实验三：逻辑回归

姓名：　　学号：

- 实验目的

理解和掌握逻辑回归模型基本原理和方法，学会使用逻辑回归模型对分类问题进行建模和预测，掌握分类问题上模型评估方法。

- 实验内容

编程实现逻辑回归模型，在给定数据集上，绘制损失函数曲线图。使用混淆矩阵、错误率、精度、查全率、查准率、F1 指标评估逻辑回归模型性能表现。

- 实验环境

python

numpy

matplotlib

- 实验代码

```
import numpy as np
import matplotlib.pyplot as plt


load_data_testing = np.loadtxt('experiment_03_testing_set.csv', delimiter=',')
load_data_training = np.loadtxt('experiment_03_training_set.csv', delimiter=',')



# sigmoid 函数
def h(x, w):
    return 1 / (1 + (np.exp(-np.dot(x, w))))
```

```
# 模型预测函数
def prediction(x, w):
    if h(x, w) > 0.5:
        return 1
    else:
        return 0


# 混淆矩阵计算函数
def mix_matrix(data_1, data_2):
    TP = np.sum((data_1 == 1.00) & (data_2 == 1.00))
    FN = np.sum((data_1 == 1.00) & (data_2 == 0.00))
    FP = np.sum((data_1 == 0.00) & (data_2 == 1.00))
    TN = np.sum((data_1 == 0.00) & (data_2 == 0.00))
    return TP, FN, FP, TN


# 损失率计算函数
def loss(y, x, w):
    loss_array = np.zeros(np.size(y, 0))
    for i in range(np.size(y, 0)):
```

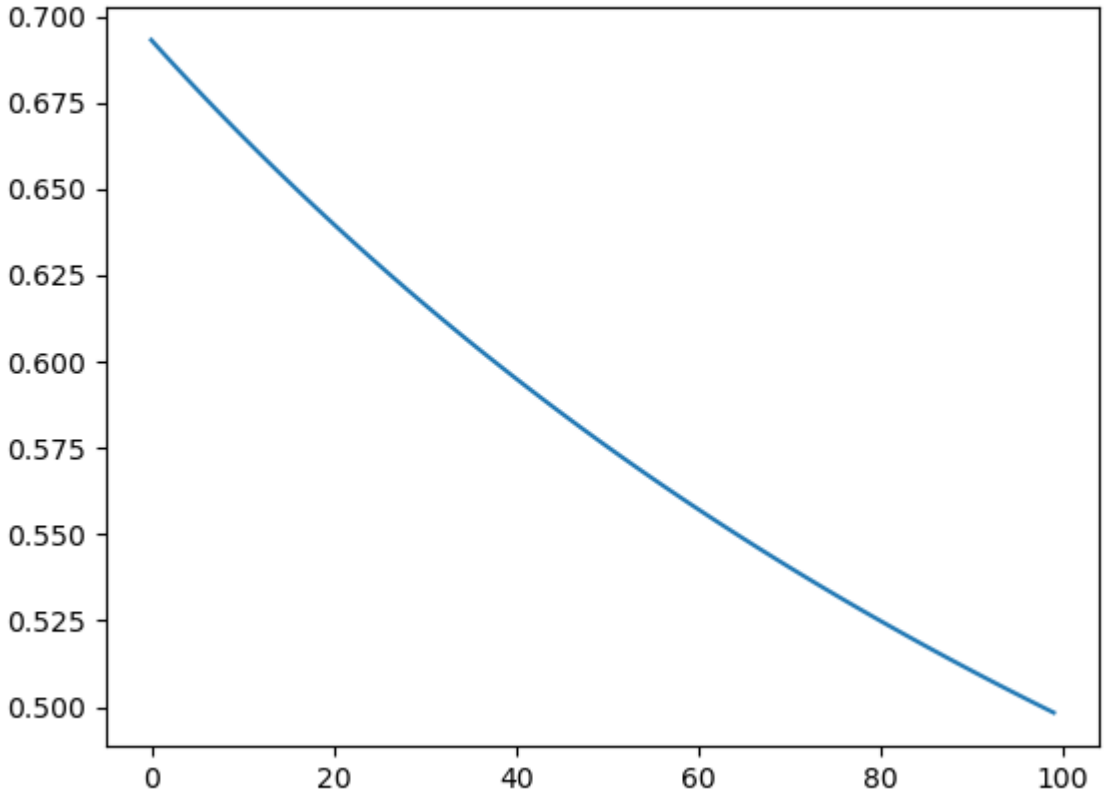
```
            loss_array[i] = y[i] * np.log(h(x[i, :], w)) + (1 - y[i]) * np.log(1 - h(x[i, :], w))
        return -np.sum(loss_array) / np.size(y, 0)


# 梯度计算函数
def loss_grad(x, y, w):
        loss_grad = np.zeros(w.shape)
        for i in range(np.size(y, 0)):
            h_array = (y[i] - h(x[i, :], w)) * x[i, :]
            loss_grad += h_array
        return -loss_grad / np.size(y, 0)


# 读取数据
x = np.ones(load_data_training.shape)
x[:, 0:load_data_training.shape[1] - 1] = load_data_training[:,
0:load_data_training.shape[1] - 1]
y = load_data_training[:, load_data_training.shape[1] - 1]
w = np.zeros(load_data_training.shape[1])

# 开始训练
k = 100
m = 0.1
```

```
loss_array = []
for i in range(k):
    gradient = loss_grad(x, y, w)
    loss_array.append(loss(y, x, w))
    w = w - gradient * m

array = np.zeros(load_data_testing.shape[0])
x_test = np.ones(load_data_testing.shape)
x_test[:, 0:load_data_testing.shape[1] - 1] = load_data_testing[:,
0:load_data_testing.shape[1] - 1]
y_test = load_data_testing[:, load_data_testing.shape[1] - 1]
for i in range(np.size(y_test, 0)):
    array[i] = prediction(x_test[i, :], w)

# 计算错误率和精度
num_equal_ones = np.sum((array == 1.0) & (y_test == 1.0))
num_equal_zeros = np.sum((array == 0.0) & (y_test == 0.0))
accuracy = (num_equal_ones + num_equal_zeros) / np.size(y_test, 0)
error_rate = 1 - accuracy
print('精度:', accuracy)
print('错误率:', error_rate)

# 计算混淆矩阵
```

```
TP, FN, FP, TN = mix_matrix(y_test, array)
print('TP:', TP, ' FN:', FN, ' FP', FP, ' TN', TN)

# 计算查准率
precision = TP / (TP + FP)
print('查准率:', precision)

# 计算查全率
recall = TP / (TP + FN)
print('查全率:', recall)

# 计算 F1
F1 = 2 * precision * recall / (precision + recall)
print('F1:', F1)

# 绘制损失函数图
plt.plot(loss_array)
plt.show()
```

- 结果分析

初始权值设为$\boldsymbol{w}$ = [0,0, …,0]，学习率设为 0. 1，迭代次数为 100。
损失曲线迭代图：



混淆矩阵：

真实情况	预测结果	
	正例	反例
正例	111	2
反例	14	42

评价指标：

指标	数值
错误率（error rate）	0.094675
精度（accuracy）	0.905325
查准率（precision）	0.888
查全率（recall）	0.982301
F1	0.932773