第 1 次实验报告

2022111663　刘博飞

2024 年 9 月 19 日

1　任务 1

对提供的两组数据 logistic_data1.csv , logisitic_data2.csv 通过逻辑回归进行分类，找出决策面。

1.1　数据 1

通过线性回归得到的运行结果如下图所示:

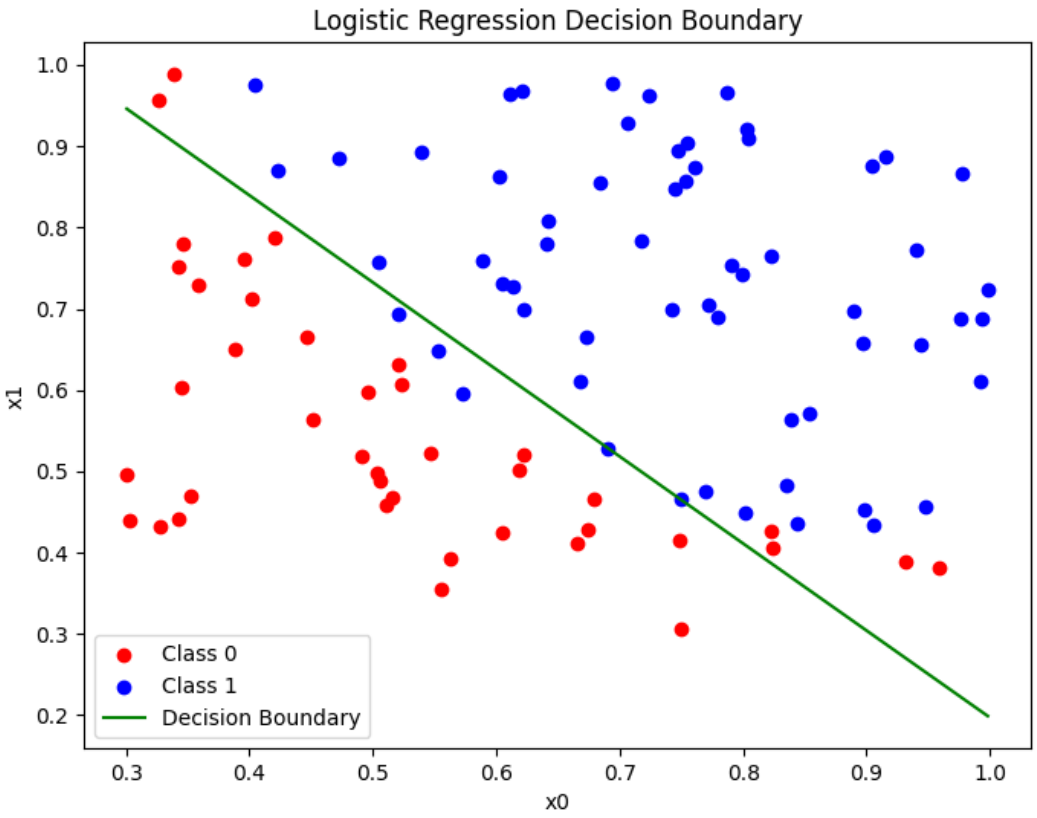


图 1: 数据 1 运行结果

1.2 数据 2

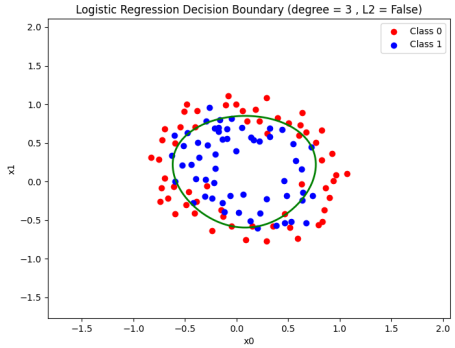
数据 2 需使用 x_0 , x_1 两个特征构造高维特征，并分别尝试无正则化和加入 L2 正则项来查找决策面，对比有无正则项时的分类难度和结果。

对于高维特征的选取，我采取了高次多项式的形式，表达式如下：

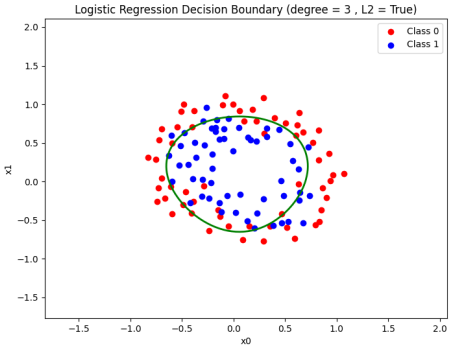
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ... + \theta_n x^n + ... \tag{1}$$

实验中，分别选择了 $n = 3, 4, 5, 6$ 四种情况，对每种情况分别进行无正则化和 L2 正则化，并记录了分类决策面和分类时间（衡量分类难度）。

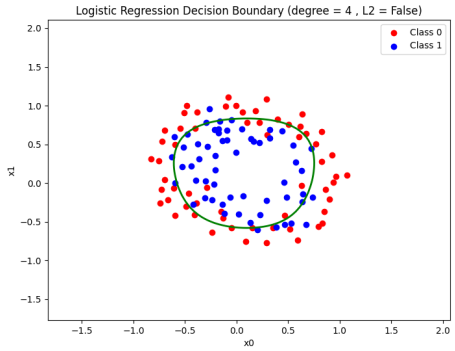
分类决策面如下图所示：



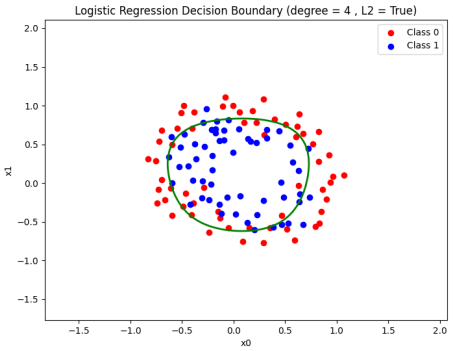
(a) n = 3, L2 = False



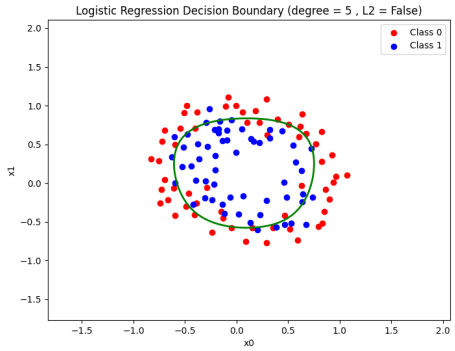
(b) n = 3, L2 = True



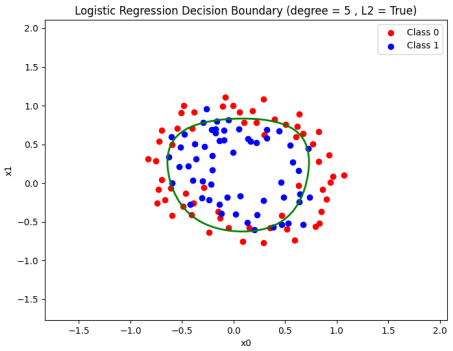
(c) n = 4, L2 = False



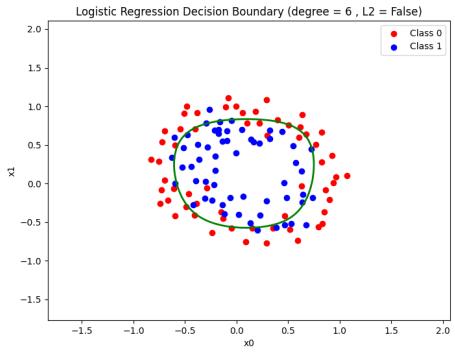
(d) n = 4, L2 = True



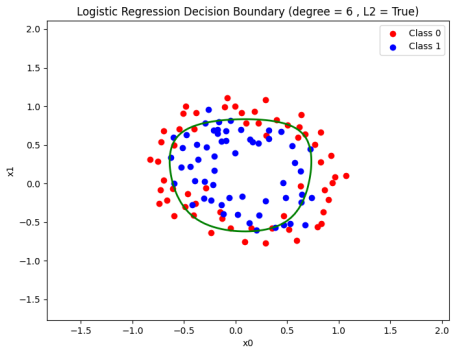
(e) n = 5, L2 = False



(f) n = 5, L2 = True



(g) n = 6, L2 = False



(h) n = 6, L2 = True

分类时间如下表所示：

表 1: 数据 2 分类运行时间

n	无正则化	L2 正则化
3	0.9981s	1.9410s
4	1.0721s	1.9867s
5	1.0111s	1.9468s
6	1.0978s	1.9374s

通过分析实验数据，我们发现，加入正则化后模型的运行时间增加了，这说明 L2 正则化会引入更多的计算量从而使分类更困难。同时，对比加入 L2 正则化后的图像，发现边界略微大于未加入正则化的图像，围住了更多的蓝色点，这是因为 L2 正则化通过加入一个参数平方和的惩罚项，限制了过大的参数，增加了模型的泛化能力，从而防止了过拟合。

2 任务 2

对提供的鸢尾花数据集通过逻辑回归进行分类。在训练集 iris_train.csv 上训练决策面，并在测试集 iris_test.csv 上执行分类任务，得到准确率如下表。

表 2: 三分类准确率

闭环准确率	开环准确率
98.0952%	95.5556%

A 任务 1.1 核心代码

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

    def compute_cost(X, y, theta):
        h = sigmoid(X @ theta)  # 计算预测值
        cost = (y.T @ np.log(h) + (1 - y).T @ np.log(1 - h)) / m
        # 对数损失函数
        return cost

    def gradient_descent(X, y, theta, alpha, num_iters):
        cost_history = []

        for _ in range(num_iters):
            h = sigmoid(X @ theta)
            gradient = X.T @ (h - y) / m
            theta -= alpha * gradient
            cost_history.append(compute_cost(X, y, theta))
        return theta, cost_history
```

B 任务 1.2 核心代码

```
    def polynomial_features(x0, x1, degree):
        # 计算 x0 和 x1 的不同幂次
        features = [np.ones_like(x0)]  # 添加偏置项列
        for d in range(1, degree + 1):
            features.append(x0 ** d)
            features.append(x1 ** d)
        # 通过按列拼接生成最终的特征矩阵
        return np.column_stack(features)

    def sigmoid(z):
        return 1 / (1 + np.exp(-z))

    # 定义损失函数和梯度计算
    def compute_cost(X, y, theta):
        h = sigmoid(X @ theta)  # 计算预测值
        # todo:决定是否正则化

        # cost = (y.T @ np.log(h) + (1 - y).T @ np.log(1 - h)) / m
        # 对数损失函数

        cost = (y.T @ np.log(h) + (1 - y).T @ np.log(1 - h)) / m

        + (Lambda / (2 * m)) * np.sum(theta[1:] ** 2)

        return cost

    def gradient_descent(X, y, theta, alpha, num_iters):
        cost_history = []

        for _ in range(num_iters):
            h = sigmoid(X @ theta)

            if L2:
                gradient = np.zeros_like(theta)
                gradient[0] = 1 / m * (X[:, 0].T @ (h - y))
                # 偏置项的梯度，不进行正则化
                gradient[1:] = 1 / m * (X[:, 1:].T @ (h - y)) + (Lambda /
                    ↪  m) * theta[1:]
                # 其他参数的梯度，加入正则化项
```

```
            else:
                gradient = X.T @ (h - y) / m
            theta -= alpha * gradient
            # cost_history.append(compute_cost(X, y, theta))
        return theta, cost_history

    # 绘制决策边界的函数
    def plot_decision_boundary(x0, x1, theta, degree):
        # 创建一个二维网格
        u = np.linspace(min(x0) - 1, max(x0) + 1, 100)
        v = np.linspace(min(x1) - 1, max(x1) + 1, 100)
        U, V = np.meshgrid(u, v)  # [[100]*100]
        # 使用多项式特征构造新的特征矩阵
        UV_poly = polynomial_features(U.ravel(), V.ravel(), degree)
        Z = UV_poly @ theta  # 10000*1
        Z = Z.reshape(U.shape)
        # 绘制决策边界
        plt.contour(U, V, Z, levels=[0], linewidths=2, colors='g')
```

C 任务 2 核心代码

```
    def softmax(z):
        e_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return e_z / np.sum(e_z, axis=1, keepdims=True)

    def compute_cost(Y, Y_hat):
        m = Y.shape[0]
        cost = -np.sum(Y * np.log(Y_hat + 1e-9))/m
        cost_history.append(cost)

    def one_hot_encoding(y):
        m = y.shape[0]
        Y = np.zeros((m, 3))
        for i in range(m):
            Y[i, int(y[i])] = 1
        return Y

    def train(X, y, alpha, iteration):
        features = 4
        species = 3
        Y = one_hot_encoding(y)
        W = np.random.randn(features, species) * 0.01
        b = np.zeros((1, species))
        m = X.shape[0]
        for i in range(iteration):
            Z = X@W + b
            Y_hat = softmax(Z)
            compute_cost(Y, Y_hat)
            dZ = Y_hat - Y
            dW = X.T@dZ / m
            db = np.sum(dZ, axis=0, keepdims=True) / m
            W -= alpha * dW
            b -= alpha * db
        return W, b

    def test(X, y, W, b):
        Z = X@W + b
        Y_hat = softmax(Z)
        y_pred = np.argmax(Y_hat, axis=1)
        sum = len(y_pred)
```

```
        correct = 0
        for i in range(sum):
            if y_pred[i] == int(y[i]):
                correct += 1
        accuracy = correct/sum*100
        return accuracy
```