实验 2：Logistic回归

实验目的

1、实现正则化的Logistic回归算法
2、实现多项式特征构造
3、实现正则化的损失函数和梯度计算

实验数据

1. ex2data1.txt-用于线性分类的数据集（高校录取预测）
2. ex2data2.txt-用于非线性分类的数据集（芯片质量预测）

实验步骤

1. 线性分类问题

在上次实验中，你已经通过建立Logistic模型解决高校录取预测问题。

2. 非线性分类问题

在本部分的练习中，你将使用正则化的Logistic回归模型来预测一个制造工厂的微芯片是否通过质量保证（QA）。假设你是这个工厂的产品经理，你希望确定这些芯片是否通过测试。你有一些以前芯片的测试情况数据集，可以作为训练Logistic模型的训练集。

在文件ex2data2.txt 中包含了我们本次非线性分类实验的数据集，数据共三列：每行表示一个芯片经过测试的的历史数据，前两列为两项测试的分数情况，；第三列为标签，1表示能够通过质量测试，0表示不能通过。

2.1 读取数据

首先你需要做的是将ex2data2.txt 文件中的数据进行读取，使用的方法是numpy.loadtxt()，具体要求如下：

1.使用loadtxt函数读取数据存于变量ex2data2

2.使用变量X储存ex2data2的前两列数据（芯片的两项测试结果）

3.使用变量y储存ex2data2的第三列数据（标签，1表示通过测试，0表示不能通过）

4.使用变量m储存样本数量

代码：

```
ex2data2 = np.loadtxt('.\ex2data2.txt',delimiter=',')
X = ex2data2[:,0:2]
y = ex2data2[:,[2]]
m = np.shape(y)[0]
```

cell正确输出：(118, 2) (118, 1)

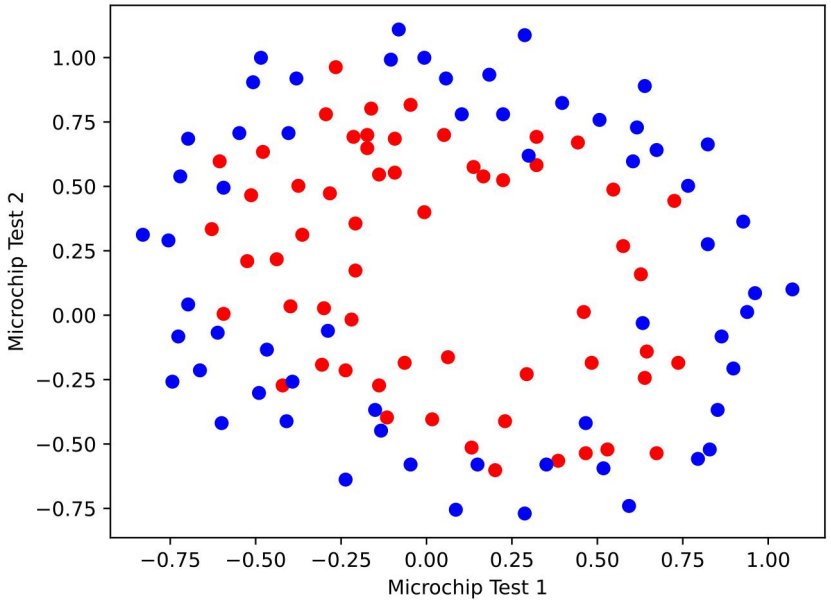
2.2 可视化数据

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。

代码：

```
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')
for i in range(m):
    if y[i] == 1: #正例
        plt.plot(X[i,0], X[i,1], 'or')
    elif y[i] == 0: #反例
        plt.plot(X[i,0], X[i,1], 'ob')
```

cell正确输出：



2.3 训练Logistic回归模型

该部分你要完成数据预处理（构造多项式特征）、定义正则化的损失函数和梯度，然后继续使用scipy.optimize.minimize()函数自动求取theta最优解。

2.3.1 数据预处理：多项式特征构造，初始化 θ ，初始化正则化参数lambda

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theta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）。

1.多项式特征构造：解决非线性分类问题需要对输入数据x进行多项式特征构造（该操作也称为特征映射）。其作用是扩展原始特征空间，使得模型能够更好地拟合非线性关系。由两个特征值进行特征映射的方式如下：

$$mapFeature(x_1, x_2) = [1, x_1, x_2, {x_1}^2, x_1x_2, {x_2}^2, \dots, x_1{x_2}^{n-1}, {x_2}^n]$$

通过将原始特征的组合形式引入模型，可以增加模型的表达能力，使其能够更好地捕捉特征之间的复杂关系。在 logistic 回归等线性模型中，引入多项式特征可以将原本线性不可分的问题转化为线性可分的问题，从而提高模型的分类性能。

该部分你需要完成mapFeature()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mapFeature()函数返回数组。

2.前面已经使用m存放样本个数，y存放标签。

3.使用np.zeros初始化theta,规模为np.shape(x)[1]*1。

4.初始化正则化参数lbd为1。

Cell正确输出：(118, 28) (28, 1)

2.3.2 定义正则化的损失函数、梯度

sigmoid函数已经写好，在这部分需要你完成costFunction()、gradient()两个函数。

1.costFunction()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：

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

其中，$h_\theta(\boldsymbol{x}^{(i)}) = g(\boldsymbol{\theta^T x}) = \frac{1}{1+e^{-\boldsymbol{\theta^T x}}}$，计算时注意矩阵乘法的维度要求。

2.gradient()为带正则项的梯度计算函数，梯度计算公式为：

$$grad = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)})\,x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

其中，$h_\theta(\boldsymbol{x}^{(i)}) = g(\boldsymbol{\theta^T x}) = \frac{1}{1+e^{-\boldsymbol{\theta^T x}}}$，sigma运算可通过残差（$h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}$）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）。

cell正确输出： 对初始零向量theta求得的cost为 [0.69314718]

梯度为:

[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]

2.3.3 使用scipy.optimize.minimize求最小损失对应的参数theta

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theta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='TNC'。

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。

Cell正确输出：1 1 1

Cell正确输出：

fun: 0.5351602547815488
jac: array([-1.87196705e-05, 4.68710496e-06, -3.22746886e-06, -2.88545190e-06,
3.22080864e-06, -4.16511723e-06, 3.53446066e-06, 2.08482022e-06,
3.39384310e-06, -3.63321242e-06, 1.02078286e-06, 2.48105559e-06,
4.99246964e-07, 2.44646750e-06, -3.02310657e-06, 5.07322066e-06,
1.02969897e-06, 1.63994131e-06, 1.60351773e-06, -4.33675488e-07,
-2.05599084e-06, -7.27288908e-07, -1.10700995e-06, 9.60549279e-07,
-1.34889142e-06, 1.26133814e-06, 6.59071689e-07, -8.67956174e-07])
message: 'Converged (|f_n-f_(n-1)| ~= 0)'
nfev: 26
nit: 6
status: 1
success: True
x: array([1.14201564, 0.60123715, 1.16715808, -1.87180902, -0.91567126,
-1.26944012, 0.12678676, -0.36850105, -0.34494249, -0.17391075,
-1.4237067 , -0.04838577, -0.60631732, -0.26916587, -1.16320191,
-0.24269702, -0.20697485, -0.04305267, -0.28008544, -0.28708932,
-0.46912457, -1.03629767, 0.02904379, -0.29250729, 0.01716375,
-0.32880515, -0.13795624, -0.93187563])

2.4 评估Logistic回归模型

2.4.1 绘制决策边界

通过训练你已经得到最佳的参数，存放于theta_star中。现在可以利用theta_star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$x_1$代值计算$x_2$，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（$\boldsymbol{\theta^T x} = 0$），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x_1和x_2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x_1和x_2，我们计算每个网格点的$\boldsymbol{z} = \boldsymbol{\theta^T x}$，最后绘制$\boldsymbol{z} = 0$的等高线，便有了决策边界。

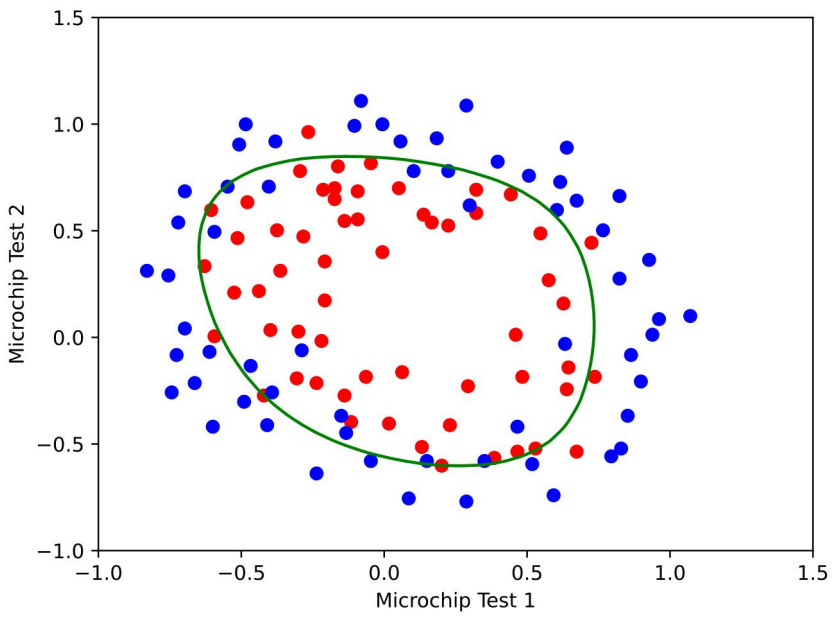
在-1~1.5区间内选取50个值可以使用numpy.linspace()函数，该函数使用简单grid_x1 = np.linspace(-1, 1.5, 50)即会在-1~1.5区间内等间隔选取50个数生成一维数组。（numpy.linspace — NumPy v1.26 Manual）

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。

代码

```
plot_x1 = np.linspace(-1, 1.5, 50)
plot_x2 = np.linspace(-1, 1.5, 50)
x_theta = np.zeros((plot_x1.size,plot_x2.size))
for i in range(plot_x1.size):
    for j in range(plot_x2.size):
        x_theta[i][j] = np.dot(mapFeature(plot_x1[i], plot_x2[j]), theta_star)
x_theta = x_theta.T#由于contour()要求高度值数组的行数对齐y坐标，列数对其x坐标，因此需要转置。
plt.contour(plot_x1, plot_x2, x_theta,[0], colors='green' )
```

cell正确输出：



2.4.2 计算模型准确率

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$h_\theta(x^{(i)})>0.5$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！），你的正确率应该约为81%（λ=1时）。