《机器学习》编程作业 1

题目 3.3

一、题目理解

采用留出法实现对率回归，给出西瓜数据集 3.0a 的性能评估结果。留出法是一种简单的交叉验证方法，将数据集划分为训练集和测试集，使用训练集训练模型，然后在测试集上评估模型的性能。使用 math 数学库手动编写对率回归算法代码，对西瓜数据集 3.0a 进行预测准确率等模型评估。同时使用 sklearn 机器学习库进行代码实现，比较两种实现方式的性能评估和效果。

二、逻辑回归算法原理阐述

2.1 逻辑函数（Sigmoid Function）

逻辑回归的模型是基于线性回归的，但不同于线性回归直接输出预测值，逻辑回归使用 Sigmoid 函数将线性回归的输出值转化为一个概率值。Sigmoid 函数表达式如下：

$$y = \frac{1}{1+e^{-z}}.$$

其中，z=θTx 是线性回归的预测值，θ 是参数向量，x 是输入特征向量。

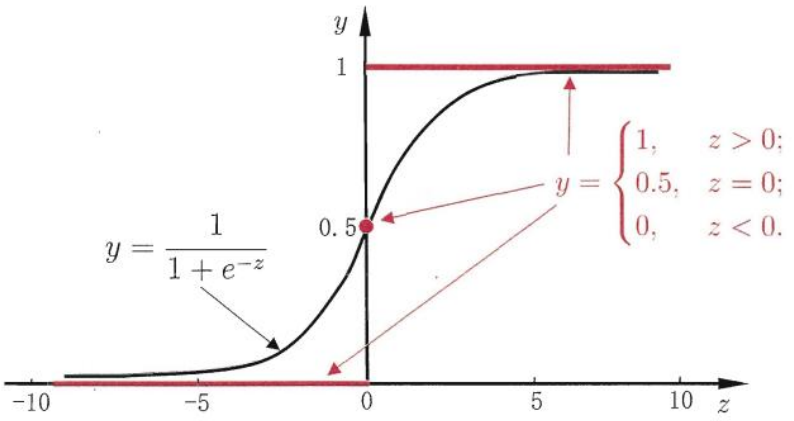


图 1 单位越阶与 sigmoid 函数

通过 Sigmoid 函数，逻辑回归的输出值被压缩在 0 到 1 之间，可以解释为属于正类的概率。当输出值大于 0.5 时，通常认为该样本属于正类；反之，则认为属于负类。

2.2 损失函数（Loss Function）

由于采用梯度下降法实现逻辑回归，故需要用到损失函数。
逻辑回归的损失函数通常使用交叉熵损失（Cross Entropy Loss），其表达式如下：

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

其中，m 是样本数量，y(i) 是第 i 个样本的真实标签（0 或 1），hθ(x(i)) 是逻辑回归模型对第 i 个样本的预测概率。

2.3 参数优化（Parameter Optimization）

为了找到使损失函数最小的参数 θ，我们可以使用梯度下降法（Gradient Descent）来迭代更新参数。梯度下降法的迭代公式如下：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

其中，α 是学习率，控制参数更新的步长；$\frac{\partial J(\theta)}{\partial \theta_j}$是损失函数对参数$\theta_j$的偏导数。

2.4 正则化

正则化是一种用于防止模型过拟合的技术。过拟合是指模型在训练数据上表现很好，但在测试数据上表现较差的现象。正则化通过在损失函数中添加一个与模型参数的平方和（L2 正则化）或绝对值和（L1 正则化）成正比的项，来限制模型参数的大小，从而减少模型的复杂度，提高模型的泛化能力。

L2 正则化在损失函数中添加一个与模型参数的平方和成正比的项：

$$\text{Loss} = \text{Cross-Entropy Loss} + \lambda \sum_{i=1}^{n} \theta_i^2$$

其中，λλ 是正则化参数，控制正则化的强度。λλ 越大，正则化越强，模型参数越小。

在逻辑回归中，正则化的步骤如下：

（1）定义损失函数：在交叉熵损失函数的基础上，添加正则化项。

（2）计算梯度：计算损失函数对模型参数的梯度时，考虑正则化项的影响。

（3）更新参数：使用梯度下降法更新模型参数时，同时考虑正则化项。

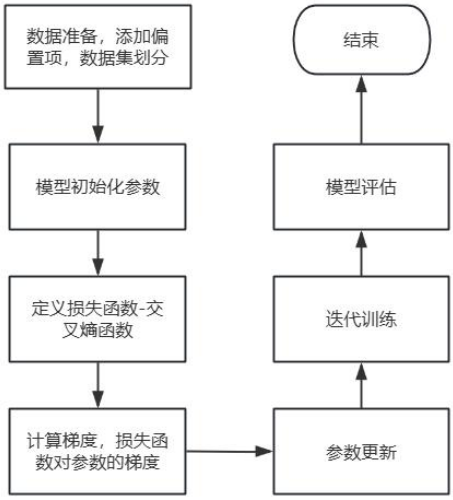
三、基于梯度下降的逻辑回归算法设计思路

首先设计两个版本的代码，一个是纯数学公式推导手动实现的逻辑回归，采用留出法将数据集划分为训练集以及测试集，比例可以自己设置。另外一个是直接调用 python 中的机器学习库 sklearn 的 LogisticRegression()，将处理后的数据交给模型。

首先读入数据，对其进行处理，将 x（即密度、糖度）插入一个全为 1 的列，作为偏置项。将数据划分为训练集以及测试集，比例为 0.25。两个版本的代码共享一个数据集。

对于手写版本，进行模型初始化，初始化模型参数（权重和偏置项）。然后，定义损失函数，通常使用交叉熵损失函数。并计算梯度，计算损失函数对模型参数的梯度。进行参数更新，使用梯度下降法更新模型参数。然后进行迭代训练，重复步骤 4 和步骤 5，直到达到预定的迭代次数或损失函数收敛。最后，进行模型评估，使用测试集评估模型的性能，计算

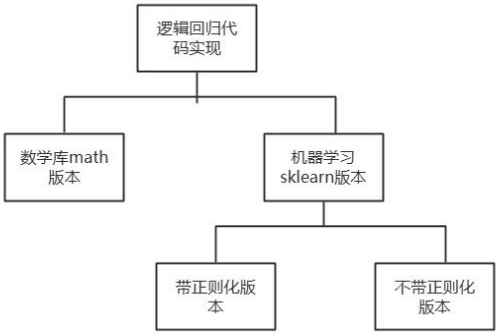
准确率、混淆矩阵等指标。



四、代码结构及核心部分介绍

4.1 整体代码结构

为了防止逻辑回归学习过拟合，加入了正则化防止过度学习，math 库实现的版本通过加入 lbd 正则化因子，使得在参数更新时能够考虑到正则化。同样的，机器学习 sklearn 版本也采用了正则化，结果表明不带正则化的版本效果不如带正则化的版本。



4.2 核心部分介绍

4.2.1 手写代码部分

1. Sigmoid 函数

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

（1）功能：sigmoid 函数将输入值 z 转换为概率值（0 到 1 之间的值）。

（2）对应公式：sigmoid(z) = 1 / (1 + exp(-z))。
在逻辑回归中，sigmoid 函数用于将线性组合的结果转换为概率值，从而进行分类。

2. 逻辑回归函数

```
def logit_regression(theta, x, y, iteration=1000, learning_rate=0.01, lbd=0.01):
    # 获取样本数量
    m = y.shape[0]

    # 迭代更新参数
    for i in range(iteration):
        # 计算线性组合
        linear_model = np.dot(x, theta)

        # 计算预测值
        y_predicted = sigmoid(linear_model)

        # 计算梯度
        gradient = np.dot(x.transpose(), (y_predicted - y)) / m

        # 更新参数
        theta = theta - learning_rate * (gradient + lbd * theta)

        # 计算损失函数
        cost = -1 / m * (np.dot(y.transpose(), np.log(y_predicted)) + np.dot((1 - y).transp
ose(),
                                                                              n
p.log(1 - y_predicted))) + lbd / (2 * m) * np.dot(theta.transpose(), theta)
    return theta
```

（1）功能：logit_regression 函数使用梯度下降法训练逻辑回归模型。
（2）参数介绍：
theta：模型参数，形状为 (n+1, 1)，其中 n 是特征数量（包括偏置项）。
x：特征矩阵，形状为 (m, n+1)，其中 m 是样本数量。
y：标签向量，形状为 (m, 1)。
iteration：迭代次数，默认值为 1000。
learning_rate：学习率，默认值为 0.01。
lbd：正则化参数，默认值为 0.01。
（3）详细解释：
np.dot(x, theta)，计算特征矩阵 x 和参数向量 theta 的线性组合。
y_predicted = sigmoid(linear_model)，计算预测值：将线性组合的结果通过 sigmoid 函数转换为概率值。
gradient = np.dot(x.transpose(), (y_predicted - y)) / m，计算梯度：计算损失函数对参数 theta 的梯度。

theta = theta - learning_rate * (gradient + lbd * theta)，更新参数：使用梯度下降法更新参数 theta，同时考虑正则化项。

cost = -1 / m * (np.dot(y.transpose(), np.log(y_predicted)) + np.dot((1 - y).transpose(), np.log(1 - y_predicted))) + lbd / (2 * m) * np.dot(theta.transpose(), theta)，计算损失函数，包括交叉熵损失和 L2 正则化项。

最后返回更新后的参数：return theta。

3. 预测函数

```
def predict(theta, x):
    pre = np.zeros([x.shape[0], 1])
    for idx, valu in enumerate(np.dot(x, theta)):
        if sigmoid(valu) >= 0.5:
            pre[idx] = 1
        else:
            pre[idx] = 0
    return pre
```

（1）功能：predict 函数根据模型参数 theta 和特征矩阵 x 进行预测。

（2）参数介绍：

theta：模型参数，形状为 (n+1, 1)。

x：特征矩阵，形状为 (m, n+1)。

（3）详细解释：

pre = np.zeros([x.shape[0], 1])，初始化预测结果：创建一个全为 0 的数组，用于存储预测结果。

np.dot(x, theta)，计算线性组合：计算特征矩阵 x 和参数向量 theta 的线性组合。

预测类别：对于每个样本，计算 sigmoid(valu)，如果结果大于等于 0.5，则预测为 1，否则预测为 0。

pre[idx] = 1 或 pre[idx] = 0，将预测结果存储在 pre 数组中。

4. 模型训练和预测

```
# 随机初始化 theta, 为参数矩阵，包括 w1, w2 和 b
theta = np.random.rand(3, 1)
theta = logit_regression(theta, xtrain, ytrain, learning_rate=1)
pre = predict(theta, xtest)
```

（1）theta = np.random.rand(3, 1)，随机初始化模型参数 theta，形状为 (3, 1)，包括 w1、w2 和 b。

（2 模型训练：theta = logit_regression(theta, xtrain, ytrain, learning_rate=1)，使用训练集 x train 和标签 ytrain 训练模型，更新参数 theta。

（3）模型预测：pre = predict(theta, xtest)，使用测试集 xtest 进行预测，得到预测结果 pr e。

4.3 sklearn 核心代码介绍

1.创建带有正则化的逻辑回归模型

```
# 创建模型，C 值越小，代表正则化越强
```

```
model = sl.LogisticRegression(solver='sag', C=200)
```

参数介绍：

solver='sag'：指定优化算法为随机平均梯度下降（Stochastic Average Gradient Descent）。

C=200：正则化强度的倒数，C 值越小，正则化越强。

2. 模型评估

评估模型的性能，计算并输出精度、查准率、查全率和 F1 得分。

```
acc = model.score(x_train, y_train)print('score', acc)
# 分类任务性能度量：精度（accuracy）、查准率（precision）、查全率（recall，召回率）和 F1print('精度：', sm.accuracy_score(y_test, pred_y))print('查准率：', sm.precision_score(y_test, pred_y, average='macro'))print('召回率：', sm.recall_score(y_test, pred_y, average='macro'))print('f1 得分：', sm.f1_score(y_test, pred_y, average='macro'))print('report', sm.classification_report(y_test, pred_y))
```

模型得分：acc = model.score(x_train, y_train)，计算模型在训练集上的得分。

精度：sm.accuracy_score(y_test, pred_y)，计算模型在测试集上的精度。

查准率：sm.precision_score(y_test, pred_y, average='macro')，计算模型在测试集上的查准率。

查全率：sm.recall_score(y_test, pred_y, average='macro')，计算模型在测试集上的查全率。

F1 得分：sm.f1_score(y_test, pred_y, average='macro')，计算模型在测试集上的 F1 得分。

分类报告：sm.classification_report(y_test, pred_y)，输出详细的分类报告。

3. 结果可视化

绘制训练集和测试集的点，并绘制逻辑回归的决策边界。

```
# 绘制训练集的所有点
plt.figure(figsize=(10, 6))
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, cmap='brg', s=80, label='Training Data')
# 绘制测试集的预测结果
plt.scatter(x_test[:, 0], x_test[:, 1], c=pred_y, cmap='brg', s=80, marker='x', label='Test Predictions')
# 绘制逻辑回归的决策边界
x_min, x_max = x[:, 0].min() - 0.1, x[:, 0].max() + 0.1
y_min, y_max = x[:, 1].min() - 0.1, x[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3, cmap='brg')

plt.title('Watermelon Classification')
plt.xlabel('Density', fontsize=14)
plt.ylabel('Sugar Content', fontsize=14)
plt.tick_params(labelsize=10)
plt.legend()
plt.show()
```

绘制决策边界：

（1）创建网格：xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))，创建一个网格，用于在特征空间中绘制决策边界。

（2）计算网格上的预测值：Z = model.predict(np.c_[xx.ravel(), yy.ravel()])，使用模型对网格上的每个点进行预测。

（3）绘制决策边界：plt.contourf(xx, yy, Z, alpha=0.3, cmap='brg')，绘制决策边界。

五、实验流程、测试结果及分析

5.1 流程与测试结果

分别使用 math 库手写代码版本、sklearn 未正则化版、sklearn 正则化版本训练数据，得到三种测试结果，给出三个模型的模型评估如下：

（1）math 库手动实现

在迭代次数为 1000 次，学习率为 0.01，正则化因子为 0.01 时，模型的测试结果如下。

```
the accuracy is 0.8
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.67      0.80         3

    accuracy                           0.80         5
   macro avg       0.83      0.83      0.80         5
weighted avg       0.87      0.80      0.80         5
```

可以看到，模型的准确率为 80%，表示模型在测试集上的预测正确率为 80%。类别 0 的精确率为 67%，召回率为 100%。类别 1 的精确率为 100%，召回率为 67%类别 0 和类别 1 的 F1 得分均为 80%。宏平均和加权平均的精确率、召回率和 F1 得分均在 80% 左右。

总体来看，模型的性能较好，但在类别 0 和类别 1 的召回率上存在一定的差异。可以通过调整模型参数或使用更复杂的模型来进一步提高模型的性能。

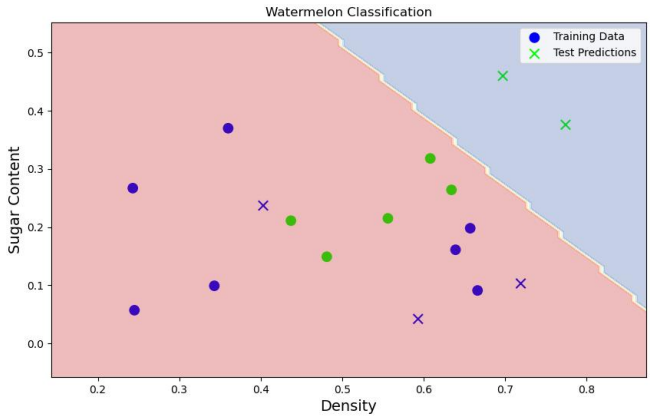


图 2 math 库实现，lbd=0.01 的决策边界

更改 math 库实现方法中的正则化因子，将 ld=0.01 改为 0.001，得到决策边界如下：

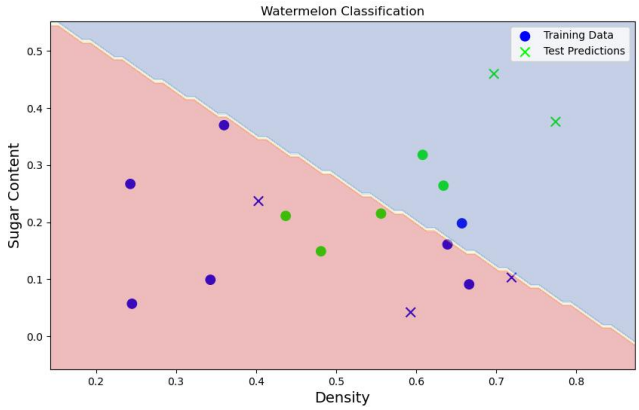


图 3 math 库实现，lbd=0.001 的决策边界

（2）sklearn 未正则化

```
score 0.5833333333333334
精度： 0.4
查准率： 0.2
召回率： 0.5
f1得分： 0.2857142857142857
report              precision    recall  f1-score   support

           0       0.40      1.00      0.57         2
           1       0.00      0.00      0.00         3

    accuracy                           0.40         5
   macro avg       0.20      0.50      0.29         5
weighted avg       0.16      0.40      0.23         5
```

模型的精度为 40%，表示模型在测试集上的预测正确率为 40%。类别 0 的精确率为 40%，召回率为 100%。类别 1 的精确率为 0%，召回率为 0%。类别 0 的 F1 得分为 57%，类别 1 的 F1 得分为 0%。宏平均和加权平均的精确率、召回率和 F1 得分均较低。

总体来看，模型的性能较差，特别是在类别 1 的预测上表现不佳。

（3）sklearn 正则化

```
score 0.5833333333333334
精度： 0.8
查准率： 0.8333333333333333
召回率： 0.8333333333333333
f1得分： 0.8
C:\Users\1234\.conda\envs\pytorch\Lib\site-packages\sklearn\metri
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(r
C:\Users\1234\.conda\envs\pytorch\Lib\site-packages\sklearn\metri
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(r
C:\Users\1234\.conda\envs\pytorch\Lib\site-packages\sklearn\metri
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(r
C:\Users\1234\.conda\envs\pytorch\Lib\site-packages\sklearn\utils
  y = column_or_1d(y, warn=True)
C:\Users\1234\.conda\envs\pytorch\Lib\site-packages\sklearn\linea
  warnings.warn(
report              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.67      0.80         3

    accuracy                           0.80         5
   macro avg       0.83      0.83      0.80         5
weighted avg       0.87      0.80      0.80         5
```

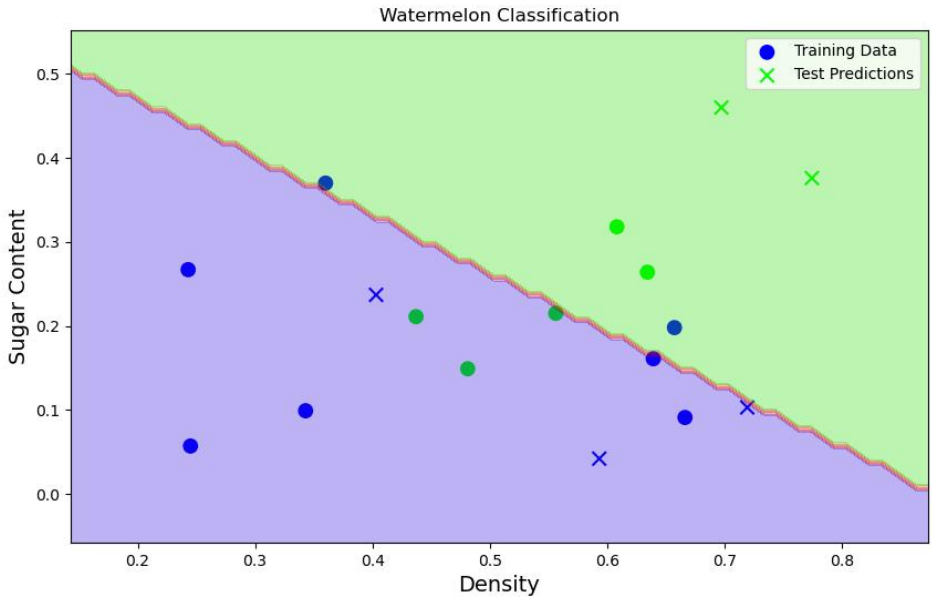


图 4 sklearn 正则化版本模型的决策边界图

模型的精度为 80%，表示模型在测试集上的预测正确率为 80%。类别 0 的精确率为 67%，召回率为 100%。类别 1 的精确率为 100%，召回率为 67%。类别 0 和类别 1 的 F1 得分均为 80%。宏平均和加权平均的精确率、召回率和 F1 得分均在 80% 左右。

总体来看，模型的性能较好，但在类别 0 和类别 1 的召回率上存在一定的差异。可以通过调整模型参数或使用更复杂的模型来进一步提高模型的性能。

5.2 结果分析

通过比较 math 库的两个 lbd 取值以及 sklearn 库分别运用与不运用正则化的版本可以发现，改数据集下，运用逻辑回归进行模型训练时，正则化是十分重要的一个操作，这样可以尽可能地防止模型过拟合，从而得到更好的结果。

六、问题与收获

6.1 遇到的问题

1. 数据预处理：

问题：在数据预处理阶段，一开始时，我并没有添加偏置项，导致模型的结果很不好。

解决方法：在特征矩阵的右边添加一列全为 1 的列，作为偏置项。

2 决策边界绘制：

问题：在绘制决策边界时，需要确保网格点的预测值计算正确，刚开始我的图一直有问题，并且决策边界的绘制的很有问题。

解决方法：使用 `meshgrid` 创建网格，计算网格点的预测值，并使用 `contourf` 函数绘制决策边界。

6.2 收获

本次实验通过手动实现逻辑回归模型和使用 `sklearn` 库中的逻辑回归模型，我深入理解了逻辑回归的原理和实现细节，掌握了数据预处理、模型训练、模型评估和结果可视化的方法。通过对比不同模型的性能，我认识到正则化在防止模型过拟合中的重要作用。总体来说，本次实验让我在机器学习领域有了更深入的理解和实践经验。

七、参考资料

[1]《西瓜书》

[2]csdn 博客：逻辑回归（Logistic Regression）原理及其应用 逻辑回归应用-CSDN 博客

[3]csdn 博客：线性回归 西瓜数据集 Python--sklearn 线性回归数据集-CSDN 博客

[4]csdn 博客：正则化逻辑回归【Python 详细注释实现】 对逻辑斯诶进行正则化怎么弄-CSDN 博客

题目 3.4

一、题目理解

结合 3.3 实现的对率回归算法，采用 10 折交叉验证法在两种 UCI 数据集上（除 iris 和 wine 之外）估计错误率。

即针对 3.3，不使用留出法进行评估，而改用 10 折交叉验证法。这里我仅对手写代码版本进行交叉验证。

选择的两种 UCI 数据集分别是：

（1）Breast Cancer Coimbra 数据集: 用于乳腺癌预测的数据集，包含 9 个特征和 1 个标签。标签为 1 表示健康，2 表示患病。

（2）Pima Indians Diabetes Database 数据集: 用于糖尿病预测的数据集，包含 8 个特征和 1 个标签。标签为 1 表示患有糖尿病，0 表示未患有糖尿病。

二、十折交叉验证算法原理阐述

十折交叉验证是一种用于评估模型性能的技术。将训练集分割成 10 个子样本，一个单独的子样本被保留作为验证模型的数据，其他 9 个样本用来训练。交叉验证重复 10 次，每个子样本验证一次，平均 10 次的结果或者使用其它结合方式，最终得到一个单一估测。这个方法的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次，10 次交叉验证是最常用的。

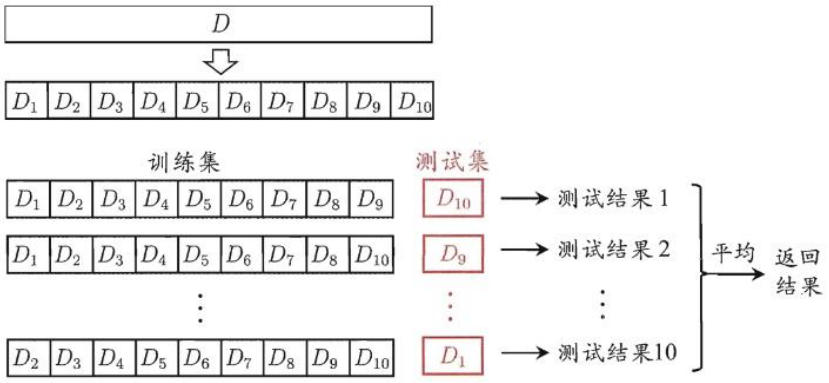


图 5 十折交叉验证算法示意图

其基本步骤如下：

（1）将数据集随机分为 10 个子集（折）。

（2）每次选择 1 个子集作为测试集，其余 9 个子集作为训练集。

（3）重复 10 次，每个子集都作为测试集一次。

（4）记录每次的错误率，最后计算 10 次的平均错误率，以获得模型的整体性能评估。

这种方法有效地利用了数据集，提高了模型的泛化能力，并减少了由于数据划分引入的偏差。

三、设计思路

本题目的目标是使用 3.3 中实现的逻辑回归模型对乳腺癌和糖尿病数据进行分类。整体设计思路为：

首先进行数据加载与预处理：从 CSV 文件中加载数据，处理特征和标签，并进行标准化和偏置处理。第一个数据集中的 1 与 2 要映射为 0 与 1。

使用 3.3 中手动实现的逻辑回归模型：构建逻辑回归算法，包括正向传播（计算预测值）和反向传播（更新模型参数）。

使用十折交叉验证评估模型性能：使用 KFold 将数据分为 10 个折。在每个折中，训练模型并对测试集进行预测，计算错误率并输出分类报告。最后，计算并打印平均错误率。在训练和测试阶段重复训练过程，评估模型的准确性和泛化能力。

四、代码结构及核心部分介绍

4.1 整体代码结构

整体代码结构主要分为以下几部分：

1. 数据加载与预处理

preprocess_data(data)函数，用于加载和处理数据集，将患病标记 2 转换为 0。标准化特征，添加偏置项，并处理标签。返回 dataArr 与 labelArr，分别是标准化后的特征数组（numpy.ndarray）和处理后的标签数组（numpy.ndarray）

2. 模型实现

（1）sigmoid(z)函数，计算输入值的 sigmoid 函数，返回值范围在 0 到 1 之间。

（2）logit_regression(theta, x, y, iteration=1000, learning_rate=0.01, lbd=0.01)函数，训练逻辑回归模型，更新参数 theta。

（3）predict(theta, x)函数，使用训练好的模型进行预测，返回预测结果。

3. 交叉验证

cross_validation(data)函数：实现十折交叉验证，训练模型并评估其性能。输入参数为 data，即输入的原始数据集（pandas.DataFrame），在每次迭代中打印分类报告和平均错误率。函数的主要步骤为：调用 preprocess_data 函数处理数据。使用 KFold 将数据划分为 10 个折。在每个折中训练模型，调用 logit_regression 更新参数，使用 predict 进行预测。计算错误率并打印分类报告。

4.2 核心部分介绍

1. preprocess_data(data) 函数

该函数的主要作用是对输入的数据集进行预处理，包括特征提取、标准化和标签转换。具体步骤如下：

（1）数据转换：

```
dataSet = data.values  # 将数据框 (DataFrame) 转换为 NumPy 数组
dataArr = dataSet[:, :-1]  # 提取特征数组 dataArr，去掉最后一列（标签）
```

labelArr = dataSet[:, -1].reshape(-1, 1) #提取标签数组 labelArr，取最后一列并调整形状为列向量

（2）进行标准化：

```
dataMax = dataArr.max(axis=0)
dataMin = dataArr.min(axis=0)
dataArr = (dataArr - dataMax) / (dataMax - dataMin)
```

首先计算每个特征的最大值和最小值。对特征进行标准化处理，使得每个特征的值范围缩放到 [0, 1] 之间。这样可以加快后续模型的收敛速度。

（3）增加偏置项：

```
dataArr = np.hstack((dataArr, np.ones([dataArr.shape[0], 1])))
```

在特征数组的末尾添加一列全为 1 的偏置项，用于逻辑回归中的截距。

（4）标签转换：

```
labelArr[labelArr == 2] = 0
return dataArr, labelArr
```

将原始标签中表示健康的 1 和表示患病的 2 转换为 0，这样符合逻辑回归模型的要求。最后返回处理后的特征数组和标签数组，供后续模型训练使用。

2. cross_validation(data) 函数

实现十折交叉验证，评估模型在不同数据子集上的性能。具体步骤如下：

（1）数据预处理

```
x, y = preprocess_data(data)
```

调用 preprocess_data 函数对输入数据进行处理，获取标准化后的特征数组 x 和标签数组 y。

（2）设置交叉验证

```
kf = KFold(n_splits=10, shuffle=True, random_state=33)
```

使用 KFold 类将数据划分为 10 个折。shuffle=True 表示在划分之前打乱数据顺序，random_state=33 确保每次运行结果一致。

（3）迭代训练与测试

```
for train_index, test_index in kf.split(x):
x_train, x_test = x[train_index], x[test_index]
y_train, y_test = y[train_index], y[test_index]
```

在每个折中，通过 kf.split(x) 获得训练集和测试集的索引。根据索引提取训练集 x_train 和测试集 x_test，以及对应的标签 y_train 和 y_test。

（4）模型训练

```
theta = np.random.rand(x_train.shape[1], 1)
theta = logit_regression(theta, x_train, y_train)
```

随机初始化权重 theta，并调用 logit_regression 函数训练模型，更新参数。

（5）模型预测

```
predictions = predict(theta, x_test)
```

使用训练好的模型对测试集进行预测。

（6）评估模型性能

```
error_rate = 1 - accuracy_score(y_test, predictions)
error_rates.append(error_rate)
```

计算模型在测试集上的错误率，并将其存入 error_rates 列表。

（7）打印分类报告

print(classification_report(y_test, predictions, target_names=['健康', '患病'], zero_division=0))

打印分类报告，包括精确率、召回率和 F1 分数。

（8）计算和打印平均错误率

avg_error_rate = np.mean(error_rates)print(f"平均错误率: {avg_error_rate:.4f}")

计算所有折的平均错误率并打印结果。

五、实验流程、测试结果及分析

5.1 流程与测试结果

设置不同的迭代次数、学习率和正则化因子，得到不同的平均错误率。

（1）iteration=1000, learning_rate=0.01, lbd=0.01

Breast Cancer Coimbra 数据集:

每一折的评估结果如下：

	precision	recall	f1-score	support
健康	0.86	0.60	0.71	10
患病	0.20	0.50	0.29	2
accuracy			0.58	12
macro avg	0.53	0.55	0.50	12
weighted avg	0.75	0.58	0.64	12

	precision	recall	f1-score	support
健康	0.40	0.80	0.53	5
患病	0.50	0.14	0.22	7
accuracy			0.42	12
macro avg	0.45	0.47	0.38	12
weighted avg	0.46	0.42	0.35	12

	precision	recall	f1-score	support
健康	0.67	0.67	0.67	9
患病	0.00	0.00	0.00	3
accuracy			0.50	12
macro avg	0.33	0.33	0.33	12
weighted avg	0.50	0.50	0.50	12

	precision	recall	f1-score	support
健康	0.56	0.71	0.62	7
患病	0.33	0.20	0.25	5
accuracy			0.50	12
macro avg	0.44	0.46	0.44	12
weighted avg	0.46	0.50	0.47	12

	precision	recall	f1-score	support
健康	0.50	1.00	0.67	6
患病	0.00	0.00	0.00	6
accuracy			0.50	12
macro avg	0.25	0.50	0.33	12
weighted avg	0.25	0.50	0.33	12

	precision	recall	f1-score	support
健康	0.55	1.00	0.71	6
患病	0.00	0.00	0.00	5
accuracy			0.55	11
macro avg	0.27	0.50	0.35	11
weighted avg	0.30	0.55	0.39	11

	precision	recall	f1-score	support
健康	0.62	1.00	0.77	5
患病	1.00	0.50	0.67	6
accuracy			0.73	11
macro avg	0.81	0.75	0.72	11
weighted avg	0.83	0.73	0.71	11

	precision	recall	f1-score	support
健康	0.40	0.29	0.33	7
患病	0.17	0.25	0.20	4
accuracy			0.27	11
macro avg	0.28	0.27	0.27	11

weighted avg	0.32	0.27	0.28	11

	precision	recall	f1-score	support
健康	0.00	0.00	0.00	1.0
患病	0.00	0.00	0.00	10.0
accuracy			0.00	11.0
macro avg	0.00	0.00	0.00	11.0
weighted avg	0.00	0.00	0.00	11.0

	precision	recall	f1-score	support
健康	0.73	1.00	0.84	8
患病	0.00	0.00	0.00	3
accuracy			0.73	11
macro avg	0.36	0.50	0.42	11
weighted avg	0.53	0.73	0.61	11

平均错误率: 0.5227

Pima Indians Diabetes Database 数据集：

	precision	recall	f1-score	support
健康	0.57	1.00	0.73	44
患病	0.00	0.00	0.00	33
accuracy			0.57	77
macro avg	0.29	0.50	0.36	77
weighted avg	0.33	0.57	0.42	77

	precision	recall	f1-score	support
健康	0.70	1.00	0.82	54
患病	0.00	0.00	0.00	23
accuracy			0.70	77
macro avg	0.35	0.50	0.41	77
weighted avg	0.49	0.70	0.58	77

	precision	recall	f1-score	support
健康	0.57	1.00	0.73	44
患病	0.00	0.00	0.00	33

accuracy			0.57	77
macro avg	0.29	0.50	0.36	77
weighted avg	0.33	0.57	0.42	77

	precision	recall	f1-score	support
健康	0.71	1.00	0.83	55
患病	0.00	0.00	0.00	22
accuracy			0.71	77
macro avg	0.36	0.50	0.42	77
weighted avg	0.51	0.71	0.60	77

	precision	recall	f1-score	support
健康	0.77	0.98	0.87	59
患病	0.50	0.06	0.10	18
accuracy			0.77	77
macro avg	0.64	0.52	0.48	77
weighted avg	0.71	0.77	0.69	77

	precision	recall	f1-score	support
健康	0.68	1.00	0.81	52
患病	0.00	0.00	0.00	25
accuracy			0.68	77
macro avg	0.34	0.50	0.40	77
weighted avg	0.46	0.68	0.54	77

	precision	recall	f1-score	support
健康	0.64	1.00	0.78	49
患病	0.00	0.00	0.00	28
accuracy			0.64	77
macro avg	0.32	0.50	0.39	77
weighted avg	0.40	0.64	0.49	77

	precision	recall	f1-score	support
健康	0.50	1.00	0.67	38

患病	0.00	0.00	0.00	38
accuracy			0.50	76
macro avg	0.25	0.50	0.33	76
weighted avg	0.25	0.50	0.33	76

	precision	recall	f1-score	support
健康	0.70	1.00	0.82	53
患病	0.00	0.00	0.00	23
accuracy			0.70	76
macro avg	0.35	0.50	0.41	76
weighted avg	0.49	0.70	0.57	76

	precision	recall	f1-score	support
健康	0.68	1.00	0.81	52
患病	0.00	0.00	0.00	24
accuracy			0.68	76
macro avg	0.34	0.50	0.41	76
weighted avg	0.47	0.68	0.56	76

平均错误率: 0.3482

总结：

iteration=1000, learning_rate=0.01, lbd=0.01

Breast Cancer Coimbra 数据集:平均错误率: 0.5227

Pima Indians Diabetes Database 数据集：0.3482

（2）**iteration=100, learning_rate=0.01, lbd=0.01**

Breast Cancer Coimbra 数据集:平均错误率: 0.4545

Pima Indians Diabetes Database 数据集：0.3482

（3）**iteration=1000, learning_rate=0.001, lbd=0.01**

Breast Cancer Coimbra 数据集:平均错误率: 0.4462

Pima Indians Diabetes Database 数据集：0.3482

（4）**iteration=1000, learning_rate=0.001, lbd=0.001**

Breast Cancer Coimbra 数据集:平均错误率: 0.5326

Pima Indians Diabetes Database 数据集：0.3456

5.2 结果分析

通过对不同超参数组合的测试，我们可以看到这些超参数对模型性能的影响。在初始设置中，迭代次数为 1000 次，学习率为 0.01，正则化强度为 0.01，平均错误率分别为 0.5227（Breast Cancer Coimbra 数据集）和 0.3482（Pima Indians Diabetes Database 数据集）。当减少迭代次数到 100 次时，平均错误率降低到 0.4545（Breast Cancer Coimbra 数据集），而 Pima Indians Diabetes Database 数据集的错误率保持不变。

降低学习率到 0.001 时，平均错误率进一步降低到 0.4462（Breast Cancer Coimbra 数据集），而 Pima Indians Diabetes Database 数据集的错误率保持不变。这表明较低的学习率可能有助于模型更稳定地收敛，从而提高模型的性能。然而，当进一步降低正则化强度到 0.001 时，平均错误率略微增加到 0.5326（Breast Cancer Coimbra 数据集），而 Pima Indians Diabetes Database 数据集的错误率略有下降到 0.3456。

综合来看，最佳的超参数组合为迭代次数 1000 次，学习率 0.001，正则化强度 0.01，其在 Breast Cancer Coimbra 数据集上的平均错误率为 0.4462，在 Pima Indians Diabetes Database 数据集上的平均错误率为 0.3482。较少的迭代次数可能已经足够使模型达到较好的性能，但增加迭代次数可以确保模型充分收敛。较低的学习率有助于模型更稳定地收敛，而较高的正则化强度有助于防止过拟合，从而提高模型的泛化能力。

六、问题与收获

6.1 遇到的问题

问题：在进行数据预处理时，一开始与 3.3 的数据处理保持一致，但是一直报错
解决：后面发现是 label 的问题，在 3.3 中，label 为列数据，但我这里一开始读入数据时保存为了数组，没有进行重塑，导致报错。

6.2 收获

通过本次实验，我尝试了不同的超参数组合。通过调整超参数和优化数据预处理步骤，我们成功地将 Breast Cancer Coimbra 数据集的平均错误率从 0.5227 降低到 0.4462，将 Pima Indians Diabetes Database 数据集的平均错误率保持在 0.3482。我发现迭代次数、学习率和正则化强度对模型性能有显著影响。选择合适的超参数组合可以显著提高模型的预测效果。这让我深刻意识到了调参在机器学习中的重要性。

通过十折交叉验证，能够全面评估模型在不同数据子集上的性能，计算平均错误率，并打印分类报告。这有助于了解模型的泛化能力和稳定性。

七、参考资料

[1] 10 折交叉验证（10-fold Cross Validation）与留一法（Leave-One-Out）_十折交叉验证-CSDN 博客
[2] 【机器学习】模型评估与选择（留出法、交叉验证法、查全率、查准率、偏差、方差）_留出法和十折交叉验证-CSDN 博客
[3] 10 折交叉验证（10-fold Cross Validation）与留一法（Leave-One-Out）_十折交叉验证-CSDN 博客

题目 3.5

一、题目理解

采用留出法实现线性判别分析，给出西瓜数据集 3.0a 的性能评估结果。留出法是一种简单的交叉验证方法，将数据集划分为训练集和测试集，使用训练集训练模型，然后在测试集上评估模型的性能。使用 math 数学库手动编写线性判别分析算法代码，对西瓜数据集 3.0a 进行预测准确率等模型评估。同时使用 sklearn 机器学习库进行代码实现，比较两种实现方式的性能评估和效果。

二、线性判别分析算法原理阐述

LDA（线性判别分析）是一个经典的二分类算法。

主要思想：以一种基于降维的方式将所有的样本映射到一维坐标轴上，然后设定一个阈值，将样本进行区分

如下图所示，把红蓝两类的点投影在了一条直线（向量 a）上，即二维变一维（本来一个点要用(x,y)来表示，投影到直线后就用一个维度来描述）。

2.1 设定

首先我们假设整个样本空间分为两个类别，分别是 1、-1；N1、N2 分别代表 1，-1 类别样本的个数；样本为 X。

那么有：

$$|X_{C1}| = N1,\ |X_{C2}| = N2$$

设定 z 为映射后的坐标（即投影后的坐标）

2.2 每一类的均值和方差

将样本数据 X 向 w 向量（设定 w 的模长为 1）做投影，则有： $z_i = w^T x_i$

接下来求出映射后的均值和方差（用来衡量样本的类间距离和类内距离）

均值: $\bar{z_1} = \frac{1}{N1}\sum_{i=1}^{N1} w^T x_{C1i}$; $\bar{z_2} = \frac{1}{N2}\sum_{i=1}^{N2} w^T x_{C2i}$

方差: $S_{z1} = \frac{1}{N1}\sum_{i=1}^{N1}(z_i - \bar{z1})^2$; $S_{z2} = \frac{1}{N2}\sum_{i=1}^{N2}(z_i - \bar{z2})^2$

2.3 目标函数

想要得到好的分类模型，即要求类内间距小，类间间距大。即：

类内间距： $S_{z1} + S_{z2}$; 小：；两个类的方差越小，说明样本越密集
类间间距 $(\overline{z_1} - \overline{z_2})^2$; 大：；用两个类的均值的距离说明两个类之间的距离

根据这样的思路构建目标函数：

$$J(w) = \frac{(\overline{z_1} - \overline{z_2})^2}{S_{z1} + S_{z2}}$$

J(w)越大越好，即我们要求的是 $\mathrm{argmax}_w J(w)$

2.4 目标函数的求解

化简目标函数：（将 w 向量与原数据的运算分隔开）

2.5 最终的实践所求

为得到数值解的稳定性，通常对S_w进行奇异值分解（$S_w = U\sum V^T$），再由$S_w^{-1} = V\sum^{-1}U^T$得到S_w^{-1}。

$$\boldsymbol{w} = \mathbf{S}_w^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) .$$

三、线性判别分析算法设计思路

自编码实现 LDA：通过手动计算类内散布矩阵、均值向量等参数，实现 LDA 模型的训练和预测。

使用 sklearn 进行 LDA：利用 `LinearDiscriminantAnalysis` 类，简化模型的训练和预测过程，并提供了更便捷的模型评估和可视化方法。

3.1 自编码实现 LDA 算法设计思路

1. 数据集划分

首先，将数据集划分为训练集和测试集。使用 `model_selection.train_test_split` 函数将数据集按照指定的比例（例如 75%训练集，25%测试集）进行划分。

2. 计算每个类别的均值向量

对于每个类别，计算其均值向量。均值向量是每个特征的平均值。

3. 计算类内散布矩阵

类内散布矩阵（Sw）是每个类别内数据点与其均值向量的协方差矩阵之和。

4. 计算 Sw 的逆矩阵

使用奇异值分解（SVD）来计算 Sw 的逆矩阵。

5. 计算参数 w

参数 w 是 Sw 的逆矩阵与两个类别均值向量之差的乘积。

6. 在散点图中绘制 LDA 线

计算 LDA 线的两个端点，并在散点图中绘制 LDA 线。

7. 预测测试集并评估结果

使用计算得到的参数 w 对测试集进行预测，并根据阈值进行分类。然后评估模型的准确率、精确率、召回率和 F1 分数。

8. 在线上绘制投影点

计算每个数据点在 LDA 线上的投影点，并在散点图中绘制这些投影点。

3.2 使用 sklearn 进行 LDA 算法设计思路

1. 数据集划分

同样，将数据集划分为训练集和测试集。

2. 模型拟合

使用 `LinearDiscriminantAnalysis` 类来拟合 LDA 模型。

3. 模型验证

使用拟合好的模型对测试集进行预测，并计算模型的性能指标。

4. 绘制分类器的决策边界

使用 `meshgrid` 生成网格点，并使用模型预测这些点的类别。然后绘制决策边界。

5. 绘制训练点

在图中绘制训练数据点。

6. 计算并绘制决策边界

计算 LDA 模型的决策边界的斜率和截距，并在图中绘制决策边界。

四、代码结构及核心部分介绍

4.1 整体代码结构

（1）数据加载与预处理：读取数据集并分离特征和标签。

（2）自定义 LDA 实现：

计算每个类别的均值。

计算类内散布矩阵（Sw）。

使用奇异值分解（SVD）求解投影向量 w。

绘制散点图及 LDA 线。

预测测试集，并评估模型性能（准确率、F1 分数等）。

（3）使用 sklearn 进行 LDA：

数据划分。

训练 LDA 模型。

模型验证与性能总结。

绘制决策边界。

4.2 核心部分介绍

4.2.1 手写代码部分

（1）类内散布矩阵计算：

```
for i in range(m):
    x_tmp = X_train[i].reshape(n, 1)
    u_tmp = u[0].reshape(n, 1) if y_train[i] == 0 else u[1].reshape(n, 1)
    Sw += np.dot(x_tmp - u_tmp, (x_tmp - u_tmp).T)
```

该部分计算每个样本与其对应均值的偏差，并累加得到类内散布矩阵。

（2）投影方向计算：

```
w = np.dot(Sw_inv, (u[0] - u[1]).reshape(n, 1))
```

这里通过类内散布矩阵的逆乘以类间均值差异，得到投影向量。

4.2.1 sklearn 核心代码介绍

（1）模型拟合：

```
lda_model = LinearDiscriminantAnalysis(solver='lsqr', shrinkage=None).fit(X_train, y_train)
```

使用 sklearn 的 LDA 模块拟合训练数据。
（2）模型验证：
y_pred = lda_model.predict(X_test)print(metrics.confusion_matrix(y_test, y_pred))print(metrics.classification_report(y_test, y_pred))

通过混淆矩阵和分类报告评估模型性能。

五、实验流程、测试结果及分析

5.1 流程与测试结果

分别测试手写实现 LDA 版本和利用 sklearn 实现版本，修改 test_size 值，分别令 test_size =0.5，0.35，0.25,分析比较不同模型的不同 test_size 的评估结果。

5.1.1 模型评估结果

（1）sklearn,test_size=0.25
准确率：0.40

	precision	recall	f1-score	support
0.0	0.50	0.33	0.40	3
1.0	0.33	0.50	0.40	2
accuracy			0.40	5
macro avg	0.42	0.42	0.40	5
weighted avg	0.43	0.40	0.40	5

（2）sklearn,test_sieze=0.35
准确率：0.50

	precision	recall	f1-score	support
0.0	0.50	0.67	0.57	3
1.0	0.50	0.33	0.40	3
accuracy			0.50	6
macro avg	0.50	0.50	0.49	6
weighted avg	0.50	0.50	0.49	6

（3）sklearn,test_size=0.5
准确率：0.67

	precision	recall	f1-score	support
0.0	0.75	0.60	0.67	5
1.0	0.60	0.75	0.67	4
accuracy			0.67	9
macro avg	0.68	0.68	0.67	9
weighted avg	0.68	0.67	0.67	9

（4）手动实现 LDA，test_size=0.25

准确率：0.60

测试集准确率: 0.60

	precision	recall	f1-score	support
0.0	0.60	1.00	0.75	3
1.0	0.00	0.00	0.00	2
accuracy			0.60	5
macro avg	0.30	0.50	0.38	5
weighted avg	0.36	0.60	0.45	5

（5）手动实现 LDA，test_size=0.35

准确率：0.50

测试集准确率: 0.50

	precision	recall	f1-score	support
0.0	0.50	1.00	0.67	3
1.0	0.00	0.00	0.00	3
accuracy			0.50	6
macro avg	0.25	0.50	0.33	6
weighted avg	0.25	0.50	0.33	6

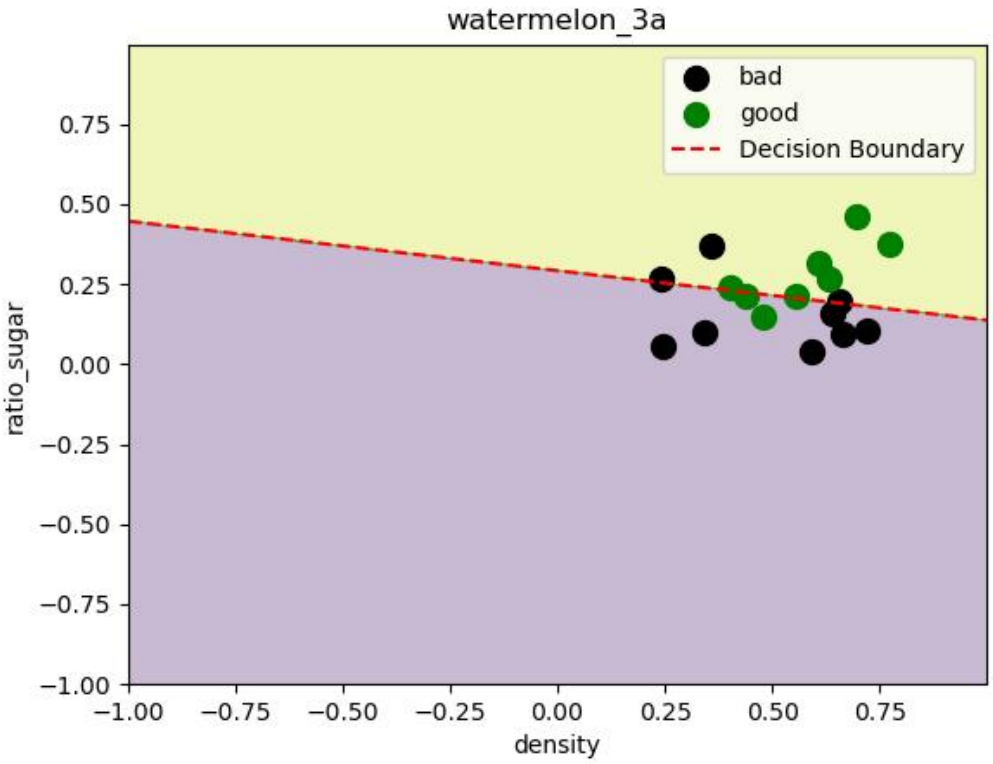
（6）手动实现 LDA，test_size=0.5

准确率：0.33

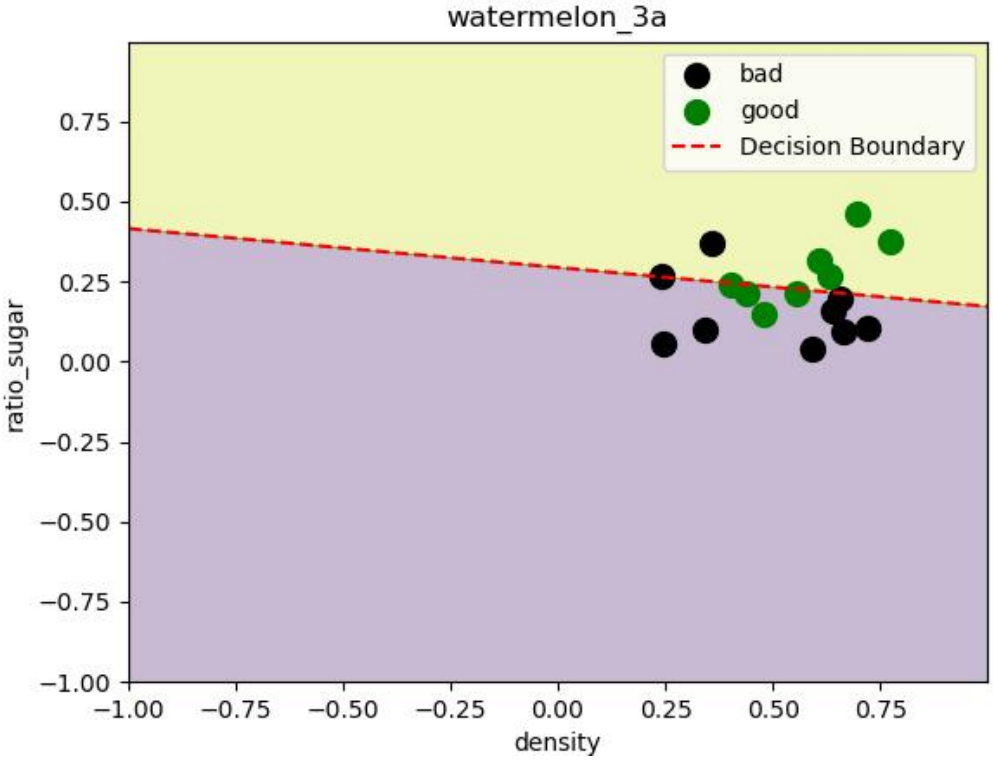
```
测试集准确率: 0.33
              precision    recall  f1-score   support

         0.0       0.40      0.40      0.40         5
         1.0       0.25      0.25      0.25         4

    accuracy                           0.33         9
   macro avg       0.33      0.33      0.33         9
weighted avg       0.33      0.33      0.33         9
```

5.1.2 决策边界结果

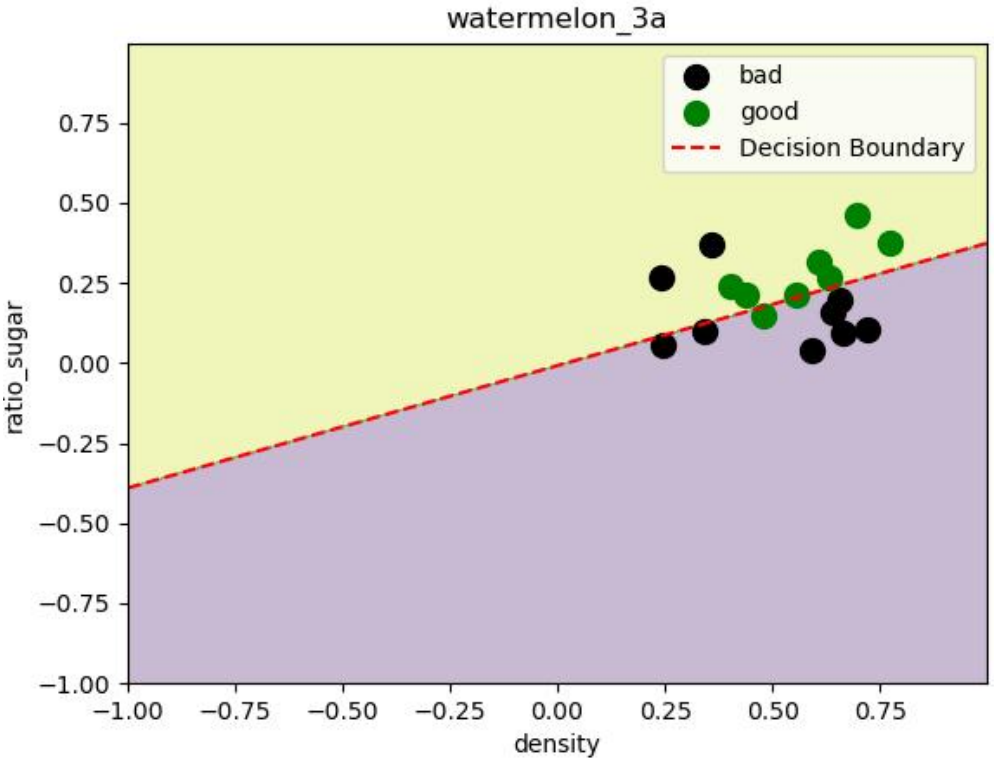
（1）sklearn,test_size=0.25



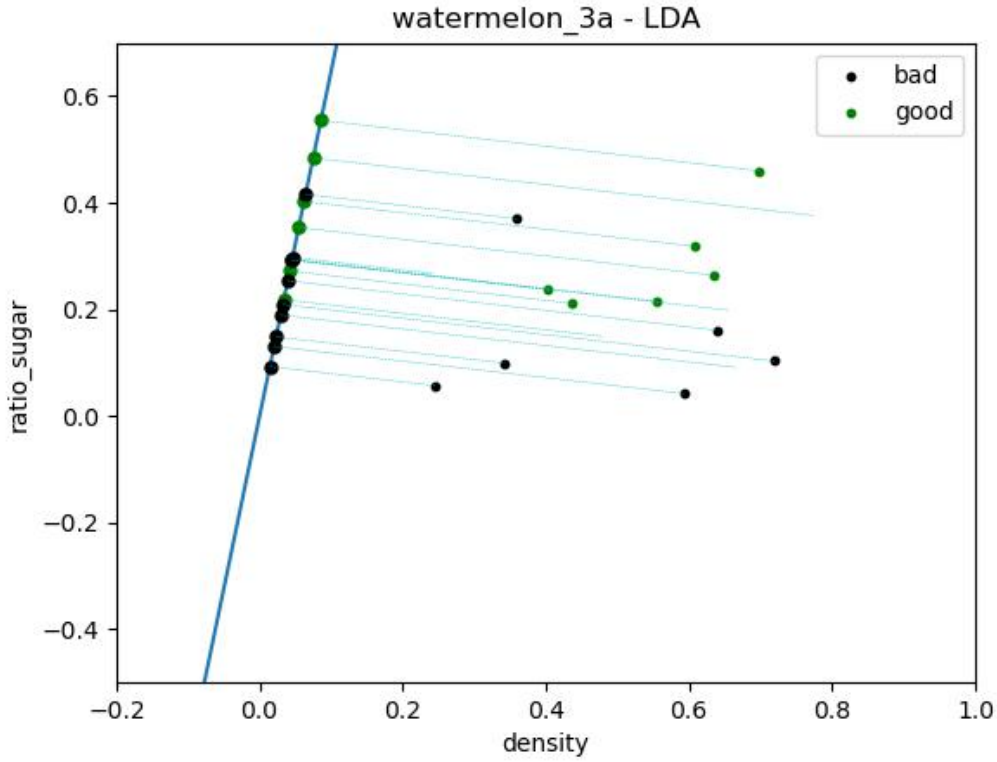
（2）sklearn,test_size=0.35



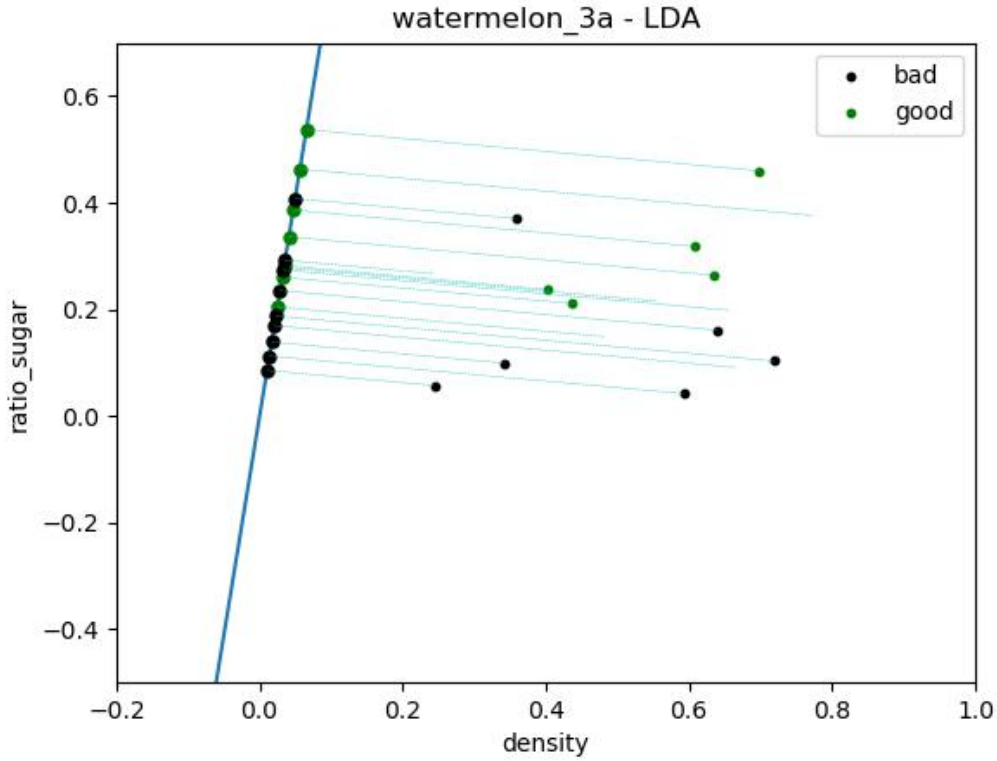
（3）sklearn,test_size=0.5



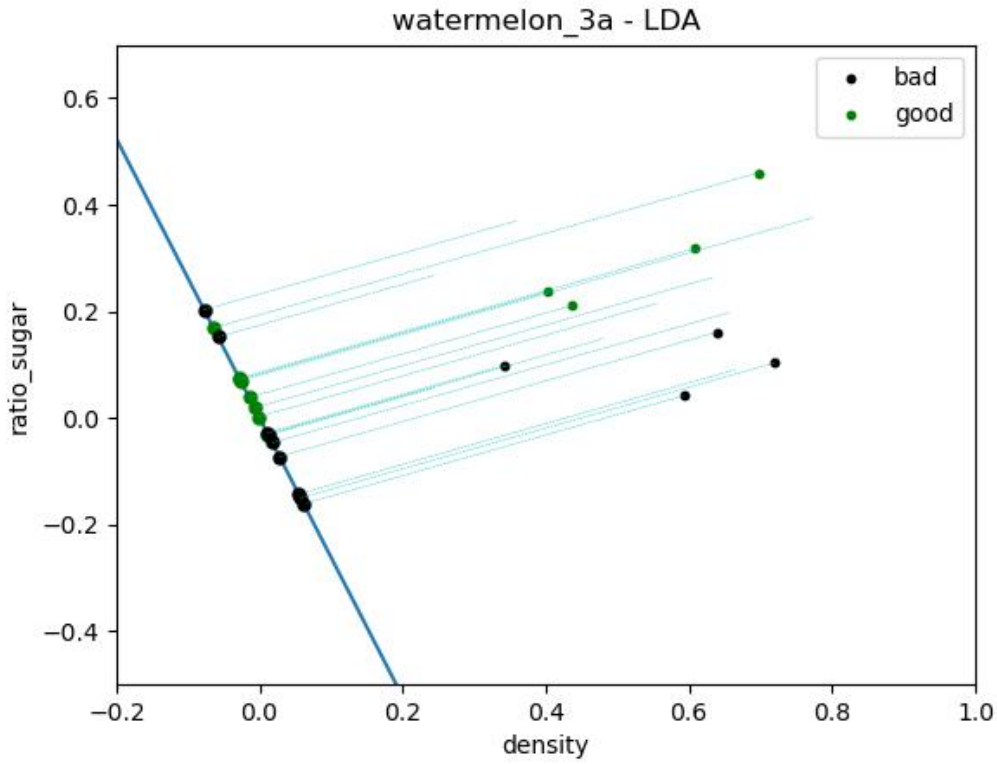
（4）手动实现 LDA，test_size=0.25



（5）手动实现 LDA，test_size=0.35



（6）手动实现 LDA，test_size=0.5



5.2 结果分析

1. sklearn 实现 LDA

test_size	准确率
0.25	0.40
0.35	0.50
0.5	0.67

从 sklearn 实现 LDA 的结果来看，随着测试集比例的增加（从 0.25 到 0.5），模型的准确率逐渐提高。这表明在更大的测试集上，模型能够更好地泛化，可能是由于更大的测试集提供了更多的数据来评估模型的性能。

2. 手动实现 LDA

test_size	准确率
0.25	0.60
0.35	0.50
0.5	0.33

从手动实现 LDA 的结果来看，准确率的变化趋势与 sklearn 实现的结果相反。随着测试集比例的增加（从 0.25 到 0.5），模型的准确率逐渐降低。这可能表明手动实现的 LDA 模型在较小的测试集上表现较好，但在更大的测试集上表现较差。

比较分析

3.测试集大小对模型性能的影响：

sklearn 实现 LDA：随着测试集大小的增加，模型的准确率逐渐提高。这表明 sklearn 实现的 LDA 模型在更大的测试集上能够更好地泛化。

手动实现 LDA：随着测试集大小的增加，模型的准确率逐渐降低。这可能表明手动实现的 LDA 模型在较小的测试集上表现较好，但在更大的测试集上表现较差。

4.模型稳定性：

sklearn 实现 LDA：在不同测试集大小下，模型的准确率变化较为稳定，且整体趋势是上升的。

手动实现 LDA：在不同测试集大小下，模型的准确率变化较大，且整体趋势是下降的。这表明手动实现的 LDA 模型可能不够稳定，容易受到测试集大小的影响。

六、问题与收获

6.1 遇到的问题

问题：在进行手动实现 LDA 模型的模型评估时，我的 y_pred 与 y_pred_classes 的类型均为 dataframe 类型，无法进行模型评估。
解决方案：后续我发现机器学习库在进行模型评估时，输入数据必须为 np.array 类型，于是我转换了一下数据类型。

6.2 收获

通过本次实验，我深入了解了线性判别分析（LDA）的原理和实现方式，并对比了手动实现和使用 sklearn 库实现 LDA 的效果。通过手动实现 LDA，我更深入地理解了 LDA 算法的理论基础，包括类内散布矩阵、类间散布矩阵、均值向量等概念。手动实现 LDA 需要进行大量的数学推导和计算，这让我对 LDA 的数学原理有了更深刻的理解。在手动实现 LDA 的过程中，我熟练掌握了 NumPy 库中的矩阵运算，如矩阵乘法、逆矩阵计算、奇异值分解等。本次实验让我使用准确率、精确率、召回率和 F1 分数等指标来评估模型的性能的技艺更加娴熟。通过本次实验，我更加了解了 sklearn 中 LDA 模型的参数设置，如 solver 和 shrinkage，并学会了如何根据数据集的特点选择合适的参数。

七、参考资料

[1]线性判别分析 (sebastianraschka.com)
[2]线性判别分析 LDA 原理总结 - 刘建平 Pinard - 博客园 (cnblogs.com)
[3]线性判别分析(LDA)详解-CSDN 博客