机器学习练习 2- 逻辑回归 Logistic Regression

本次实验我们将一步步引导大家熟悉逻辑回归算法。并基于该方法，根据一个学生的成绩，判断该生是否能够入学。在本练习中，我们还将尝试正则化方法。
注意，逻辑回归是一个分类算法。

逻辑回归Logistic regression

在这个练习的第一部分，我们将建立一个逻辑回归模型来预测学生是否考上大学。假设你是一个大学系的管理员，你想根据两个考试的结果确定每个申请人的录取机会。您可以将以前申请者的历史数据用作逻辑回归的培训集。对于每个培训示例，您都有申请人的两次考试成绩和录取决定。为了实现这一点，我们将建立一个分类模型，根据考试成绩估计入学概率。

我们首先来观察一下数据

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
path = 'ex2data1.txt'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2', 'Admitte
data.head()
```

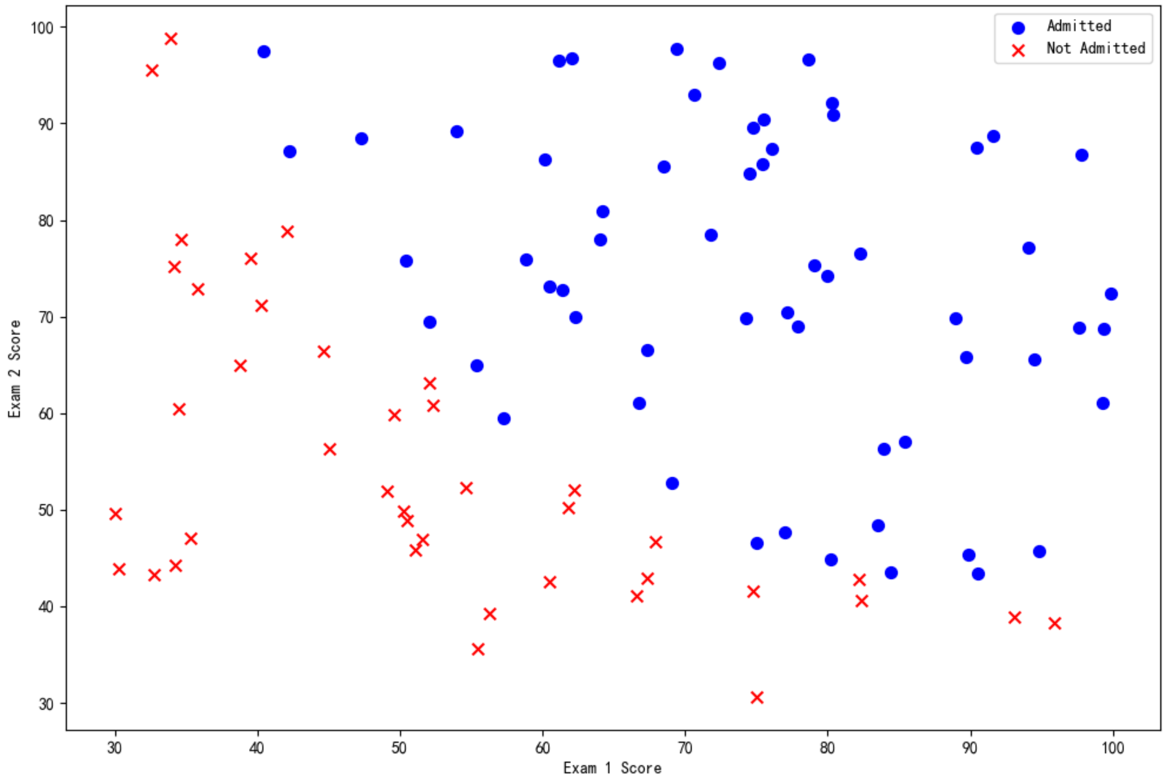
Out[ ]:

	Exam 1	Exam 2	Admitted
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

下面我们创建一个两个分数的散点图，并使用颜色编码来可视化示例是阳性（允许）还是阴性（不允许）。

In [ ]:
```
positive = data[data['Admitted'].isin([1])]
negative = data[data['Admitted'].isin([0])]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='c
ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')
```

Out[ ]: Text(0, 0.5, 'Exam 2 Score')

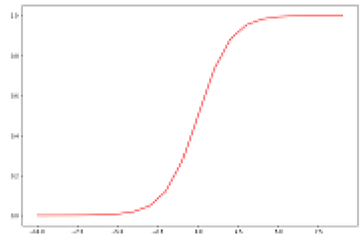


从上图中可以看到，在两个类之间似乎有一个明确的决策边界。现在我们利用逻辑回归来寻找这个边界

下面的代码用于定义sigmoid()函数。请将该函数补充完整。

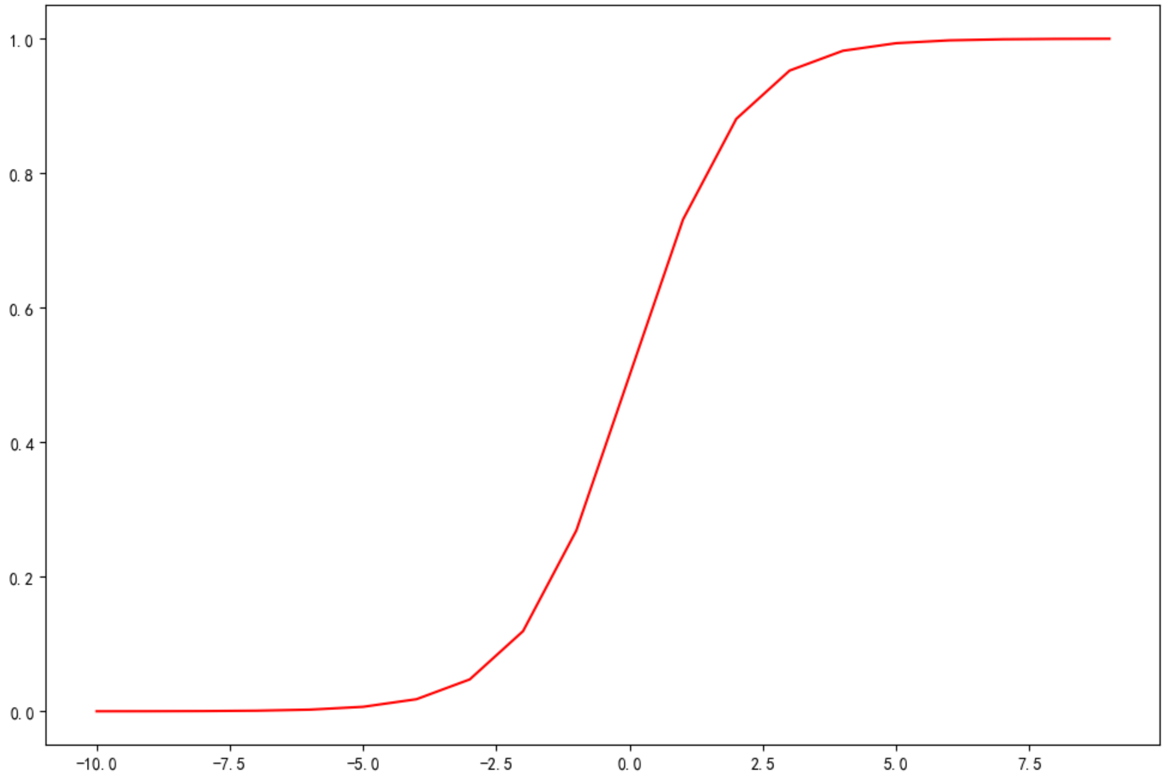
In [ ]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

如果sigmoid()函数定义正确，那么运行下面的代码后，你将看到如下的图：



In [ ]:
```
nums = np.arange(-10, 10, step=1)

fig, ax = plt.subplots(figsize=(12,8))
ax.plot(nums, sigmoid(nums), 'r')
```

Out[ ]: [<matplotlib.lines.Line2D at 0x16ddb2eb0>]



接着，请定义代价函数cost()，这里theta就是参数，X是输入样本，y是输出样本，函数的返回值就是对给定的theta参数，模型定义的代价函数的误差。注意，这里的X已经包括了全1的特征偏置项。

In [ ]:
```
def cost(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
    second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
    return np.sum(first - second) / (len(X))
```

In [ ]:
```
# add a ones column - this makes the matrix multiplication work out easie
data.insert(0, 'Ones', 1)

# set X (training data) and y (target variable)
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]

# convert to numpy arrays and initalize the parameter array theta
X = np.array(X.values)
y = np.array(y.values)
theta = np.zeros(3)
```

通过shape查看一下各个变量是否正确

In [ ]:
```
X.shape, theta.shape, y.shape
```

Out[ ]:
```
((100, 3), (3,), (100, 1))
```

这里将调用你定义的代价函数，如果函数定义正确，那么你将看的输出为：
0.6931471805599453

In [ ]:
```
cost(theta, X, y)
```

Out[ ]:
```
0.6931471805599453
```

gradient（）函数用于计算梯度值。请将该部分代码补全

In [ ]:
```
def gradient(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)

    parameters = int(theta.ravel().shape[1])
    grad = np.zeros(parameters)

    error = sigmoid(X * theta.T) - y

    for i in range(parameters):
        grad[i] = np.sum(np.multiply(error, X[:, i])) / len(X)


    return grad
```

请注意，在这个函数中，我们实际上并没有执行梯度下降-我们只计算一个梯度步长。在下面的实验中，我们实际将使用python中的optimize优工具库优化给定函数的参数，计算成本和梯度。

下述代码将调用你定义的gradient（）函数，这里theta均被初始化为0。如果gradient（）设计正确，那么你将看到的输出为：array([-0.1 , -12.00921659, -11.26284221])

In [ ]:
```
gradient(theta, X, y)
```

Out[ ]:
```
array([ -0.1        , -12.00921659, -11.26284221])
```

现在我们将实验 SciPy中的 truncated newton (TNC) 来完成参数最优化工作

In [ ]:
```
import scipy.optimize as opt
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X, y))
result
```

```
  NIT   NF   F                       GTG
    0    1  6.931471805599453E-01   2.71082898E+02
    1    3  6.318123602631536E-01   7.89087138E-01
    2    5  5.892425222593011E-01   7.39226590E+01
    3    7  4.227824082768085E-01   1.85265802E+01
    4    9  4.072926971534283E-01   1.68671130E+01
    5   11  3.818854920309407E-01   1.07735097E+01
    6   13  3.786234896709935E-01   2.31584926E+01
tnc: stepmx = 1000
    7   16  2.389268303582261E-01   3.00822039E+00
    8   18  2.047203891858869E-01   1.52227714E-01
    9   20  2.046713899360368E-01   6.62495142E-02
   10   22  2.035303163190396E-01   9.30780772E-04
tnc: fscale = 32.7775
   11   24  2.035293522100511E-01   8.07207683E-06
   12   26  2.035251114039714E-01   1.80213850E-04
   13   28  2.034984103693545E-01   5.02836184E-04
   14   30  2.034978377466289E-01   9.88454531E-06
   15   32  2.034977904843622E-01   3.76915430E-06
   16   34  2.034977386092095E-01   1.93988086E-05
   17   36  2.034977015894744E-01   2.42606408E-13
tnc: |pg| = 1.50271e-08 -> local minimum
   17   36  2.034977015894744E-01   2.42606408E-13
tnc: Local minima reach (|pg| ~= 0)
```

Out[ ]:
```
(array([-25.16131867,   0.20623159,   0.20147149]), 36, 0)
```

我们可以使用优化后的参数计算代价

In [ ]:
```python
cost(result[0], X, y)
```

Out[ ]:
```
0.2034977015894744
```

接下来，我们需要编写一个函数，该函数将使用优化后的参数θ为数据集X输出预测。然后，我们可以使用此函数对分类器的训练精度进行评分。
如果代码运行正确，则程序输出精度为89%。

In [ ]:
```python
def predict(theta, X):
    probability = sigmoid(X * theta.T)
    return [1 if x >= 0.5 else 0 for x in probability]
```

In [ ]:
```python
theta_min = np.matrix(result[0])
predictions = predict(theta_min, X)
correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (
accuracy = (sum(map(int, correct)) % len(correct))
print ('accuracy = {0}%'.format(accuracy))
```

```
accuracy = 89%
```

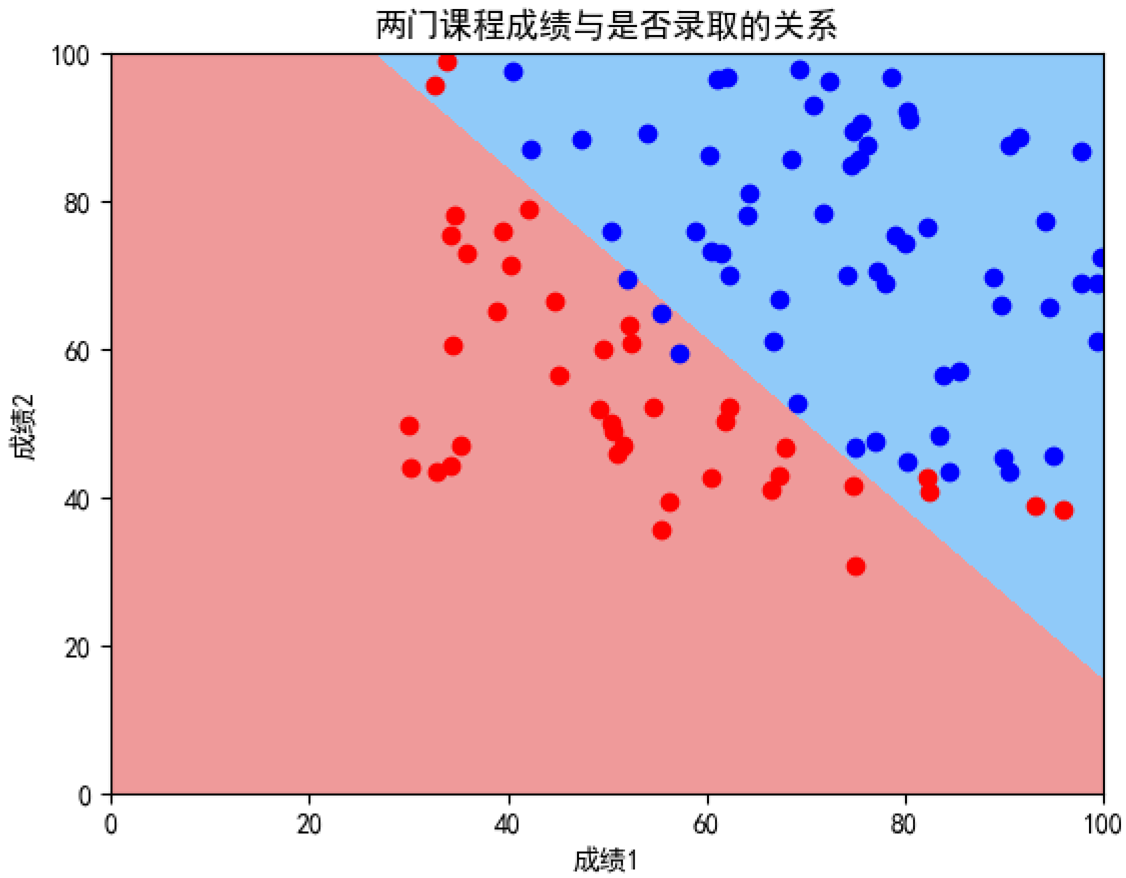
基于sklearn 实现逻辑回归 Logistic regression

In [ ]:
```python
from sklearn.model_selection import train_test_split
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
```

```python
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

# 数据格式：成绩1,成绩2,是否被录取（1代表被录取，0代表未被录取）

# 读取数据
data = np.loadtxt('ex2data1.txt', delimiter=',')
data_X = data[:, 0:2]
data_y = data[:, 2]
```

In [ ]:
```python
# 函数（画决策边界）定义
def plot_decision_boundary(model, axis):
    x0, x1 = np.meshgrid(
        np.linspace(axis[0], axis[1], int((axis[1] - axis[0]) * 100)).res
        np.linspace(axis[2], axis[3], int((axis[3] - axis[2]) * 100)).res
    )
    X_new = np.c_[x0.ravel(), x1.ravel()]

    y_predict = model.predict(X_new)
    zz = y_predict.reshape(x0.shape)

    custom_cmap = ListedColormap(['#EF9A9A', '#FFF59D', '#90CAF9'])
    plt.contourf(x0, x1, zz, cmap=custom_cmap)
```

In [ ]:
```python
# 数据分割
X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, rando

# 训练模型
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# 结果可视化
plot_decision_boundary(log_reg, axis=[0, 100, 0, 100])
plt.scatter(data_X[data_y == 0, 0], data_X[data_y == 0, 1], color='red')
plt.scatter(data_X[data_y == 1, 0], data_X[data_y == 1, 1], color='blue')
plt.xlabel('成绩1')
plt.ylabel('成绩2')
plt.title('两门课程成绩与是否录取的关系')
plt.show()
```



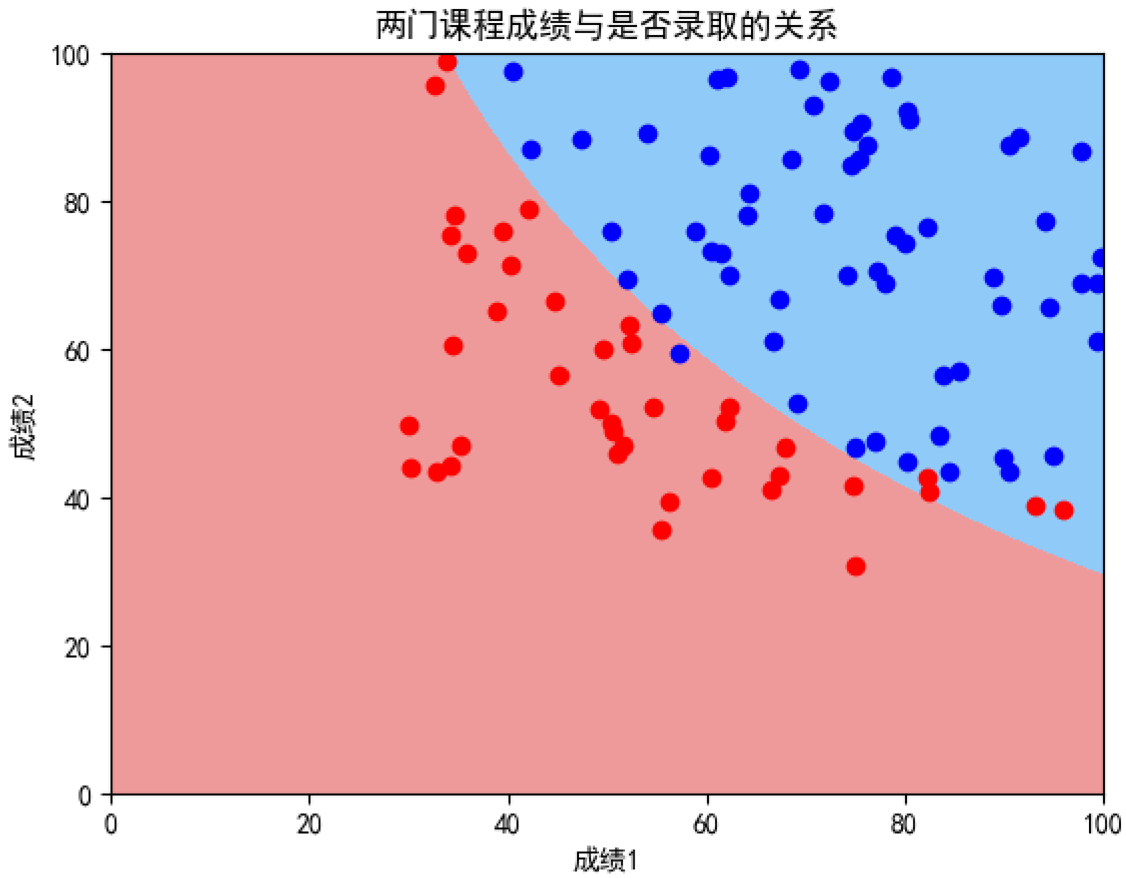
In [ ]:
```
# 模型测试
print(log_reg.score(X_train, y_train))
print(log_reg.score(X_test, y_test))
```

```
0.9066666666666666
0.92
```

接下来，请利用特征工程，构造多项式特征，提高分类精度

In [ ]:
```
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


# 读取数据
path = 'ex2data1.txt'
data = np.loadtxt(path, delimiter=',')
data_X = data[:,:2]
data_y = data[:,2]


# 多项式特征
def PolynomialLogisticRegression(degree):
    return Pipeline([
```

```
        # first step: 给样本特征添加多项式项
        ('poly', PolynomialFeatures(degree=degree, include_bias=False, ir
        # second step: 数据归一化处理
        ('std_scaler', StandardScaler()),
        # third step: 逻辑回归
        ('log_reg', LogisticRegression())
    ])


# 画决策边界
def plot_decision_boundary(model, axis):
    x0, x1 = np.meshgrid(
        np.linspace(axis[0], axis[1], int((axis[1] - axis[0]) * 100)).res
        np.linspace(axis[2], axis[3], int((axis[3] - axis[2]) * 100)).res
    )
    X_new = np.c_[x0.ravel(), x1.ravel()]

    y_predict = model.predict(X_new)
    zz = y_predict.reshape(x0.shape)

    custom_cmap = ListedColormap(['#EF9A9A', '#FFF59D', '#90CAF9'])
    plt.contourf(x0, x1, zz, cmap=custom_cmap)


# 数据分割
X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, randc

# 训练模型
# log_reg = LogisticRegression()
# log_reg.fit(X_train, y_train)
poly_log_reg = PolynomialLogisticRegression(degree=3)
poly_log_reg.fit(X_train, y_train)

# 结果可视化
# plot_decision_boundary(log_reg, axis=[0, 100, 0, 100])
plot_decision_boundary(poly_log_reg, axis=[0, 100, 0, 100])
plt.scatter(data_X[data_y == 0, 0], data_X[data_y == 0, 1], color='red')
plt.scatter(data_X[data_y == 1, 0], data_X[data_y == 1, 1], color='blue')
plt.xlabel('成绩1')
plt.ylabel('成绩2')
plt.title('两门课程成绩与是否录取的关系')
plt.show()
```



如果代码正确，你将得到类似下图的实验结果图。