实验7. 多项式扩展的Logistic回归

实验目的

1、实现多项式特征构造，了解欠拟合和过拟合

2、实现正则化的Logistic回归算法，了解通过正则化一定程度解决过拟合

3、实现正则化的损失函数和梯度计算

4、掌握调试技术

实验数据

ex7data.txt-用于非线性分类的数据集（芯片质量预测）

数据共三列：每行表示一个芯片经过测试的的历史数据，前两列为两项芯片测试的分数；第三列为标签，1表示通过质量测试，0表示未通过测试。

实验步骤

目标任务是预测工厂制造的芯片是否能够通过质量保证（QA）。根据历史上的芯片质量测试数据分布来看，无法通过线性分类器解决。因此，本实验通过特征的多项式扩展，将其从非线性可分问题转换为线性可分问题，并使用带有正则项的Logistic回归控制扩展特征带来的复杂性。

1 准备数据

1.1 读取数据

首先使用numpy.loadtxt()从 ex4data.txt 文件中的读取数据。

1.使用loadtxt函数读取数据存于变量data

2.使用变量label储存第三列数据（标签，1表示通过测试，0表示未通过测试）

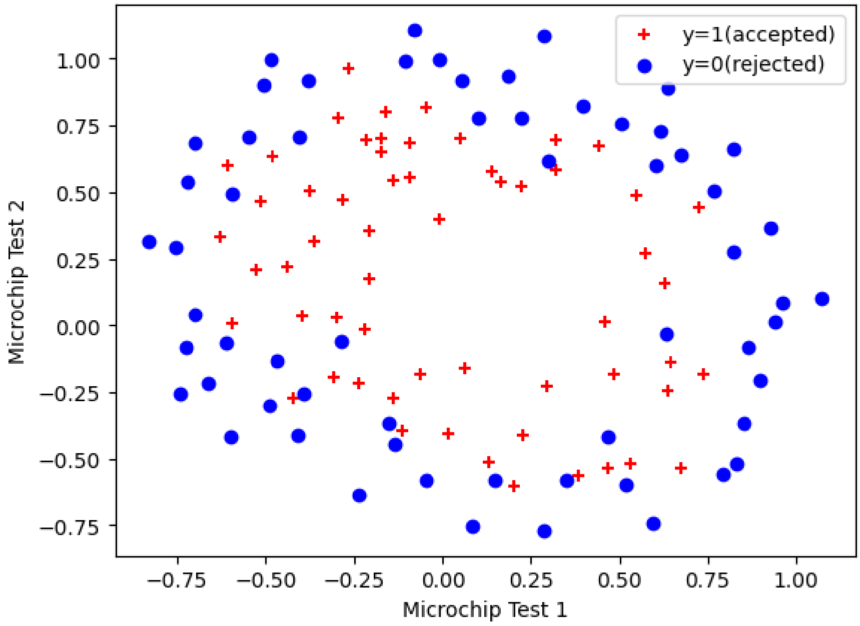
```
data = np.loadtxt()
label = data[:, 2]
```

1.2 可视化数据

跟前次实验相似，对于本次实验的数据， 可以通过plt.plot()函数绘制散点图。具体要求为：分别以两次测试的分数为x、y轴绘制散点图，并使用不同颜色和形状的散点区分正例和反例。

```
pos = 
neg = 

plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')
lgd_pos = plt.scatter(pos[:,0], pos[:,1], marker='+', c='red')
lgd_neg = plt.scatter(neg[:,0], neg[:,1], marker='o', c='blue')
plt.legend(handles=[lgd_pos, lgd_neg], labels=['y=1(accepted)', 'y=0(rejected)'], loc='best')
plt.show()
```

输出的散点图应为：



1.3 使用多项式法扩展特征

面向非线性可分的分类问题，可以通过对输入数据x进行多项式特征扩展。使得模型在高维特征变成线性可分问题。在本实验中，通过实现mapFeature()函数将两维特征值扩展为28维特征（最高的阶数为6）：

$$
\begin{aligned}
&1\\
&x_1, \; x_2\\
&x_1^2, \; x_1x_2, \; x_2^2\\
&x_1^3, \; x_1^2x_2, \; x_1x_2^2, \; x_2^3\\
&x_1^4, \; x_1^3x_2, \; x_1^2x_2^2, \; x_1x_2^3, \; x_2^4\\
&x_1^5, \; x_1^4x_2, \; x_1^3x_2^2, \; x_1^2x_2^3, \; x_1x_2^4, \; x_2^5\\
&x_1^6, \; x_1^5x_2, \; x_1^4x_2^2, \; x_1^3x_2^3, \; x_1^2x_2^4, \; x_1x_2^5, \; x_2^6
\end{aligned}
$$

该部分你需要完成mapapFeature()函数，函数的参数为两列特征值（即数组X的前两列），函数的返回值为构造好的多项式特征数组，我们设定特征最高次幂为6次，因此构造完成后的数组应有28列。使用变量x接收mapapFeature()函数返回数组。

```python
def mapFeature(X1,X2):
    degree = 6 # 每个Featuer的最高次
    '''

    code here

    '''
    return out

X = mapFeature(data[:,[0]].T, data[:,[1]].T)
y = data[:, [2]].T
print(f"X.shape{X.shape}")
```

输出数据X的尺寸应为（28， 118）

2 训练用于非线性可分的Logistic回归模型

首先需要进行数据的准备，包括用于训练的样本x和标签y，初始化输出w数组以及用于正则化的参数lbd（lambda为python的保留字，用于定义匿名函数，因此使用变量名lbd）。

```python
n, m = X.shape
print('n=', n, 'm=', m)
w = np.zeros(n)
lbd = 1
```

2.1 定义Sigmoid函数、带正则项的代价函数costFunction()

costFununction()为带正则项的损失函数。正则项是在机器学习中用于控制模型复杂度的一种技术，通过在损失函数中引入正则化项来约束模型的参数，防止模型过拟合训练数据。

正则化项是在损失函数中添加一个惩罚项，惩罚模型参数的大小，防止模型出现过大的参数值，使得模型更加简单、在学习过程中更加平滑，减少复杂度。公式为：

$$L(\boldsymbol{w})=\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))-(1-y^{(i)})\log(1-f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))\right]+\frac{\lambda}{2m}\sum_{j=1}^{n}\boldsymbol{w}_j^2$$

其中，

$$f_{\boldsymbol{w}}(\boldsymbol{x})=g(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}}$$

```python
def costFunction(w, X, y, lbd):
    w = w[:, np.newaxis]
    fx = sigmoid()
    '''
```

```

    code here

    '''
    return cost.flatten()

print('对初始零向量w求得的cost为',costFunction(w, X, y, lbd))
w = np.ones(n)
print('对初始全一w求得的cost为',costFunction(w, X, y, lbd))
```

cell正确输出:

📌 对初始零向量w求得的cost为 [0.69314718]

对初始全一w求得的cost为 [2.1390856]

2.2 梯度gradient()

gradientnt()为带正则项的梯度计算函数，梯度计算公式为:

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\left(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right) \cdot \boldsymbol{x}_j^{(i)} \right) + \frac{\lambda}{m} \boldsymbol{w}_j$$

其中，

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}}}$$

sigma运算可通过残差 $(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})$ 数组与x做矩阵乘法实现，最后与正则项相加。该步计算得到的grad为28*1的数组。经过flatten()操作后为28维行向量（一维数组）。

```
def gradient(w, X, y, lbd):
    w = w[:, np.newaxis]
    fx = sigmoid()
    '''

    code here

    '''
    return grad.flatten()

w = np.zeros(n)
grad = gradient(w, X, y, lbd)
print('对初始零向量w求得的gradient为', grad)
w = np.ones(n)
grad = gradient(w, X, y, lbd)
```

```python
print('对初始全一w求得的gradient为',grad)
```

cell正确输出：

> 📌 对初始零向量w求得的gradient为 [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]
>
> 对初始全一w求得的gradient为 [0.35451965 0.08508073 0.11852457 0.1505916 0.01591449 0.16811439
>
> 0.06712094 0.03217053 0.02604321 0.10719727 0.09725885 0.01098433
>
> 0.04195657 0.00957212 0.12367776 0.05895534 0.01870409 0.01729323
>
> 0.02352665 0.01513039 0.09858123 0.07328323 0.01051447 0.02270567
>
> 0.00904832 0.02563548 0.00823079 0.10601204]

2.3 使用共轭梯度法最小化带正则项的损失函数，求对应的w（确认x0=w一维数组的影响）

首先确认参数维度是否正确 grad, w.flatten(), X ,y.flatten()

```python
print(np.shape(grad),np.shape(w.flatten()),np.shape(X),np.shape(y.flatten()))
```

cell正确输出：(28,) (28,) (28, 118) (118,)

在这部分，你再次使用scipy.optimize.minimize()自动求取最优参数。使用时主要需要传入的参数为minimize(fun, x0, args=(), method, jac)，minimize()函数对参数的要求如下：

1.fun为进行优化的目标函数，传入需调用的函数名（不需要加括号）。

2.x0即w需传入一维数组。

3.args传入fun需要的其他参数，需用tuple传入。

4.method指定优化算法，此处我们使用method='CG'。

5.jac调用梯度计算函数传入参数需与fun调用函数完全相同，且返回值为一维数组。

高维数组a调整为一维可以使用a.flatten()，该函数会产生一个副本，不会直接改变a的维度

在运行优化函数前首先验证要传入的参数是否符合维度要求。

```
w = np.zeros((n, 1))
result = op.minimize(fun=costFunction, x0=w.flatten(), args=(X, y, lbd), method='CG', jac=gradient)
print(result)
min_w = result.x[:, np.newaxis].flatten()
print(min_w)
```

cell正确输出：

📌 message: Optimization terminated successfully.

success: True

status: 0

fun: 0.5351602541409688

x: [1.142e+00 6.013e-01 ... -1.380e-01 -9.322e-01]

nit: 20

jac: [2.201e-07 -1.701e-06 ... 4.763e-07 -3.204e-06]

nfev: 59

njev: 59

[1.1422266   0.60133262 1.16704491 -1.87184549 -0.91530674 -1.26956018

0.126668   -0.36878573 -0.34528112 -0.17358168 -1.42399329 -0.04879156

-0.60651133 -0.26925627 -1.16308888 -0.24316378 -0.20714934 -0.04338032

-0.28025341 -0.28696345 -0.46919527 -1.03640644 0.02904496 -0.29267072

0.01721546 -0.32898817 -0.13797868 -0.9321943]

2.4 使用模型进行预测

1、使用mapFeature()函数对输入样本进行多项式特征扩展；

2、使用扩展后的特征以及模型min_w进行计算

3、来判别当前芯片是否通过质量测验

❗ 另外，在predict函数中设置断点，然后使用开发环境进行调试（debug the cell），查看运行中的变量取值。

```python
def predict(w, x):
    x_ext = mapFeature(x[0, :], x[1, :])
    fx = w.T @ x_ext
    # 获取输入数据的样本数量
    pred_m = x.shape[1]
    if pred_m == 1 :
        if fx > 0 :
            return 1
        else :

        '''

        code here

        '''

    if pred_m > 1 :
        # 初始化一个全零的数组 pred_rslt 用于存储预测结果
        pred_rslt = np.zeros((1, pred_m))

        # 将 fx 中大于等于 0 的元素对应的预测结果设为 1
        # 将 fx 中小于 0 的元素对应的预测结果设为 0
        '''

        code here

        '''

    return pred_rslt

chipTest = np.array([[0, 0.6]]).T
predRslt = predict(min_w, chipTest)
print(predRslt)
```

cell正确输出：

📌 1

2.5 绘制决策边界

通过训练你已经得到最佳的参数，存放于min_w中。现在可以利用min_w绘制决策边界。在线性分类问题中，决策边界的方程为：$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$，我们可以通过给代值计算，然后连线画图。但是对于非线性分类问题，决策边界为复杂的多项式方程（$\theta^T x = 0$），难以通过代值计算绘制。我们采取如下策略：

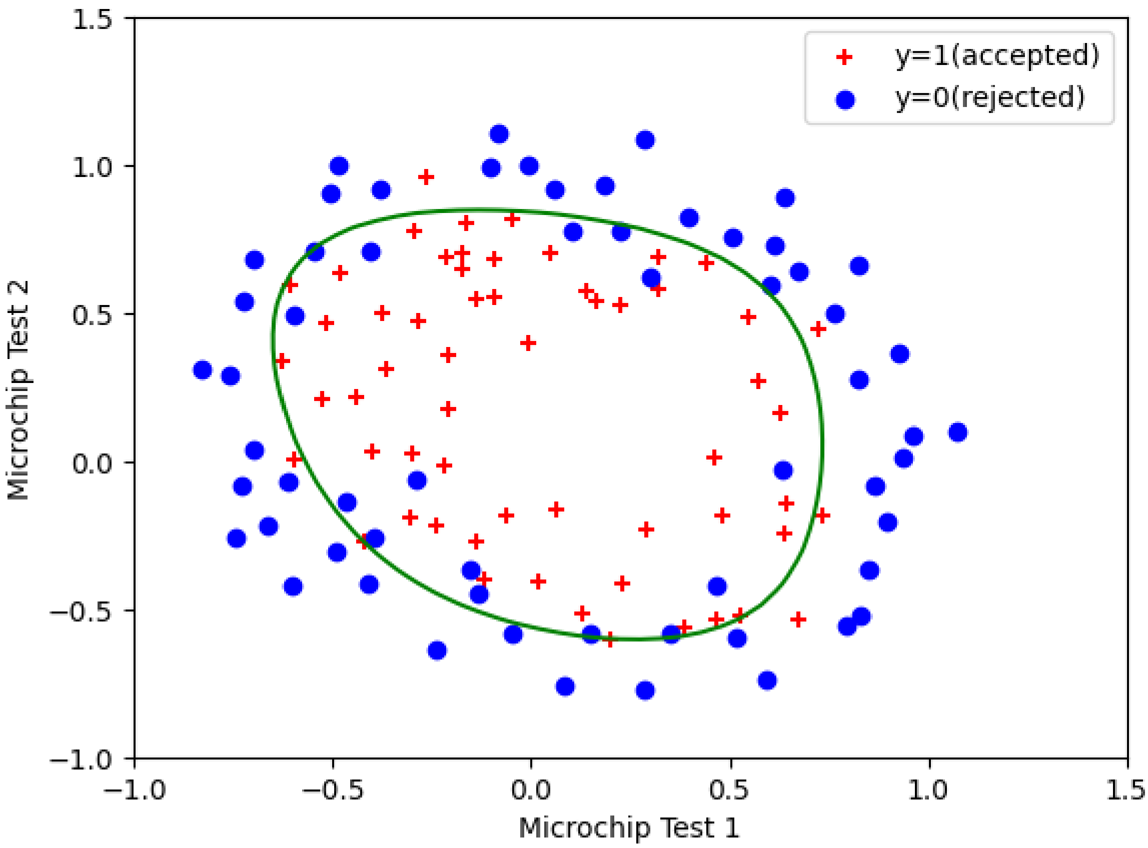
首先我们依然画出数据集的散点图。注意到和的取值范围集中在-1~1.5之间，我们可以想象在x和y坐标取值均属于-1~1.5区间内的平面里之间建立一个50*50（或其他维度）的二维网格，其中每个网格点的x、y坐标分别对应特征值和，我们计算每个网格点的 $z = \theta^T x$，最后绘制z = 0的等高线，便有了决策边界。

在-1~1.5区间内选取50个值可以使用numpy.linspace()函数，该函数使用简单grid_x1 = np.linspace(-1, 1.5, 50)即会在-1~1.5区间内等间隔选取50个数生成一维数组。（numpy.linspace — NumPy v1.26 Manual）

绘制等高线可以使用函数matplotlib.contour()函数。给函数传入的参数包括x坐标、y坐标、高度值数组以及指定的高度绘制等高线。（matplotlib.pyplot.contour — Matplotlib 3.8.3 documentation）。该部分代码在下面给出，请参考官网文档理解。

```python
# 1. 绘制所有样本
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')

lgd_pos = plt.scatter(pos[:,0], pos[:,1], marker='+', c='red')
lgd_neg = plt.scatter(neg[:,0], neg[:,1], marker='o', c='blue')
plt.legend(handles=[lgd_pos, lgd_neg], labels=['y=1(accepted)', 'y=0(rejected)'], loc='best')

# 2. 用等高线绘制
plot_x = np.linspace(-1, 1.5, 50)
plot_y = np.linspace(-1, 1.5, 50)
boundary = np.zeros((plot_x.size,plot_x.size))

for i in range(plot_x.size):
    for j in range(plot_y.size):
        boundary[i][j] = np.dot(mapFeature(plot_x[i], plot_y[j]).T, min_w)
boundary = boundary.T
plt.contour(plot_x, plot_y, boundary,[0], colors='green' )

plt.show()
```

绘制决策边界图为：



2.6 计算模型的训练准确率

在这部分你需要编写函数计算模型在训练集上的正确率，注意预测值z >0则为正例,反之则为负例。模型预测分类结果与样本真实标签相同则为预测准确（注意正例负例均可预测准确！），正确率应该约为82%（lbd=1时）。

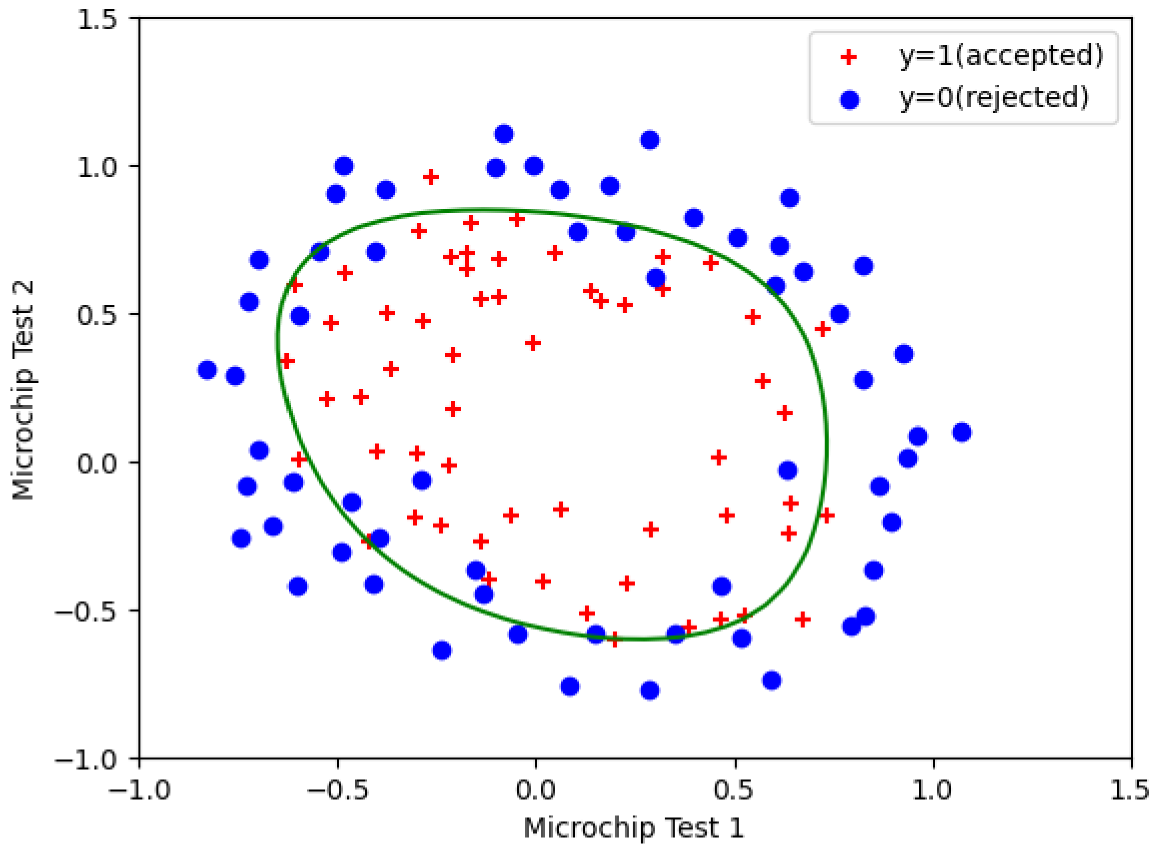
```
pred_rslt = predict(min_w, data[:,0:2].T) # data[:, [0, 1]]
print(np.mean(pred_rslt == y))
```

cell正确输出：

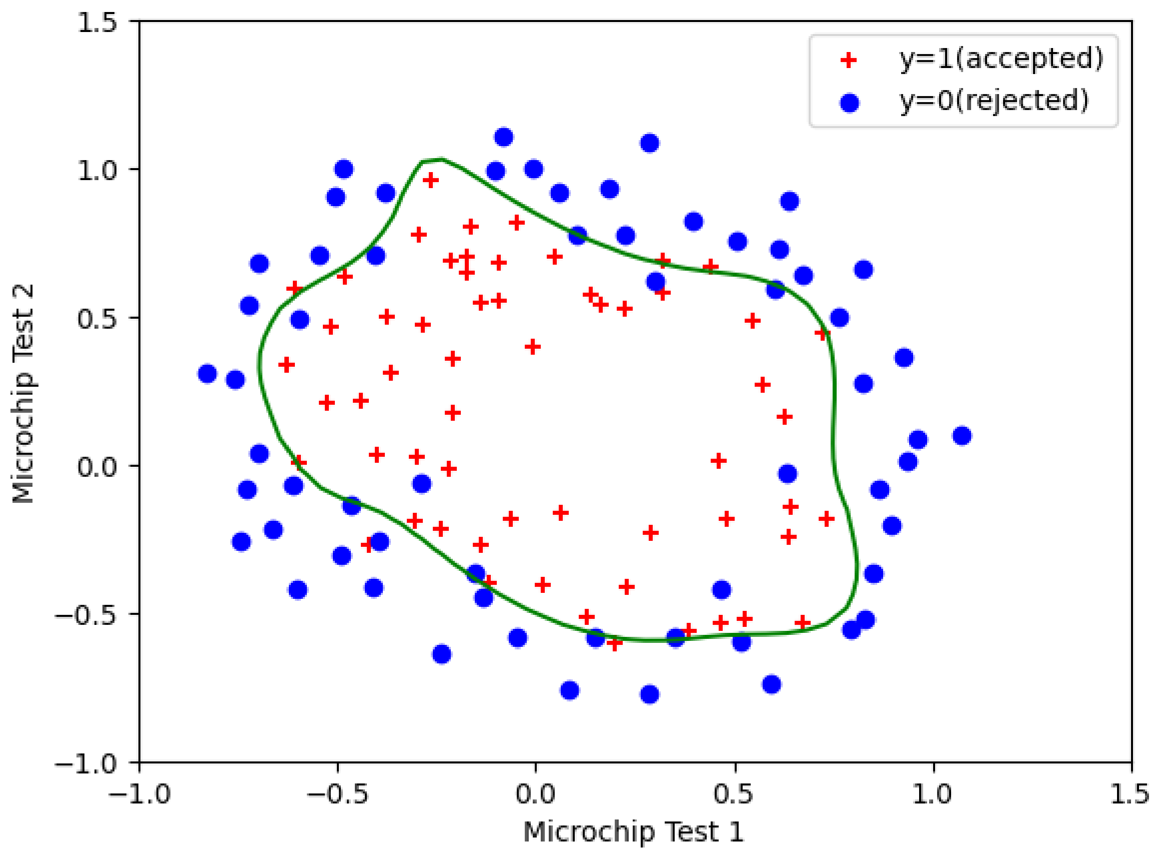
> 📌 0.8220338983050848

2.7 关于正则项作用的实验

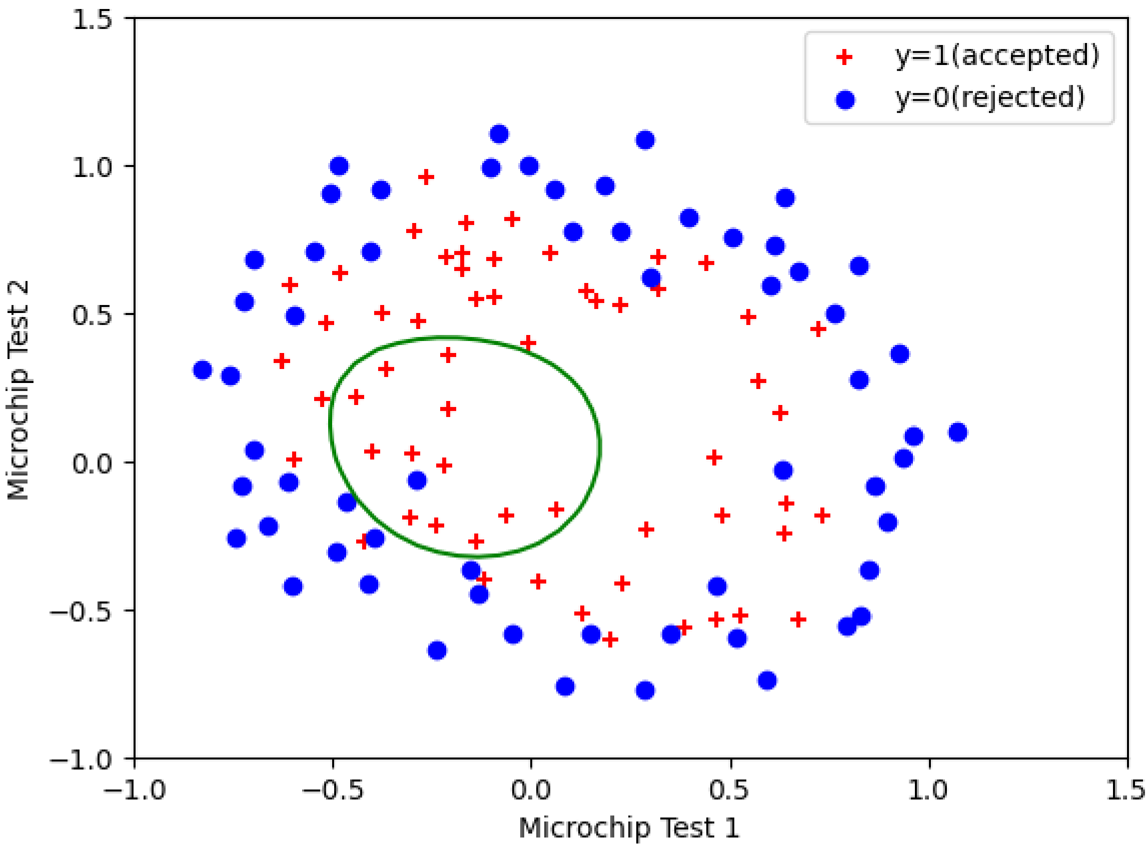
令lambda = 0， 1 和100。然后绘制模型边界



当lambda=1时的判决边界



当lambda=0时的判决边界，此种情况下正则项不起作用，但特征被多项式法扩展到28维使得模型过于复杂，此种情况被称为过拟合（overfitting）



当lambda=100时的判决边界。此种情况下正则项起作用大，训练误差起作用相对小。此种情况被称为欠拟合（underfitting）。