra forma. Por último realizamos el contorno con la función `plt.contour()` y guardamos la gráfica con el nombre especificado.

La función sigmoid() que se utiliza en varios puntos de la práctica tiene esta forma:

```
def sigmoid(X):
    return 1 / (1 + np.exp(-X))
```

Y se representa de esta forma:

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



Ésta función considera que todos aquellos valores en el intervalo [0.5, 1] serán tratados como admitidos y aquellos valores contenidos en el intervalo [0, 0,5) serán tratados como no admitidos.

Por último queda calcular el porcentaje de datos correctamente clasificados con respecto a los datos de entrada. Para ello tenemos la función `evaluation()`:

```
def evaluation(X, Y, Theta):
    hipotesis = sigmoid(np.matmul(X, Theta))

    hipotesis[hipotesis >= 0.5] = 1
    hipotesis[hipotesis < 0.5] = 0

    result = hipotesis == Y

    return (np.sum(result) / len(result)) * 100
```

Lo que hacemos es, utilizando la función sigmoide y el theta óptimo obtenido, calculamos el resultado con respecto a las `X` de entrada. Luego aproximamos a 0 o 1 dependiendo de si es menor que 0.5 o mayor respectivamente, y comparamos con la `Y`. Hacemos un conteo de los valores que se han clasificado correctamente y sacamos el porcentaje dividiendo por el número de casos.

Llegamos ya a la segunda parte de la práctica. Comenzamos con la llamada a la función `regularized_regression()`:

```
def regularized_regression():
    Data = load_csv('Práctica 2/Recursos/ex2data2.csv')

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

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

    Theta = np.zeros(len(X_n[0]))
    Lambda = 1

    result = opt.fmin_tnc(func=cost_reg, x0=Theta, fprime=gradient_reg,
                  args=(X_n, Y, Lambda))
    theta_opt = result[0]

    show_result(X, Y, theta_opt, 'Microchip test 1', 'Microchip test 2',
         'y = 1', 'y = 0', Lambda, poly, name='reg_graph')

    print('Correctly classified: {} %'.format(evaluation(X_n, Y,
         theta_opt)))
```

Hace exactamente lo mismo que la función de la primera parte pero con la diferencia de que ahora los datos de entrada los vamos a ampliar utilizando la función `PolynomialFeatures()`. Tambien vamos a tener una variable `Lambda` inicializada a 1. El resto como se puede observar es igual, lo único que cambia son las funciones que le pasamos por parámetro a la función `opt.fmin_tnc()`. Estas funciones son las siguientes.

La función de coste se sustituye por `cost_reg()`:

```
def cost_reg(Theta, X, Y, Landa):
    C = cost(Theta, X, Y)

    return C + Landa / (2 * len(X)) * np.sum(Theta ** 2)
```

Que es la versión regularizada del coste representada de forma vectorizada por:

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

La función de gradiente se sustituye por `gradient_reg()`:

```
def gradient_reg(Theta, XX, Y, Landa):
    G = gradient(Theta, XX, Y)

    G[1:] += Landa / len(XX) * Theta[1:]

    return G
```

Que es la versión regularizada del gradiente representada de forma vectorizada por:

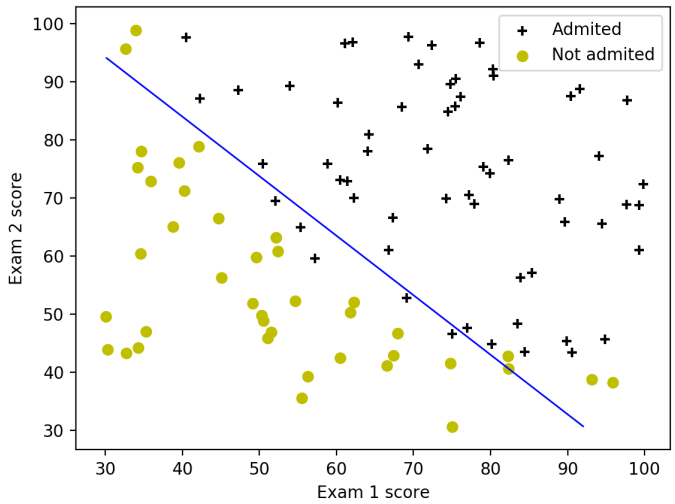
$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} X^T (g(X\theta) - y) + \frac{\lambda}{m} \theta_j$$

Hay que tener en cuenta de no incluir el término de regularización en el cálculo del gradiente respecto de theta 0.

3. Resultados

En este apartado vamos a ir comentando los distintos resultados que hemos obtenido.

Vamos a comenzar con el primer apartado, correspondiente a la compañía de distribución de comida.



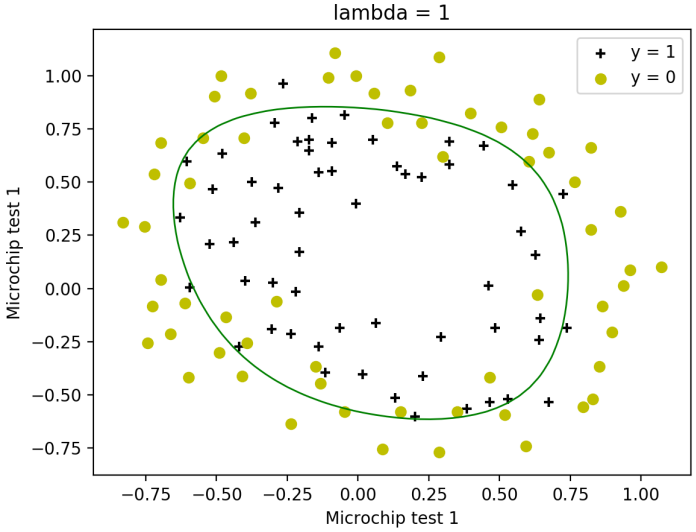
Tras llamar a la función que nos calcula la regresión obtenemos un valor de Theta óptimo con el que podemos obtener, mediante un corte, la recta de separación que se muestra en la figura.

Obtuvimos también el porcentaje de datos que se clasificaron correctamente:

```
> print('{} %'.format(evaluation(X, Y, theta_opt)))
```

```
  Correctly classified: 89.0 %
```

Para la segunda parte obtuvimos la siguiente gráfica:



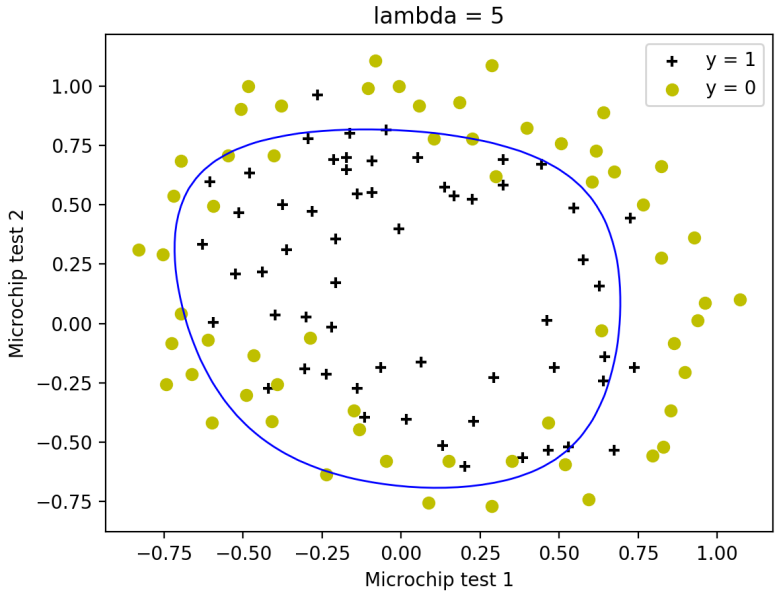
Al aumentar el tamaño de los datos de entrada pudimos obtener una separación mucho más precisa como se puede observar en la gráfica.

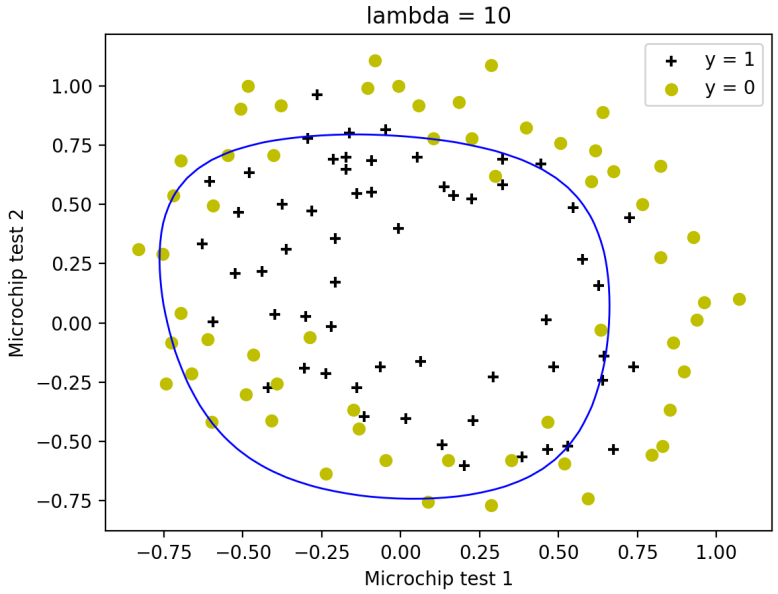
Obtuvimos también el porcentaje de datos que se clasificaron correctamente:

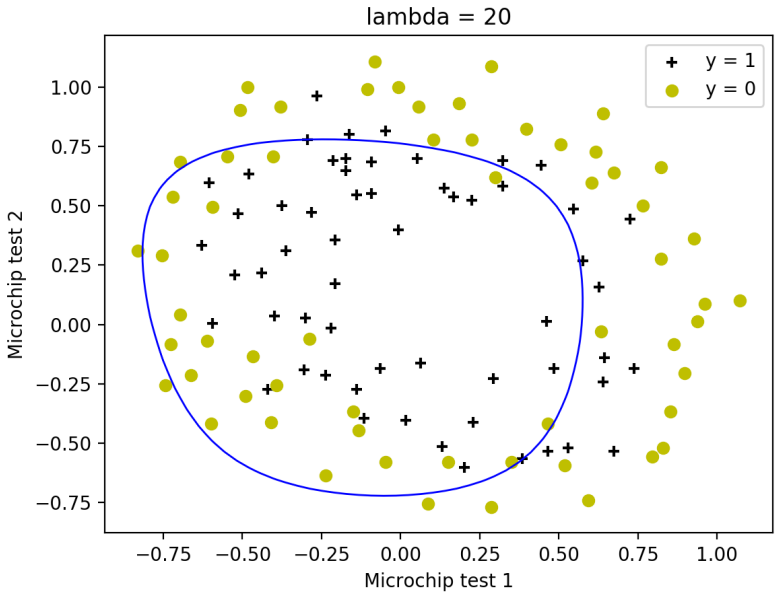
```
> print('{} %'.format(evaluation(X_n, Y, theta_opt)))
```

```
  Correctly classified: 83.0 %
```

Para ver como afecta el término de regularización al aprendizaje logístico hemos realizado esta segunda parte para 3 valores distintos de Lambda que se muestran a continuación:







Como podemos observar, a mayor valor de lambda, el "círculo" que separa los casos admitidos de los no admitidos se va desplazando más a la esquina inferior izquierda, resultando en una peor predicción.

4. Conclusión

Esta práctica nos ha servido para entender el funcionamiento de la regresión logística normal y regularizada así como la utilidad de las bibliotecas y funciones que aporta Python y que nos facilitan mucho los cálculos. Hemos aprendido a utilizar la función sigmoide, que es derivable, para, dependiendo del valor obtenido, determinar si la hipótesis es 0 o 1. También nos hemos familiarizado con la función `fmin_tnc` que nos realiza la regresión únicamente pasando el nombre de las funciones, además de la función `PolynomialFeatures` que nos ha permitido expandir de forma sencilla el tamaño de entrada de datos.

En el siguiente apartado se muestra el código completo por si se quiere ver seguido o hacer alguna prueba compilada.

5. Código

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

def load_csv(file_name):
    values = read_csv(file_name, header=None).values

    return values.astype(float)

def show_data(X, Y, x_lable, y_lable, ad_legends, not_legend, name):
    admited = np.where(Y == 1)
    not_admited = np.where(Y == 0)

    ad = plt.scatter(X[admited, 0], X[admited, 1], marker='+', c='k')
    not_ad = plt.scatter(X[not_admited, 0], X[not_admited, 1], c='y')

    plt.xlabel(x_lable)
    plt.ylabel(y_lable)
    plt.legend((ad, not_ad), (ad_legends, not_legend))
    plt.savefig('Práctica 2/Recursos/{}.png'.format(name), dpi=200)

def show_result(X, Y, Theta, x_lable, y_lable, ad_legends, not_legend,
Lambda=None, poly=None, name='graph'):
    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))

    show_data(X, Y, x_lable, y_lable, ad_legends, not_legend, name)

    if Lambda == None:
        h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)), xx1.ravel(),
xx2.ravel()].dot(Theta))
    else:
        plt.title('lambda = {}'.format(Lambda))

        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='b')
    plt.savefig('Práctica 2/Recursos/{}_result.png'.format(name), dpi=200)
    plt.close()

def sigmoid(X):
```

```
    return 1 / (1 + np.exp(-X))


def gradient(Theta, XX, Y):
    H = sigmoid(np.matmul(XX, Theta))

    return (1 / len(Y)) * np.matmul(XX.T, H - Y)

def cost(Theta, X, Y):
    H = sigmoid(np.matmul(X, Theta))

    return (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
np.log(1 - H)))

def evaluation(X, Y, Theta):
    hipotesis = sigmoid(np.matmul(X, Theta))

    hipotesis[hipotesis >= 0.5] = 1
    hipotesis[hipotesis < 0.5] = 0

    result = hipotesis == Y

    return (np.sum(result) / len(result)) * 100

def logistic_regression():
    Data = load_csv('Práctica 2/Recursos/ex2data1.csv')

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

    X = np.hstack([np.ones([len(X), 1]), X])
    Theta = np.zeros(len(X[0]))

    result = opt.fmin_tnc(func=cost , x0=Theta, fprime=gradient , args=(X,
Y))
    theta_opt = result [0]

    show_result(X[:, 1:], Y, theta_opt, 'Exam 1 score', 'Exam 2 score',
'Admited', 'Not Admited')

    print('Correctly classified: {} %'.format(evaluation(X, Y, theta_opt)))


def gradient_reg(Theta, XX, Y, Landa):
    G = gradient(Theta, XX, Y)

    G[1:] += Landa / len(XX) * Theta[1:]

    return G

def cost_reg(Theta, X, Y, Landa):
    C = cost(Theta, X, Y)

    return C + Landa / (2 * len(X)) * np.sum(Theta ** 2)

def regularized_regression():
    Data = load_csv('Práctica 2/Recursos/ex2data2.csv')

    X = Data[:, :-1]
    Y = Data[:, -1]
```

```
    poly = PolynomialFeatures(6)
    X_n = poly.fit_transform(X)

    Theta = np.zeros(len(X_n[0]))
    Lambda = 1

    result = opt.fmin_tnc(func=cost_reg, x0=Theta, fprime=gradient_reg,
args=(X_n, Y, Lambda))
    theta_opt = result[0]

    show_result(X, Y, theta_opt, 'Microchip test 1', 'Microchip test 2', 'y =
1', 'y = 0', Lambda, poly, name='reg_graph')

    print('Correctly classified: {} %'.format(evaluation(X_n, Y, theta_opt)))


def main():
    logistic_regression()
    regularized_regression()


main()
```