# Práctica 2 – Regresión logística

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

En esta práctica, se aplicará la regresión logística a dos conjuntos de datos, y se tratará de predecir si 'aceptamos o no'.

El primer conjunto de datos representa las notas obtenidas en dos exámenes por una serie de candidatos, junto a un booleano que indica si fueron aceptados o no. Deberemos construir un modelo por regresión logística que estime si un estudiante será admitido o no según sus notas. En este conjunto de datos se ve claramente que se puede realizar una división con una línea recta, es decir, son linealmente separables.

El segundo conjunto de datos representa dos pruebas aplicadas a una serie de microchips, junto al resultado booleano que indica si pasan la prueba de calidad o no. En este conjunto de datos la cosa se complica, ya que no son linealmente separables. Por ello, la región de división no vendrá definida por una simple recta, si no que será una región más compleja. Al haber multitud de datos, el modelo se ajustará demasiado al conjunto de datos, y no será bueno para otras predicciones (no generaliza) por lo que se aplicará la técnica de la regularización.

- Código de la práctica: parte 1 (datos linealmente separables)

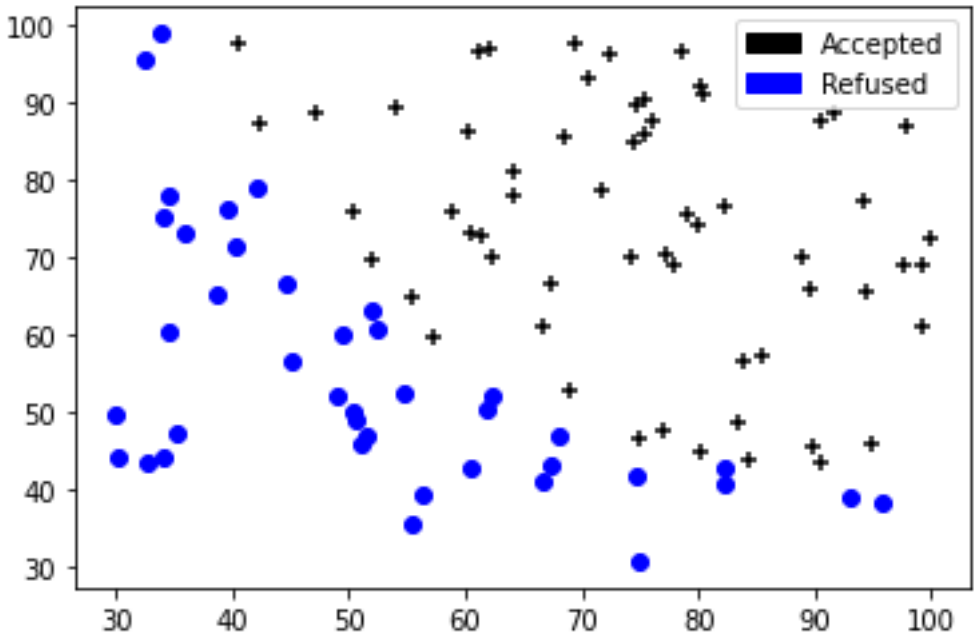
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from pandas.io.parsers import read_csv
import scipy.optimize as opt

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


def print_data(X, Y):
    pos = np.where(Y == 1)
    # Dibuja los ejemplos positivos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
    pos = np.where(Y == 0)
    plt.scatter(X[pos, 0], X[pos, 1], marker='o', c='b')
    plt.savefig("data.pdf")
    plt.legend(
        handles=[
            mpatches.Patch(color='black', label='Accepted'),
            mpatches.Patch(color='blue', label='Refused')
        ])
```

```
def sigmoide(Z):
    return 1 / (1 + np.exp(-Z))


def coste(theta, X, Y):
    m = np.shape(X)[0]
    H = sigmoide(np.matmul(X, np.transpose(theta)))

    l1 = np.transpose(np.log(H))
    l2 = np.transpose(np.log(1 - H))

    return (-1 / m) * ((np.matmul(l1, Y)) + (np.matmul(l2, (1 - Y))))


def gradiente(theta, X, Y):
    H = sigmoide(np.dot(X, np.transpose(theta)))
    grad = (1 / np.shape(X)[0]) * np.matmul(np.transpose(X), H - Y)
    return grad


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

    plt.contour(xx1, xx2, h, [0.5], linewidths=2, colors='green')


def porcentaje_aciertos(X, Y, theta):
    aciertos = 0
    j = 0
    for i in X:
        pred = sigmoide(np.dot(i, theta))
        if pred >= 0.5 and Y[j] == 1:
            aciertos += 1
        elif pred < 0.5 and Y[j] == 0:
            aciertos += 1
        j += 1
    return aciertos / len(Y) * 100


if __name__ == '__main__':
    datos = load_data('./ex2data1.csv')
```

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

    print_data(X, Y)

    X = np.hstack([np.ones([np.shape(X)[0], 1]), X])
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    theta = np.zeros(n)

    print('1.3. Calculo: ', str(coste(theta, X, Y)), str(gradiente(the
ta, X, Y)))

    nX = datos[:, :-1]
    result = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente, args
=(X, Y))

    print('Resultado opt.fmin_tnc: ' + str(result))

    theta_opt = result[0]

    print('Coste final: ' + str(coste(theta_opt, X, Y)))

    pinta_frontera_recta(nX, Y, theta_opt)

    print("Prediccion con un porcentaje de aciertos de:", porcentaje_a
ciertos(X, Y, theta_opt))

    plt.show()
```

- Resultados de ejecución: parte 1 (datos linealmente separables)

Para comprobar que los datos son linealmente separables, se ha generado la siguiente gráfica sobre el primer conjunto de datos.




Si iniciamos a 0 los elementos theta, se puede comprobar que obtenemos el valor de la función de coste en torno a 0,693 y un gradiente de [ -0.1 -12.0092 -11.2628].

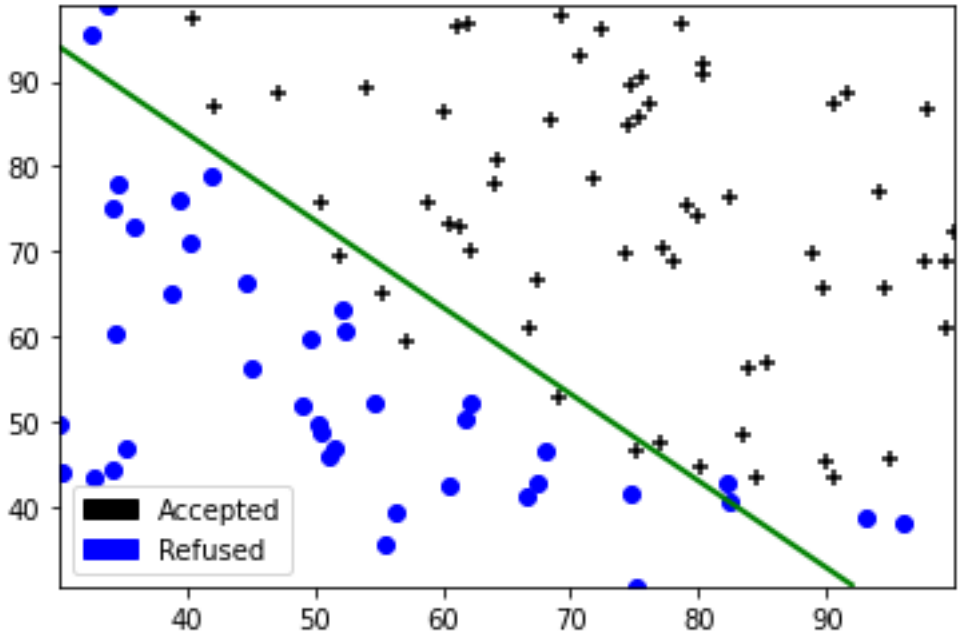
```
1.3. Calculo:  0.6931471805599453 [ -0.1        -12.00921659 -11.26284221]
```

Para obtener el valor de los parámetros theta que minimizan la función de coste, hacemos uso de la función scipy.optimize.fmin_tnc de SciPy.

Obtenemos los siguiente valores de coste óptimo y valores del gradiente:

```
Resultado opt.fmin_tnc: (array([-25.16131854,   0.20623159,   0.20147149])
Coste final: 0.20349770158947497
```

Gracias a esto, podemos pintar la frontera de decisión (descomentando la llamada a pinta_frontera_recta en el main):



Por último, hemos implementado una función que interpreta los resultados obtenidos de nuestro modelo. Si el resultado de la función sigmonide, utilizando el vector theta calculado, es mayor a 0,5, el alumno se considera admitido, mientras que si es menor, será rechazado. Si este resultado coincide con el resultado real, consideraremos un acierto de nuestro modelo, y obtenemos así un porcentaje de aciertos:

```
Prediccion con un porcentaje de aciertos de: 89.0
```

- Código de la práctica: parte 2 (varias variables)

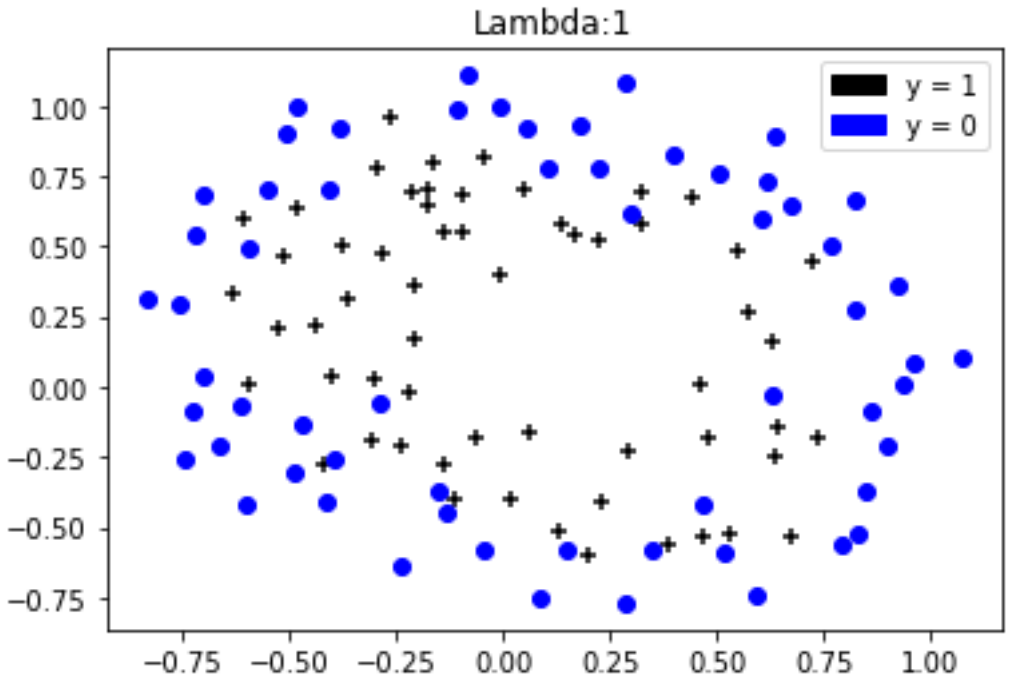
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn
from sklearn import preprocessing
from pandas.io.parsers import read_csv
import scipy.optimize as opt



def load_data(file_name):
    return read_csv(file_name, header=None).values.astype(float)


def print_data(X, Y):
    pos = np.where(Y == 1)
    # Dibuja los ejemplos positivos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
    pos = np.where(Y == 0)
    plt.scatter(X[pos, 0], X[pos, 1], marker='o', c='b')
    plt.savefig("data.pdf")
    plt.legend(
        handles=[
            mpatches.Patch(color='black', label='y = 1'),
            mpatches.Patch(color='blue', label='y = 0')
        ])


def sigmoide(Z):
    return 1 / (1 + np.exp(-Z))


def coste(theta, X, Y, l):
    m = np.shape(X)[0]
    H = sigmoide(np.matmul(X, np.transpose(theta)))

    l1 = np.transpose(np.log(H))
    l2 = np.transpose(np.log(1 - H))
```

```
    ret = (-1 / m) * ((np.matmul(l1, Y)) + (np.matmul(l2, (1 - Y))))
    return ret + l / (2 * m) * np.sum(theta * theta)


def gradiente(theta, X, Y, l):
    m = np.shape(X)[0]
    H = sigmoide(np.matmul(X, np.transpose(theta)))
    ret = (1 / m) * np.matmul(np.transpose(X), H - Y)
    return ret + (l / m) * theta


def plot_decisionboundary(X, Y, theta, poly):
    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')
    plt.savefig("boundary.pdf")


def porcentaje_aciertos(X, Y, theta):
    aciertos = 0
    j = 0
    print(X, theta)
    for i in range(len(X)):
        pred = sigmoide(np.dot(X[i], theta))
        if pred >= 0.5 and Y[j] == 1:
            aciertos += 1
        elif pred < 0.5 and Y[j] == 0:
            aciertos += 1
        j += 1
    return aciertos / len(Y) * 100


if __name__ == '__main__':
    datos = load_data('./ex2data2.csv')
    X = datos[:, :-1]
    Y = datos[:, -1]

    print_data(X, Y)

    mapFeature = sklearn.preprocessing.PolynomialFeatures(6)
    mapFeatureX = mapFeature.fit_transform(X)
    theta = np.zeros(mapFeatureX.shape[1])
    l=0.01
```

```
    print('2.2. Calculo: ', str(coste(theta, mapFeatureX, Y, 1)), np.a
rray2string(gradiente(theta, mapFeatureX, Y, 1)))
    result = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente, args
=(mapFeatureX, Y,1))
    print(result)
    theta_opt = result[0]
    data = np.delete(mapFeatureX, 0, axis=1)

    plot_decisionboundary(data, Y, theta_opt,mapFeature)
    plt.title("Lambda:" + str(1))
    plt.show()
    print("Prediccion con un porcentaje de aciertos de:",
        porcentaje_aciertos(mapFeatureX, Y, theta_opt))
```

- Resultados de ejecución: parte 2 (varias variables)

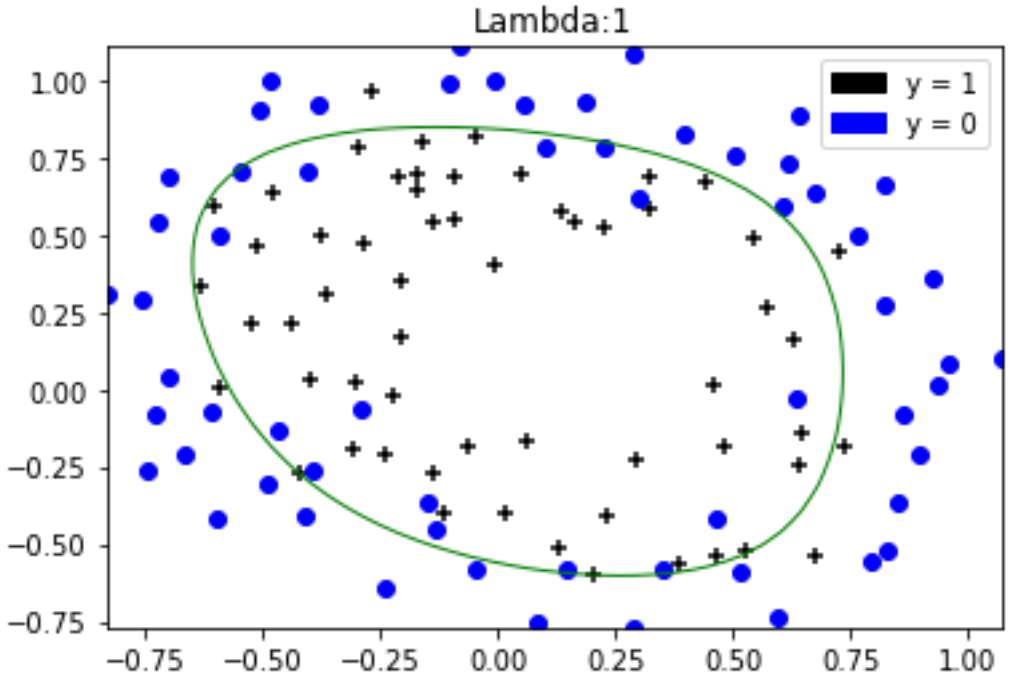
En primer lugar, podemos comprobar que, efectivamente, los datos no son linealmente separables.




Si iniciamos el vector de Theta a 0 y utilizamos el término de regularización lambda =1, obtenemos un coste aproximado de 0,693. También se incluyen los valores del gradiente, aplicando la regularización.

```
2.2. Calculo:  0.6931471805599453 [8.47457627e-03 1.87880932e-02 7.77711864e-05 5.03446395e-02
 1.15013308e-02 3.76648474e-02 1.83559872e-02 7.32393391e-03
 8.19244468e-03 2.34764889e-02 3.93486234e-02 2.23923907e-03
 1.28600503e-02 3.09593720e-03 3.93028171e-02 1.99707467e-02
 4.32983232e-03 3.38643902e-03 5.83822078e-03 4.47629067e-03
 3.10079849e-02 3.10312442e-02 1.09740238e-03 6.31570797e-03
 4.08503006e-04 7.26504316e-03 1.37646175e-03 3.87936363e-02]
```

Si descomentamos la llamada a plot_decisionboundary, obtenemos en el gráfico la región de decisión con los valores theta óptimos. Estos valores óptimos se calculan llamando a la función scipy.optimize.fmin_tnc.

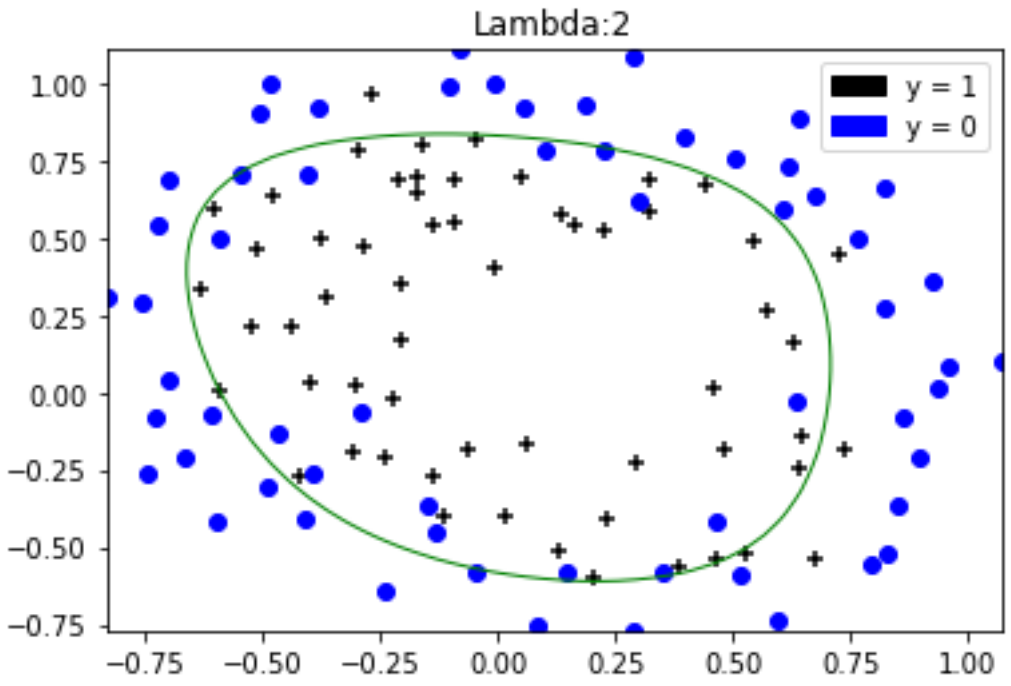



Con este valor lambda, se obtiene el siguiente porcentaje de acierto:

```
Prediccion con un porcentaje de aciertos de: 81.35593220338984
```

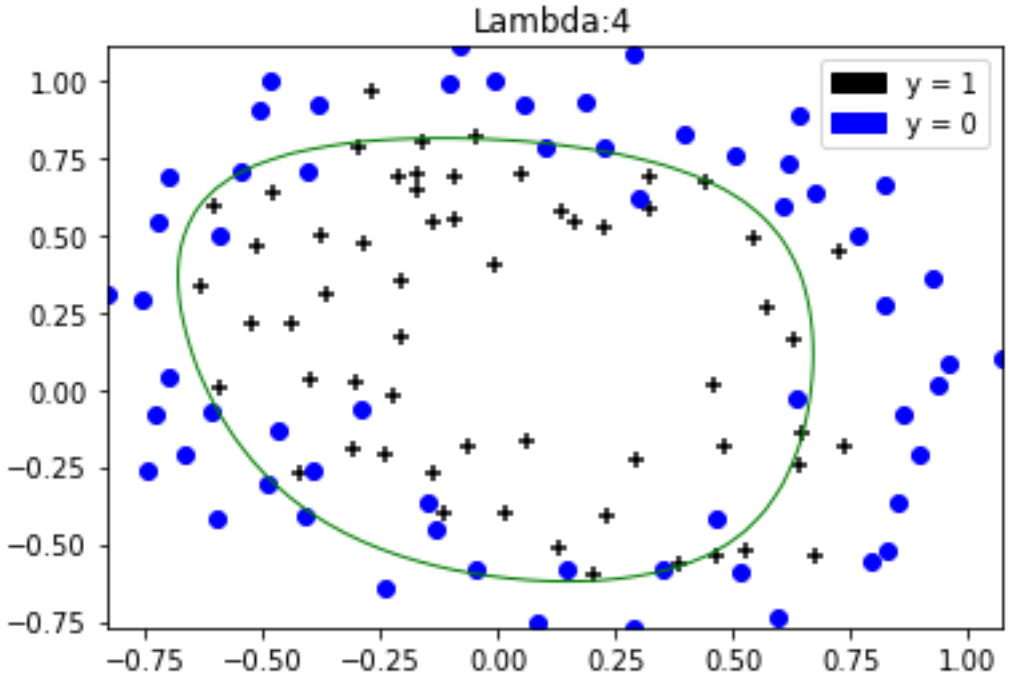
Si ajustamos el término de regularización, podemos obtener resultados interesantes.

En primer lugar, si lo aumentamos, podemos comprobar que, poco a poco, se va alejando de ser un modelo de predicción válido, ya que no se ajusta lo suficiente al modelo de datos.

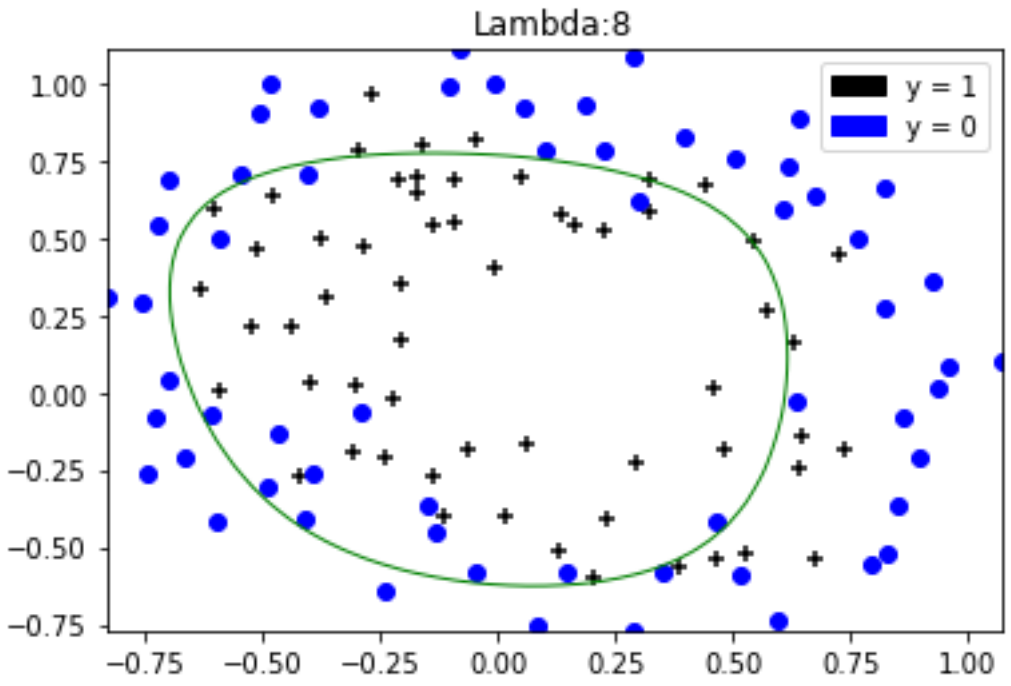



```
Prediccion con un porcentaje de aciertos de: 82.20338983050848
```

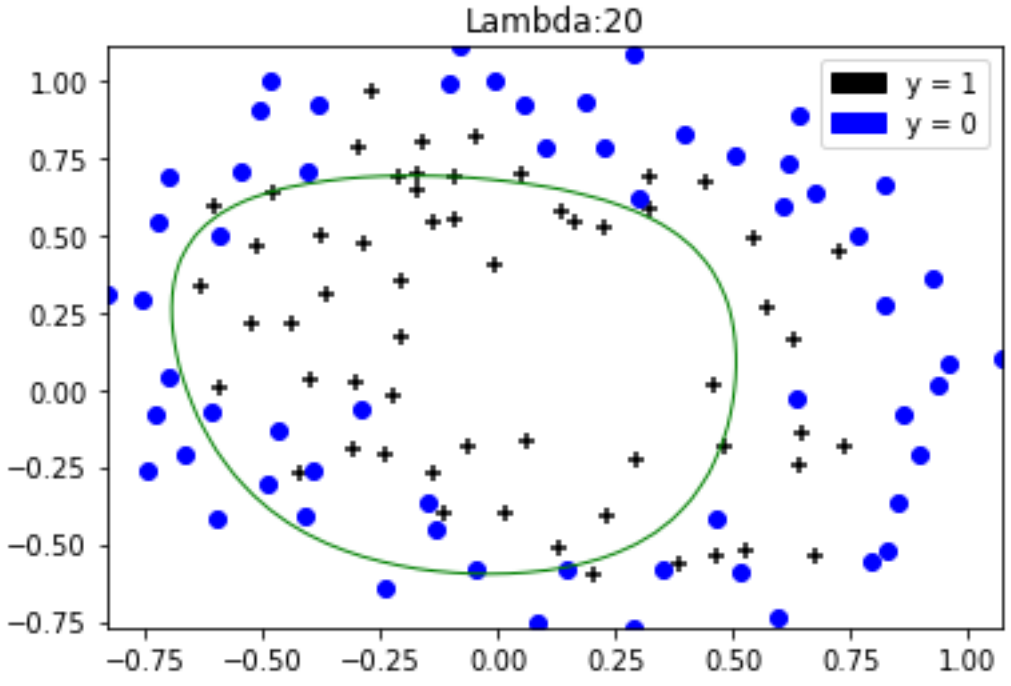
Aumentando Lambda a 2, obtenemos una mejora, pero a continuación veremos como esto no será siempre así.



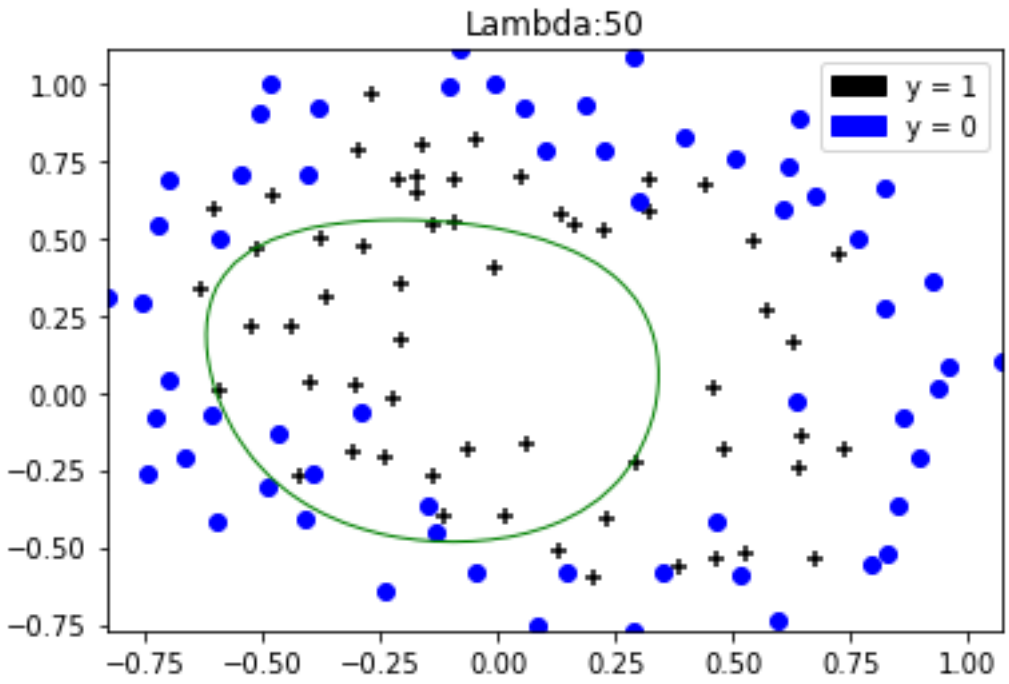

```
Prediccion con un porcentaje de aciertos de: 81.35593220338984
```




```
Prediccion con un porcentaje de aciertos de: 74.57627118644068
```

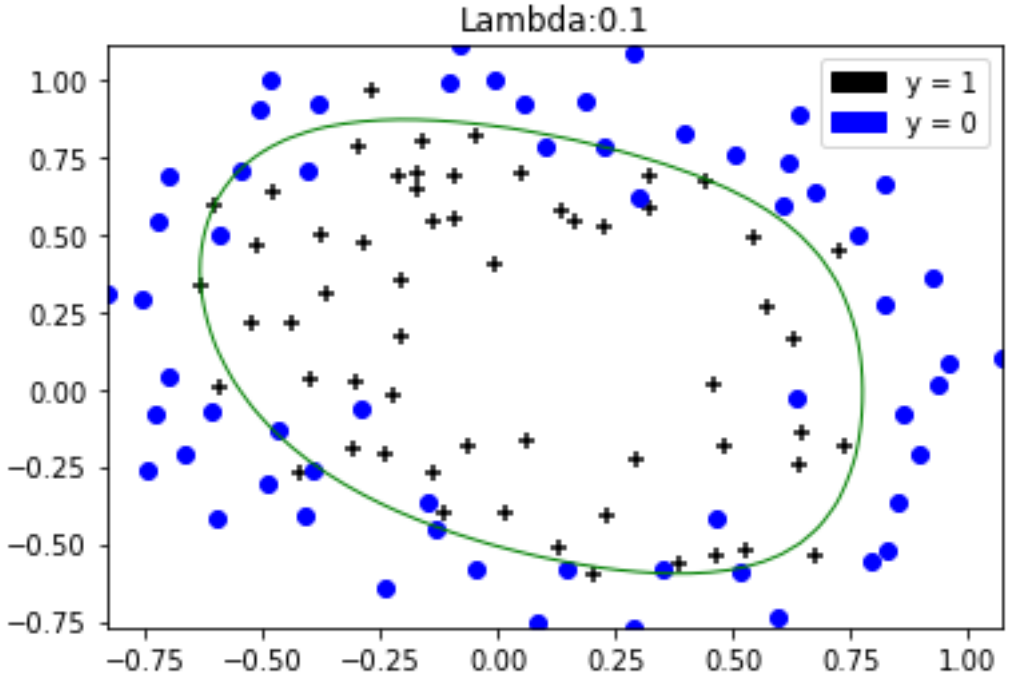



```
Prediccion con un porcentaje de aciertos de: 71.1864406779661
```

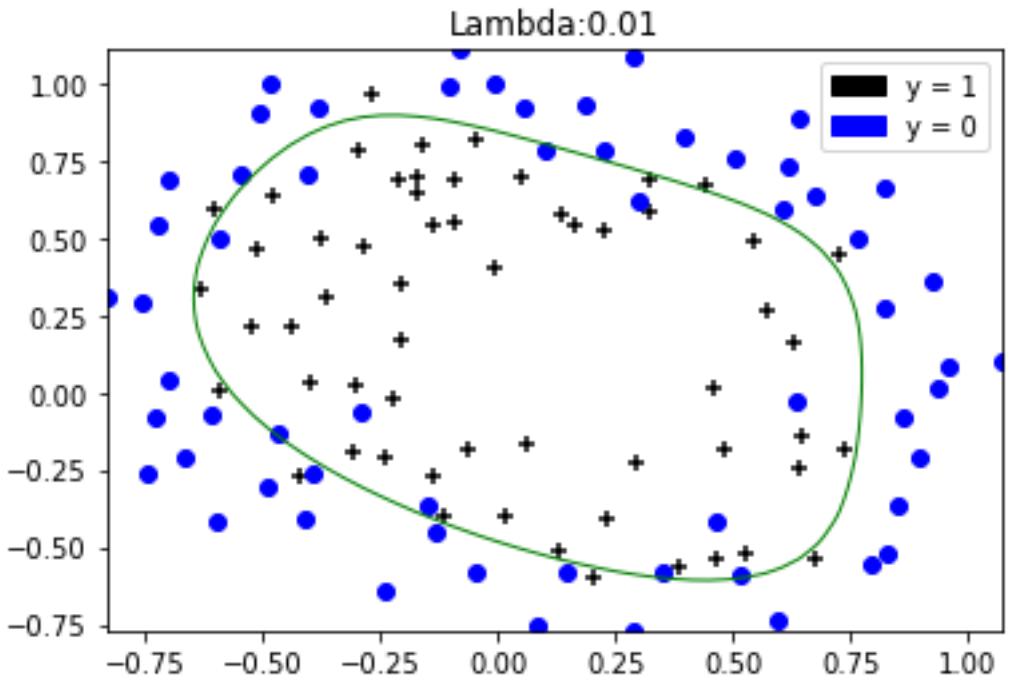



```
Prediccion con un porcentaje de aciertos de: 65.2542372881356
```

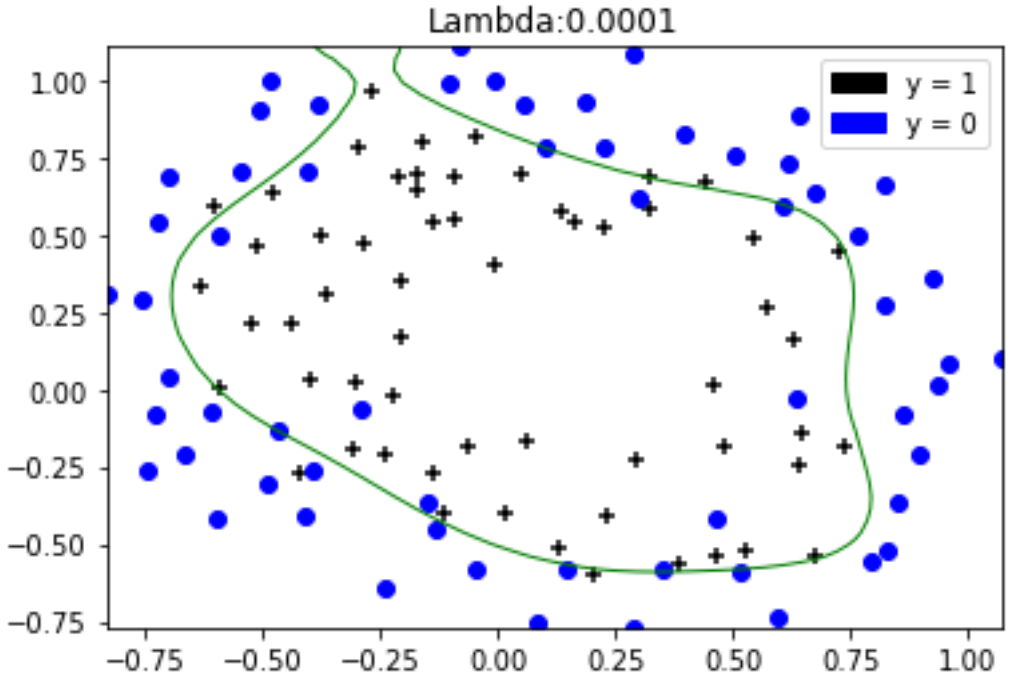
De igual modo, podemos provocar el efecto contrario con valores Lambda menores. Obtendremos un mejor porcentaje de aciertos, claro está, pero lo más probable es que, este modelo, falle con nuevos datos, ya que se ha ajustado demasiado a los dados inicialmente.



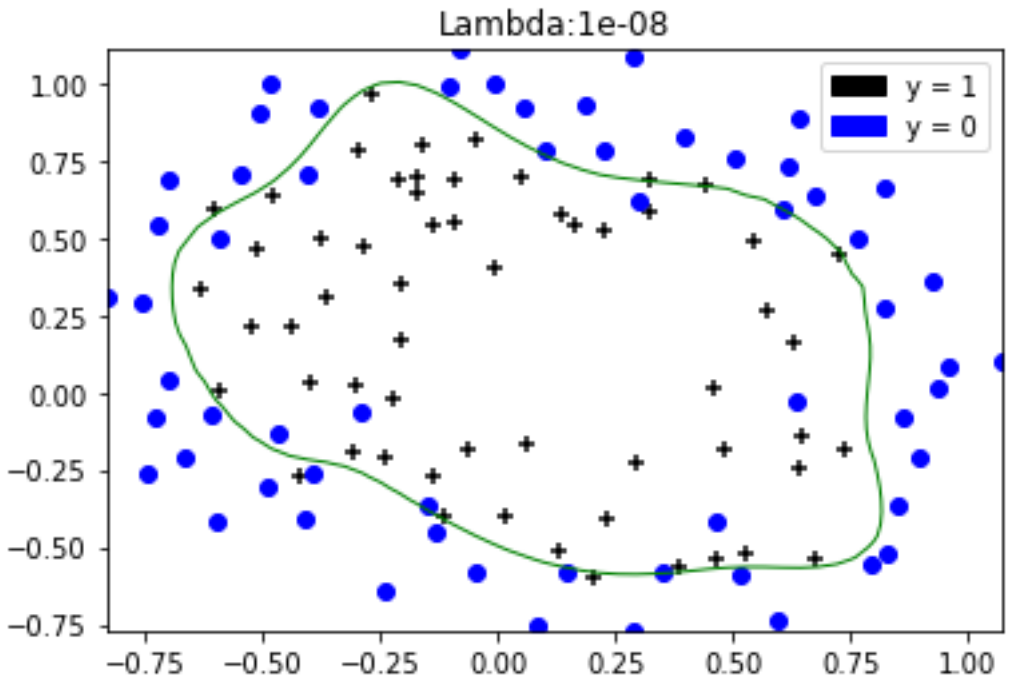

```
Prediccion con un porcentaje de aciertos de: 83.89830508474576
```




```
Prediccion con un porcentaje de aciertos de: 83.89830508474576
```




```
Prediccion con un porcentaje de aciertos de: 86.4406779661017
```




```
Prediccion con un porcentaje de aciertos de: 87.28813559322035
```

Llegados a este punto, la biblioteca que calcula los valores óptimos nos arroja mensajes de error, no puede hacer más cálculos.

```
tnc: Maximum number of function evaluations reached
```