Logistic regression

First Part:

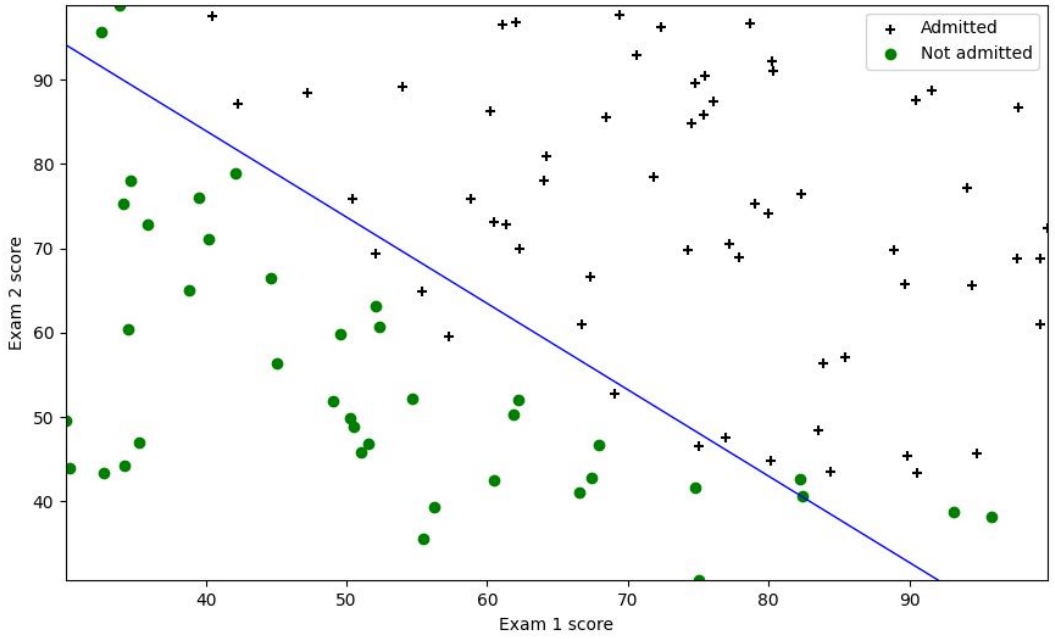
In this practice, our first dataset consists of the marks obtained by a group of candidates in two of the admission exams in a university and if they were or not admitted. With this data, our objective was to build a model using Logistic regression capable of predicting if a student will be admitted regarding his/her mark.

The code for this part:

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

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]
Y = datos[:, -1]
m = np.shape(X)[0]
n = np.shape(X)[1]

def dibuja_casos(X, Y):
    # Obtiene un vector con los índices de los ejemplos positivos y negativos
    pos = np.where(Y == 1)
    neg = np.where(Y != 1)

    # Dibuja los ejemplos positivos y negativos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label='Admitted')
    plt.xlabel("Exam 1 score")
    plt.ylabel("Exam 2 score")
    plt.scatter(X[neg, 0], X[neg, 1], c='green', label = 'Not admitted')
    plt.legend(loc = 'upper right')

def pinta_frontera_recta(X, Y, theta):
```

```
    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))

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

    # el cuarto parÃ¡metro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')

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

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


def gradient(theta, X, Y):
    theta = theta.reshape((len(theta), 1))
    Y = Y.reshape((len(Y), 1))
    A = sigmoid(np.matmul(X, theta))
    return np.dot(X.T, (A - Y)) / len(Y)

def classified_correct__percentage(theta, X, Y):
    theta = theta.reshape((len(theta), 1))
    A = sigmoid(np.matmul(X, theta))
    predictions = [1 if a > 0.5 else 0 for a in A]
    different = np.sum(predictions != Y)
    return len(predictions) - different / len(predictions) * 100
```

```
def model(x, Y):
    X = np.hstack([np.ones([m, 1]), x])
    theta = np.zeros(X.shape[1])
    result = opt.fmin_tnc (func=cost, x0=theta, fprime=gradient, args = (X,
Y))
    theta = result[0]
    print('cost: ' + str(cost(result[0], X, Y)))
    print('predicted correctly: ' + str(classified_correct__percentage(theta,
X, Y)) + "%")
    pinta_frontera_recta(x, Y, theta)
    dibuja_casos(x, Y)

model(X, Y)
plt.show()
```



The above graph represents the decision boundary that separates admitted students from not admitted.

The cost of our model is **0.20349770158947394** and correctly predicted examples are **89.0%**.

Second Part:

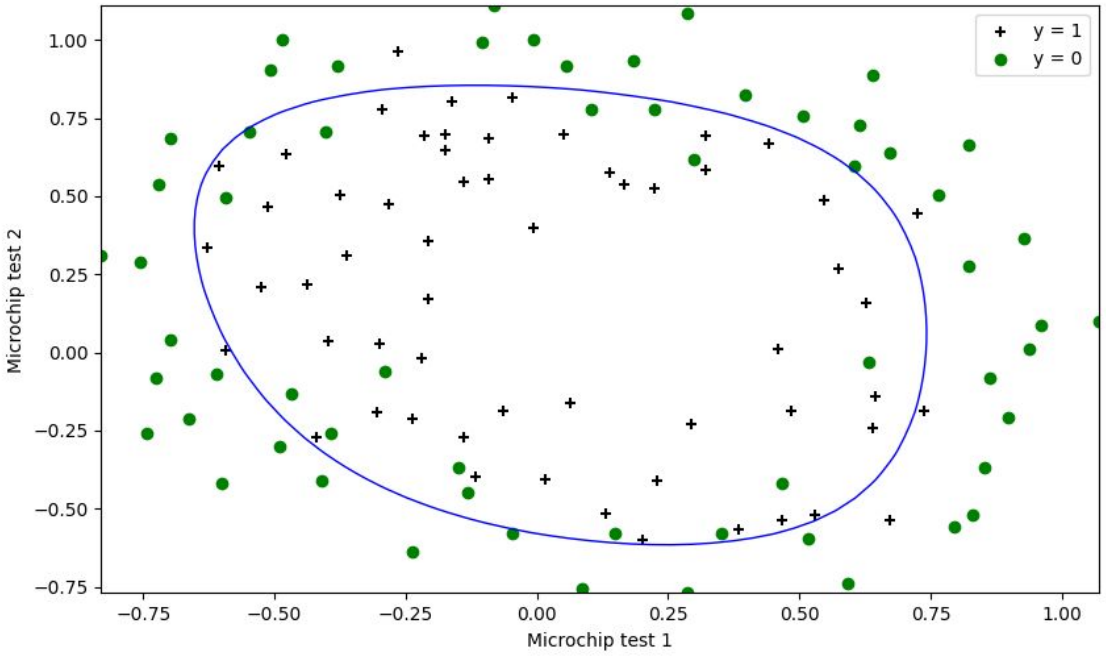
In the second part of the practice, we have been given another dataset with the results from two tests. The tests were on several microchips to define if they can pass the quality control or not. Our aim is, creating a regularized logistic regression capable of achieving the very same objective.

Code for this part:

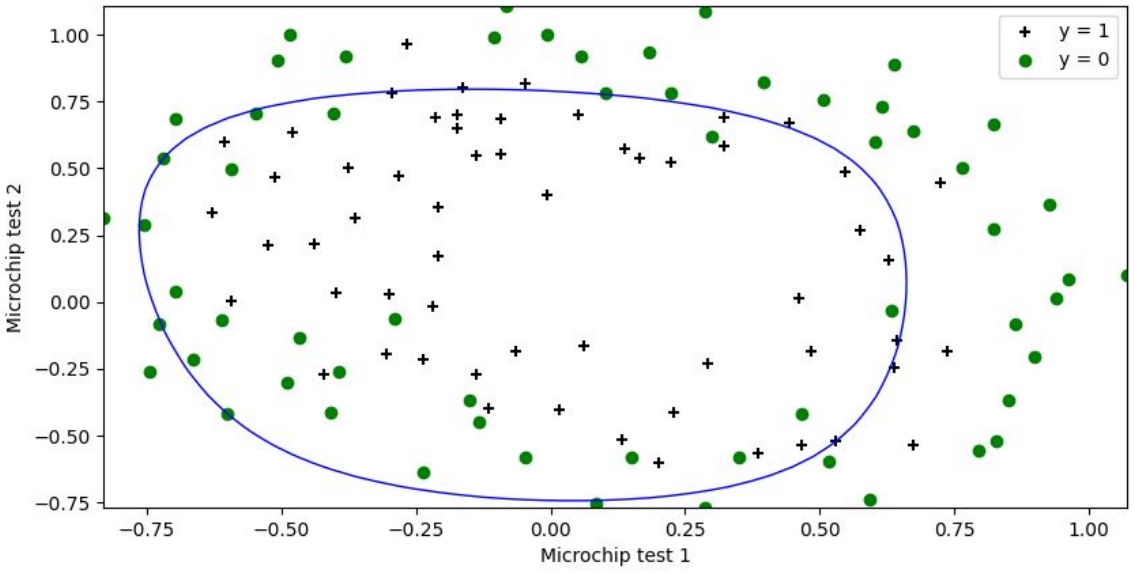
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize as opt
from pandas.io.parsers import read_csv
from sklearn.preprocessing import PolynomialFeatures

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('ex2data2.csv')
X = datos[:, :-1]
Y = datos[:, -1]
m = np.shape(X)[0]
n = np.shape(X)[1]

def dibuja_casos(X, Y):
    # Obtiene un vector con los índices de los ejemplos positivos y
negativos
    pos = np.where(Y == 1)
    neg = np.where(Y != 1)

    # Dibuja los ejemplos positivos y negativos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label='y = 1')
    plt.scatter(X[neg, 0], X[neg, 1], c='green', label = 'y = 0')
    plt.xlabel("Microchip test 1")
    plt.ylabel("Microchip test 2")
    plt.legend(loc = 'upper right')

def pinta_frontera_recta(X, Y, theta):
    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))

    theta = theta.reshape((len(theta), 1))

    trans = PolynomialFeatures(degree = 6, interaction_only=False,
include_bias=True)
    x = np.c_[xx1.ravel(), xx2.ravel()]
    X = trans.fit_transform(x)
    print(X.shape)
    h = sigmoid(np.matmul(X, theta))
    h = h.reshape(xx1.shape)

    # el cuarto parÃ¡metro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')

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

def cost(theta, X, Y, lambd = 1):
    theta = theta.reshape((len(theta), 1))
    Y = Y.reshape((len(Y), 1))
    A = sigmoid(np.matmul(X, theta))
    reg = (lambd / (2 * m)) * np.sum(theta ** 2)
    return (- 1 / (len(X))) * np.sum(Y * np.log(A) + (1 - Y) * np.log(1 -
A)) + reg


def gradient(theta, X, Y, lambd = 1):
    theta = theta.reshape((len(theta), 1))
    Y = Y.reshape((len(Y), 1))
    A = sigmoid(np.matmul(X, theta))
    identity = np.identity(theta.shape[0])
    identity[0][0] = 0
    return np.dot(X.T, (A - Y)) / len(Y) + (lambd / m) * np.dot(identity,
theta)


def model(x, Y):
```

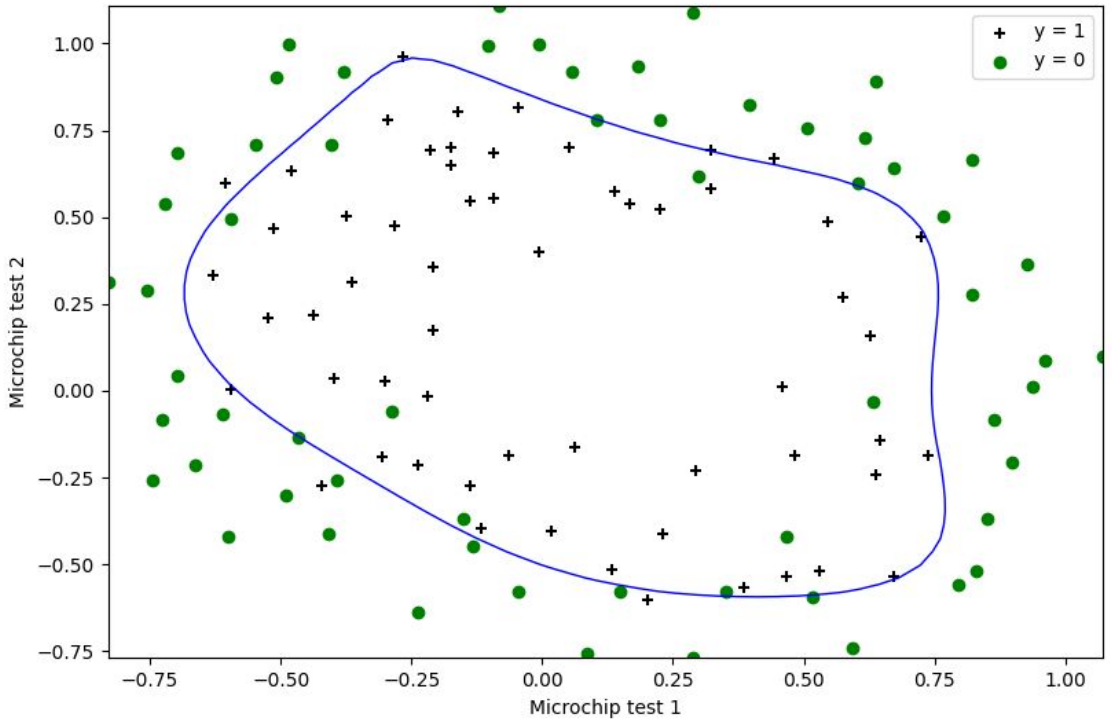
```
    trans = PolynomialFeatures(degree = 6, interaction_only=False,
include_bias=True)
    X = trans.fit_transform(x)

    theta = np.zeros(X.shape[1])
    result = opt.fmin_tnc (func=cost, x0=theta, fprime=gradient, args = (X,
Y))
    theta = result[0]
    pinta_frontera_recta(x, Y, theta)
    dibuja_casos(x, Y)

model(X, Y)
plt.show()
```



In the above graph is represented the decision boundary that separates chips passing the quality control from those not passing it.



In this image, the regularization parameter(**λ**) is equal to 10.



In this image, the regularization parameter(**λ**) is equal to 0.005.

Conclusion:

In this practice, we used two different datasets and created two logistic regression models. The first, without regularization, with a correct prediction rate of **89.0%** and

another with regularization. In the second, executing it with the different regularization parameter(**λ**), the results slightly vary. As expected, lower values of **λ** help the model fit the data better.

Gasan Nazer and Veronika Yankova