实验3. Logistic回归（线性）

实验目的

1. 实现Logistic回归算法
2. 实现和sigmoid函数、基于交叉熵的损失函数和梯度计算
3. 调用最小化函数实现梯度下降算法

实验数据

ex3data.txt-用于线性分类的数据集（高校录取预测）

数据共三列：每行表示一个申请人的历史数据，前两列为申请人的两门考试成绩；第三列为录取结果，1表示能够入学录取(admitted)，0表示不录取（not admitted）。

实验步骤

线性分类问题

目标任务是根据学生两门考试的成绩来判定是否被大学录取。在数据分布上大致是线性可分问题。在实验中使用Logistic回归作为线性分类模型。

1. 数据准备

1.1 读取数据

首先你需要做的是将 ex3data1.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：

a. 使用loadtxt函数读取数据存于变量data，注意指定分隔符参数。
b. 使用变量X储存ex3data1的前两列数据（申请人的两门考试成绩）。
c. 使用变量y储存ex3data1的第三列数据（标签，1表示能够入学，0表示不能入学）。
d. 使用变量m,n储存样本数量。

代码：

```
data =
X =
```

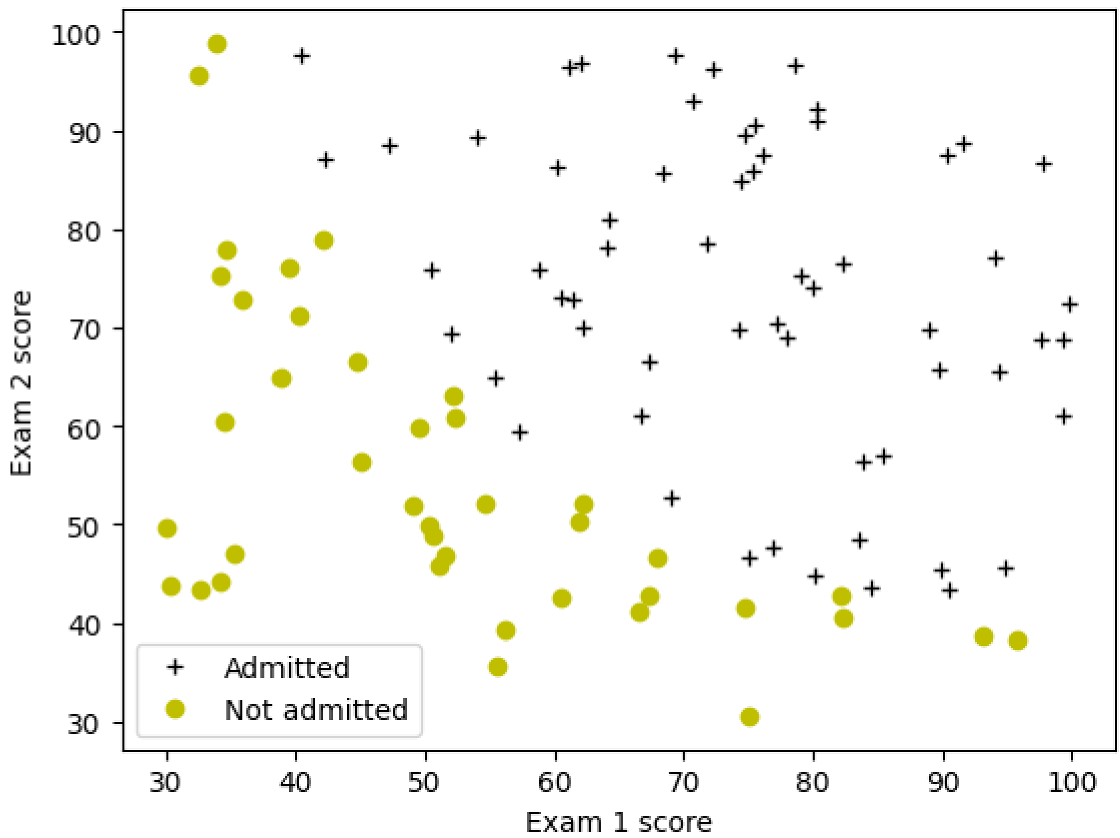
```
y =     # 此处返回长度为m的一维数组，没有将其转化为m*1的二维数组。原因在于pos=X[y==1,:]
m, n = X.shape
```

1.2 绘制散点图

对数据进行可视化有助于更好的理解数据集的分布，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：分别以两次考试的成绩为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。比如被录取（y标签为1）则点显示为黑色“+”点，未被录取（y标签为0）则点显示为黄色“o”点。

```
pos = X[data[:,2] == 1, :]
neg = X[data[:,2] == 0, :]
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.plot(pos[:,0], pos[:,1], '+k', label='Admitted')
plt.plot(neg[:,0], neg[:,1], 'oy', label='Not admitted')
plt.legend()
plt.show()
```

cell正确输出：



1.3 数据预处理

1. 为输入数据X加全1列（X0）
2. X，y做好转置，转置后的y为1*m的向量

一种方式使用上一节的拼接方式

```
X = np.hstack((np.ones((m,1)), data[:,0:2])).T
y = np.hstack((np.ones((m,1)), data[:,[0,1]])
```

另一种可使用附加函数numpy.append()

```
X = np.append(np.ones((m,1)), data[:,0:2], axis=1).T
```

2. 实现基于交叉熵的损失函数

2.1 定义sigmoid函数

【Sigmoid函数】完成sigmoid()函数，函数的定义如下式。其中，自然指数计算使用numpy.exp()来计算，参数可以是数值也可以是多维数组。

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

```
# 利用实验说明书中的公式定义sigmoid函数
# 该sigmoid函数可以接受单个数值和向量作为参数
def sigmoid(z):
    g =              # 使用np.exp()进行自然指数运算
    return g
```

对函数进行测试

```
print(sigmoid(0), sigmoid(-100),sigmoid(10))
tmp = np.array([-5, -2, 0, 1, 3])
print(sigmoid(tmp))
```

🥇 输出结果

0.5 3.7200759760208356e-44 0.9999546021312976

[0.00669285 0.11920292 0.5   0.73105858 0.95257413]

2.2 实现损失函数、梯度

1. 【代价函数（损失函数）】Logistic回归的代价函数costFunction()采用交叉熵来度量输出与标注之间的误差，具体公式如下。其中，对数函数可使用numpy.log()来计算，参数可以是数值也可以是多维数组。

$$L(\boldsymbol{w})=\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log_2\left(f_{\boldsymbol{w}}\left(\boldsymbol{x}^{(i)}\right)\right)-\left(1-y^{(i)}\right)\log_2\left(1-f_{\boldsymbol{w}}\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

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

2. 【代价函数的梯度】gradient()

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

使用np.zeros初始化w，尺寸为(n+1)*1（每个样本有两个特征值，此外，因为w0被纳入w中，使得X额外加了一维数值X0=1）。

需要说明的是gradient()会供后续的最小化函数使用，按照minimize()函数对参数的要求，应返回一维数组。

```
grad = grad.flatten() # 返回一维数组
```

使用如下两组对两个函数进行测试

```
w = np.zeros((n + 1,1))
[cost, grad] = [costFunction(w, X, y),gradient(w, X, y)]
```

costFunction()应该为0.6931；gradient结果应该为[-0.1, -12.0092, -11.2628]。

```
test_w = np.array([-24, 0.2, 0.2]).T
[test_cost, test_grad] = [costFunction(test_w, X, y),gradient(test_w, X, y)]
```

costFunction()应返回0.2183；gradient()结果应该为[0.0429, 2.566, 2.6468]。

2.3 使用scipy.optimize.minimize()最小化损失函数

为最小化上一节给出的代价函数，可以继续使用梯度下降法。此处，采用scipy.optimize.minimize()最小化代价函数，进而获得最优参数。使用时主要需要传入的参数为minimize(fun, x0, args=(), method, jac)，minimize()函数对参数要求很严格，具体要求如下：

1.fun：要被最小化的目标函数。具体到本节的实现代码，此处传入的是Logistic回归的代价函数costFunction（此处不需要加括号）。

2.x0：目标函数fun的参数列表。此处传入的是模型参数w。需要注意的是此处的要求是传入一维数组，因此，此处尺寸为(n+1*1)的w需通过flatten()函数转为长度为n+1的一维数组。

3.args：目标函数fun的其他参数。此处传入的是costFunction()的第2和第3个参数，即训练数据集的输入X和输出y。形式上，对于多个参数需用tuple封装这些参数后传入。

4.method：指定优化算法，此处我们使用收敛速度快于梯度下降法的共轭梯度法。method='CG'。

5.jac：用于传入计算梯度的函数。此处的梯度计算函数的参数需与fun对应的代价计算函数完全相同，且返回值为一维数组。所传入的gradient()函数返回的梯度也通过flatten()转换成一维数组。

> 需要说明的是SciPy使用的数据结构以一维数组为主，因此，我们(n+1)*1维的参数w需要flatten成长度为n+1的一维数组。

```python
result = op.minimize(fun=costFunction, x0=w.flatten(), args=(X, y.flatten()), method='CG', jac=gradient)
print(result)
min_w = result.x
print(min_w)
```

在代价函数被最小后，从返回结果中提取模型参数至min_w中，这段程序运行正确后应输出：

📍 message: Optimization terminated successfully.

success: True

status: 0

fun: 0.2034977314878733

x: [-2.518e+01 2.063e-01 2.016e-01]

nit: 53

jac: [-4.252e-06 5.238e-06 -9.691e-06]

nfev: 121

njev: 121

min_w= [-25.17551603 0.20634537 0.20158612]

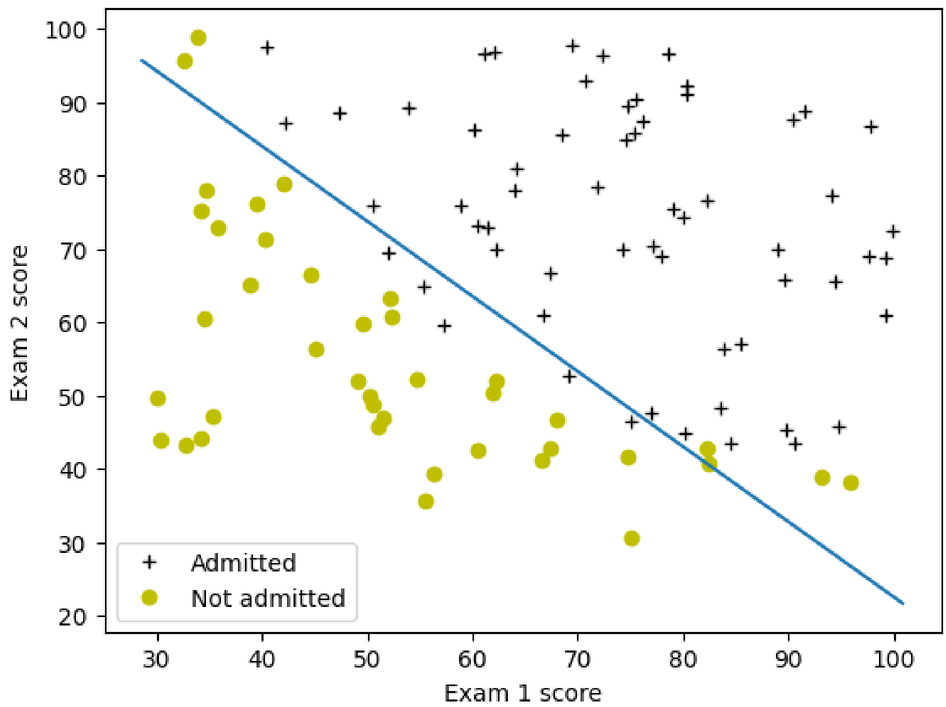
2.4 使用Logistic回归模型进行预测

2.4.1 绘制决策边界曲线

通过训练已得到最佳模型参数，并存放于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：w0+w1*x1+w2*x2 = 0。

首先我们依然画出数据集的散点图，然后通过带入两个x1的值计算得到x2，由于我们的图像是以x1和x2为x、y轴做出，因此相当于得到了图像上的两个点，连线画出决策边界即可。注意两个x1的选取尽量在x1的最大值和最小值以外，这样使用plot连线的长度足以区分所有散点。

cell正确输出：



2.4.2 使用Logistic回归模型进行分类预测

某考生在考试1中得45分，在考试2中得85分。用训练得到的min_w，预测该生被录取的可能

```
score = np.array([1, 45, 85])
prob = sigmoid(np.dot(score, min_w)) # 此处score和min_w都是一维数组，可直接使用该函数计算点积
```

运行结果应为:0.7764

2.4.3 计算训练准确率

在这部分你需要计算模型在训练集上的准确率，决策边界的方程为：w0+w1*x1+w2*x2 = 0，我们可以令z=w0+w1*x1+w2*x2 = 0，当z>0为正例，z<0为负例。

请完成predict(w, X)函数对所有训练样本进行预测。并使用下面的代码来计算训练数据集上的准确率。

```
# 定义一个预测函数，用于根据给定的权重向量 w 和特征矩阵 X 进行预测
# w 是逻辑回归模型训练得到的权重向量
```

```python
# X 是特征矩阵，每一列代表一个样本，每一行代表一个特征
def predict(w, X):
    # 创建一个形状为 (m, 1) 的零矩阵 admitRslt，用于存储预测结果

    # 计算线性组合 z，即 z = w^T * X

    # 分别找出线性组合 z 中大于等于 0 和小于 0 的元素的索引

    # 找出对应的样本正负例

    # 返回预测结果矩阵
        return admitRslt
```

```python
admit = predict(min_w, X)
np.mean(admit == y.T)
```

正确结果为89%。