计算机学院

机器学习实验报告

实验 2：回归分析

姓名：王旭

学号：2312166

专业：计算机科学与技术

2025 年 10 月 30 日

目录

1 实验名称

回归分析

2 实验目标

1. 从零开始编程实现线性回归的核心算法，理解其数学原理。
2. 掌握并对比解析解（正规方程）与迭代解（梯度下降）的求解过程与优劣。
3. 在实验中理解学习率、正则化、模型复杂度等关键因素对模型性能的影响。
4. 学会通过分析训练/测试误差、绘制收敛曲线等方式来评估和诊断模型。

3 基础任务 1: 线性回归–最小二乘法

3.1 任务要求:

1. 根据数据集，利用正规方程（Normal Equation）求得最小二乘解 $\theta = (X^{\top}X)^{-1}X^{\top}y$
2. 使用你得到的回归方程，自行构造 5 个新的测试样本点并进行预测。
3. 画出训练数据的散点图和拟合的回归直线，给出训练误差和测试误差。

3.2 代码实现:

正规方程 normal_equation_fit 函数实现

```python
def normal_equation_fit(x_train: np.ndarray, y_train: np.ndarray):
    """
    使用正规方程求解线性回归参数
    参数:
        x_train: 训练特征 (m,)
        y_train: 训练目标 (m,)
    返回:
        w: 斜率
        b: 截距
    """
    # 1. 构造设计矩阵 Xb = [x, 1]
    # 添加偏置项（全1列）
    Xb = np.column_stack((x_train, np.ones_like(x_train)))

    # 2. 计算正规方程:  = (Xb^T Xb)^{-1} Xb^T y
    # 使用伪逆提高数值稳定性
    theta = np.linalg.pinv(Xb.T @ Xb) @ Xb.T @ y_train

    # 3. 提取参数
    w = theta[0]  # 斜率
    b = theta[1]  # 截距
```

```

    return w, b
```

首先我们先构建好我们使用正规方程求解所需要的矩阵 X_b，X_{train} 是原始特征向量，np.ones_like(X_{train}) 是创建与 X_{train} 相同长度的全 1 向量，然后通过 np.column_stack 将两列并排组合成矩阵，构造好矩阵 X_b。

然后进行正规方程的计算，并使用 np.linalg.pinv() 函数计算伪逆矩阵，比 inv() 更稳定。最后从结果 theta 矩阵中提取参数，求得斜率 w 与截距 b。这样我们的正规方程函数就实现了。

3.3 运行结果

在实现正规方程 normal_equation_fit() 函数后，我们运行整体的代码，观察实验结果。

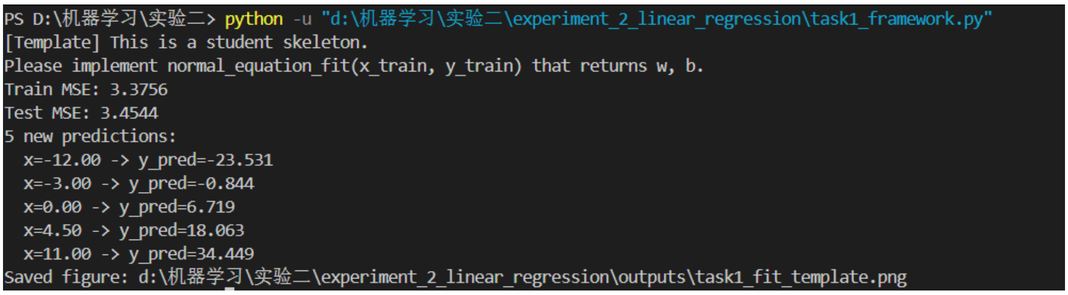


图 3.1: 代码输出结果

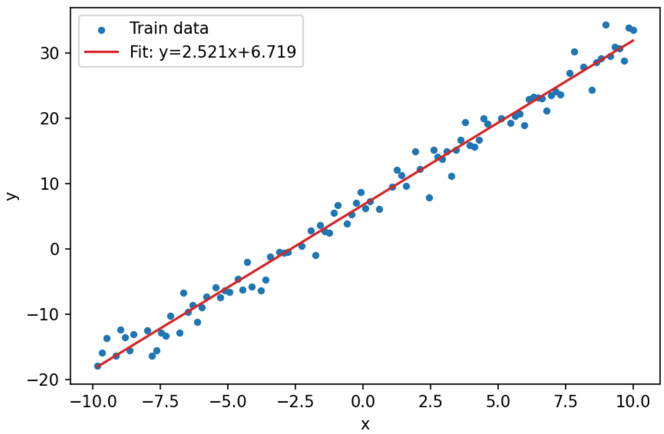


图 3.2: 训练数据散点图与拟合直线

从结果我们可以看出:

1. 训练 MSE 为 3.3756，
2. 测试 MSE 为 3.4544。

训练误差和测试误差接近，说明模型没有过拟合，泛化能力良好，同时训练集和测试集性能一致，模型较为稳定，并且对于 5 个测试样本点来说，预测值随着 x 的增加而线性增加，符合线性回归的预期。从图像来看，直线能够较好地穿过数据点的中心区域，拟合较好。

3.4 思考题:

思考: 正规方程的求解核心是哪一步? 这一步在什么情况下可能会失败?
正规方程的求解核心步骤就是计算 $(X^\top X)^{-1}$（矩阵求逆）这一步。可能失败的情况:

1. 特征之间存在线性关系，导致 $(X^\top X)$ 矩阵不可逆
2. 如果特征数 > 样本数，那么会导致矩阵不是满秩的，不可逆
3. 如果特征向量中各值数值不稳定，差异过大时，矩阵条件数很大，可能求逆失败。

4 基础任务 2: 线性回归 - 梯度下降法

4.1 任务要求:

1. 读取数据集，将其按 4:1 的比例随机划分为训练集和测试集。注意：数据的第一行是表头，分隔符是分号;。
2. 编程实现批量梯度下降（BGD）或随机梯度下降（SGD）算法。
3. 训练你的线性回归模型，并记录下每次迭代后在训练集上的均方误差（MSE），输出最终模型在训练集和测试集上的 MSE。
4. 可视化：画出训练过程中的 MSE 收敛曲线（横轴为迭代次数，纵轴为 MSE）。

4.2 代码实现:

BGD 批量梯度下降算法 gd_train() 实现

```python
def gd_train(X: np.ndarray, y: np.ndarray, lr=0.01, epochs=200):
    """
    使用批量梯度下降训练线性回归模型
    参数:
        X: 特征矩阵 (m, n)
        y: 目标向量 (m,)
        lr: 学习率
        epochs: 迭代次数
    返回:
        w: 权重向量 (n,)
        b: 偏置项
        history_mse_list: 每次迭代的MSE历史记录
    """
    m, n = X.shape  # m: 样本数, n: 特征数
    # 初始化参数
    w = np.zeros(n)  # 权重初始化为0
    b = 0.0          # 偏置初始化为0
    # 记录每次迭代的MSE
    history_mse = []

    # 添加偏置项到特征矩阵（可选方案，这里我们显式处理b）
```

```
    # 另一种方案是将b作为w的一部分，这里选择分开处理更清晰

    # 批量梯度下降迭代
    for epoch in range(epochs):
        # 计算预测值
        y_pred = X @ w + b
        # 计算误差
        error = y_pred - y
        # 计算当前MSE并记录
        mse = np.mean(error ** 2)
        history_mse.append(mse)
        # 计算梯度
        # 对w的梯度: dJ/dw = (1/m) * X^T @ (y_pred - y)
        dw = (1/m) * (X.T @ error)
        # 对b的梯度: dJ/db = (1/m) * sum(y_pred - y)
        db = (1/m) * np.sum(error)
        # 更新参数
        w = w - lr * dw
        b = b - lr * db
        # 每50轮打印一次进度
        if epoch % 50 == 0:
            print(f'Epoch {epoch}, MSE: {mse:.4f}')

    return w, b, history_mse
```

首先在取得样本数 m 与特征数 n 之后进行参数初始化，将权重向量 w(n,) 初始化为全 0，偏重 b 初始化为 0, 并使用 history_mse 作为记录每次迭代的 MSE 的表。

下面就开始进入批量梯度下降迭代的循环，首先使用矩阵乘法计算预测值：$\hat{y} = Xw + b$，计算误差与当前 mse 后，开始本次梯度的计算:

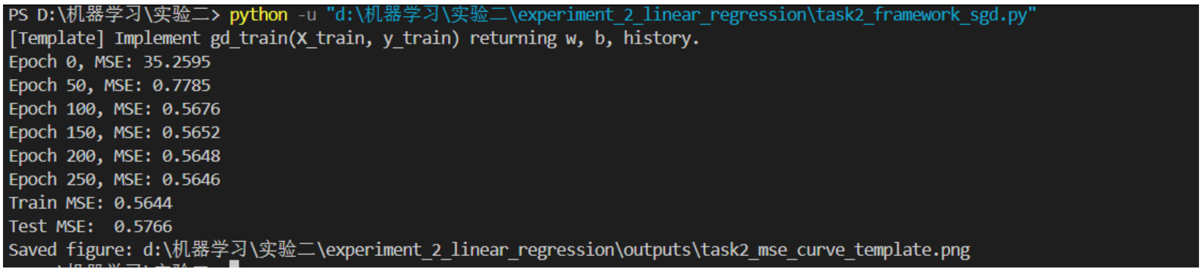
$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{m} \mathbf{X}^\top (\mathbf{X}\boldsymbol{w} + \boldsymbol{b} - \boldsymbol{y})$$

$$\nabla_b J(\boldsymbol{b}, \boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{X}\boldsymbol{w} + \boldsymbol{b} - \boldsymbol{y})$$

计算完之后依据梯度进行参数的更新，沿梯度反方向更新参数，学习率控制步长，完成一轮循环。

4.3 运行结果

在实现 BGD 批量梯度下降算法 gd_train() 函数之后，我们运行整体的代码，观察实验结果。



```
PS D:\机器学习\实验二> python -u "d:\机器学习\实验二\experiment_2_linear_regression\task2_framework_sgd.py"
[Template] Implement gd_train(X_train, y_train) returning w, b, history.
Epoch 0, MSE: 35.2595
Epoch 50, MSE: 0.7785
Epoch 100, MSE: 0.5676
Epoch 150, MSE: 0.5652
Epoch 200, MSE: 0.5648
Epoch 250, MSE: 0.5646
Train MSE: 0.5644
Test MSE:  0.5766
Saved figure: d:\机器学习\实验二\experiment_2_linear_regression\outputs\task2_mse_curve_template.png
```

图 4.3: 代码输出结果

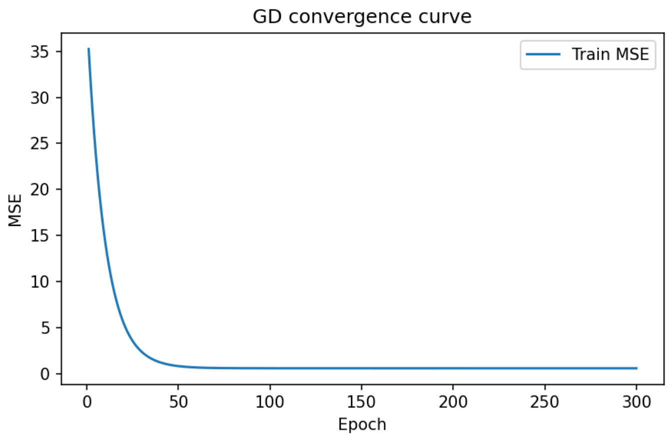


图 4.4: MSE 随迭代次数的收敛曲线

从结果我们可以看出:

1. 初始 MSE 为 35.2595，说明模型初始预测误差很大，但在 50 轮内迅速下降到 0.7785，说明梯度下降在前 50 轮就完成了大部分优化工作
2. 后期 MSE 从 0.7785 缓慢下降到 0.5652，最后 MSE 稳定在 0.5644 左右，下降速度明显变缓，接近最优解
3. 训练 MSE 为 0.5644，测试 MSE 为 0.5766，训练测试误差接近，说明模型没有过拟合，并且泛化能力良好

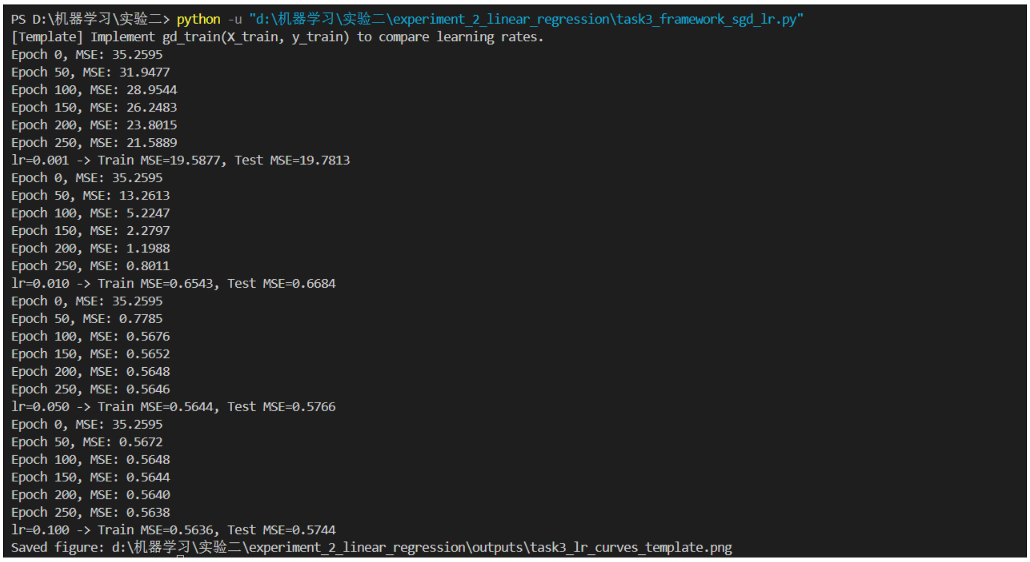
5 中级任务 3: 超参数调优 - 学习率分析

5.1 任务要求:

1. 基于任务 2 的代码，尝试几个不同的学习率。
2. 为每个学习率绘制训练过程的 MSE 收敛曲线。
3. 分析并解释：分析最佳学习率，思考学习率过大或过小分别对模型收敛过程产生了什么影响?

5.2 运行结果

在任务 2 中实现了 BGD 批量梯度下降算法 gd_train() 函数之后，我们将任务 2 的实现复制到任务 3 中，然后进行整体的学习率参数调优，观察实验结果。



```
PS D:\机器学习\实验二> python -u "d:\机器学习\实验二\experiment_2_linear_regression\task3_framework_sgd_lr.py"
[Template] Implement gd_train(X_train, y_train) to compare learning rates.
Epoch 0, MSE: 35.2595
Epoch 50, MSE: 31.9477
Epoch 100, MSE: 28.9544
Epoch 150, MSE: 26.2483
Epoch 200, MSE: 23.8015
Epoch 250, MSE: 21.5889
lr=0.001 -> Train MSE=19.5877, Test MSE=19.7813
Epoch 0, MSE: 35.2595
Epoch 50, MSE: 13.2613
Epoch 100, MSE: 5.2247
Epoch 150, MSE: 2.2797
Epoch 200, MSE: 1.1988
Epoch 250, MSE: 0.8011
lr=0.010 -> Train MSE=0.6543, Test MSE=0.6684
Epoch 0, MSE: 35.2595
Epoch 50, MSE: 0.7785
Epoch 100, MSE: 0.5676
Epoch 150, MSE: 0.5652
Epoch 200, MSE: 0.5648
Epoch 250, MSE: 0.5646
lr=0.050 -> Train MSE=0.5644, Test MSE=0.5766
Epoch 0, MSE: 35.2595
Epoch 50, MSE: 0.5672
Epoch 100, MSE: 0.5648
Epoch 150, MSE: 0.5644
Epoch 200, MSE: 0.5640
Epoch 250, MSE: 0.5638
lr=0.100 -> Train MSE=0.5636, Test MSE=0.5744
Saved figure: d:\机器学习\实验二\experiment_2_linear_regression\outputs\task3_lr_curves_template.png
```

图 5.5: 代码输出结果

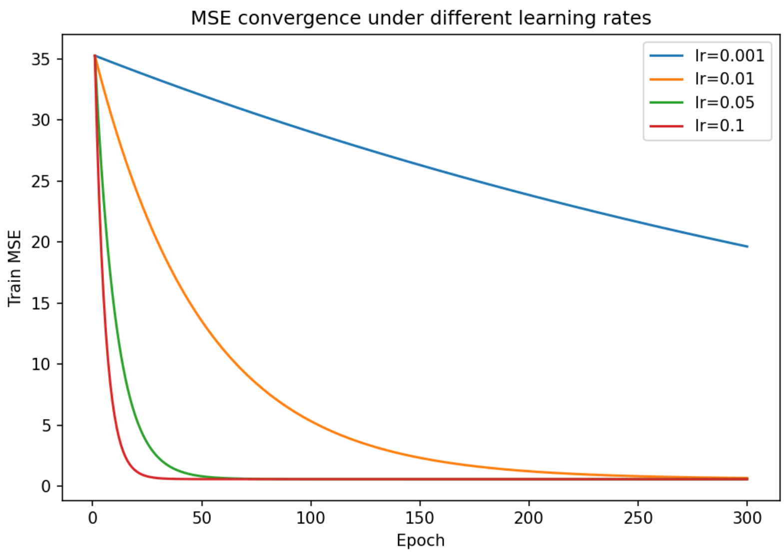


图 5.6: 多学习率下的 MSE 随迭代次数的收敛曲线对比图

学习率	最终训练 MSE	最终测试 MSE	收敛速度	稳定性	评价
0.001	19.5877	19.7813	极慢	很高	学习率过小
0.01	0.6543	0.6684	较慢	高	偏小但可用
0.05	0.5644	0.5766	较快	高	**最佳选择**
0.1	0.5636	0.5744	最快	中等	良好但轻微震荡

表 1: 不同学习率下的模型 MSE 与性能

5.2.1 最佳学习率的判断

对于 lr=0.05：50 轮内快速收敛到 0.78，随后平稳下降到 0.56。对于 lr=0.1：收敛速度更快，但初期有轻微震荡。两者最终精度相近，都达到了较好的优化效果。

而我们目标的合适学习率希望能够在收敛速度和稳定性间取得平衡，既能快速接近最优点，又不会因步长过大而震荡，因此:

1. lr=0.05 更稳定，适合对稳定性要求高的场景
2. lr=0.1 收敛更快，适合对训练速度敏感的场景

并且 lr=0.05 的情况下，50 轮内完成主要优化，收敛速度适中，并且收敛过程平滑无震荡，稳定性较好，最终也能够达到较低的 MSE(0.5644)，并且泛化能力强，测试误差与训练误差接近，因此综合考虑来说，最佳学习率我们定为 0.05。

5.2.2 学习率过大/过小对模型收敛过程的影响

学习率过小 (lr=0.001) 的情况下，300 轮后 MSE 仅从 35.26 降到 19.59，收敛极其缓慢，并且从图像上来看，曲线几乎呈直线缓慢下降，没有明显的加速过程，并且与其他学习率相比，最终训练误差 MSE 远高于其他学习率的最终训练 MSE。

因此学习率过小时，会产生如下对收敛过程的影响:

1. 收敛速度极慢，计算效率低下，在有限迭代次数内无法达到满意精度
2. 可能陷入较差的局部最优而无法跳出

学习率过大时 (lr=0.1)，我们发现即使在 lr=0.1 时，模型也没有发散，这说明我们的数据特征标准化效果良好，使得梯度不会过大，但是 lr=0.1 时曲线的轻微震荡提示已接近稳定边界，理论上有发散风险，参数更新步长过大会越过最优点，并且也可能存在收敛过程不稳定的风险，可能在不同局部最优间震荡。

6 高级任务 4: 正则化 - 岭回归

6.1 任务要求:

1. 使用任务 2 的数据集 winequality-white.csv。
2. 编程实现岭回归（Ridge Regression）。你可以选择解析法、批量梯度下降法或随机梯度下降法实现。
3. 设置一个正则化参数。
4. 训练模型并计算最终在训练集和测试集上的平均误差。
5. （选做）尝试不同的正则化参数值，观察它对模型参数（权重）大小以及测试误差的影响。

6.2 代码实现:

岭回归的闭式解算法 ridge_fit_closed_form 的实现

```
def ridge_fit_closed_form(X: np.ndarray, y: np.ndarray, lam: float) -> np.ndarray:
    """
    使用岭回归的闭式解求解参数
```

```
    参数:
        X: 设计矩阵 (已包含偏置项)
        y: 目标向量
        lam: 正则化参数
    返回:
        theta: 参数向量 [w, b]
    """
    m, n = X.shape  # m: 样本数, n: 特征数(包括偏置项)

    # 构造正则化矩阵:  I
    # 注意: 通常不对偏置项进行正则化, 所以最后一个元素设为0
    I = np.eye(n)
    I[-1, -1] = 0  # 不对偏置项正则化

    # 计算岭回归闭式解:   = (X^T X +  I)^(-1) X^T y
    # 使用伪逆提高数值稳定性
    theta = np.linalg.pinv(X.T @ X + lam * I) @ X.T @ y

    return theta
```

首先在取得样本数 m 与特征数 n 之后进行正则化矩阵构造，np.eye(n) 创建 $n \times n$ 的单位矩阵，I[-1, -1] = 0 将最后一个对角线元素设为 0，表示不对偏置项进行正则化

然后计算岭回归闭式解，并使用伪逆 np.linalg.pinv 提高数值稳定性

$$\boldsymbol{\theta} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

6.3 运行结果

在实现岭回归闭式解算法 ridge_fit_closed_form 函数后，我们运行整体的代码，观察实验结果。

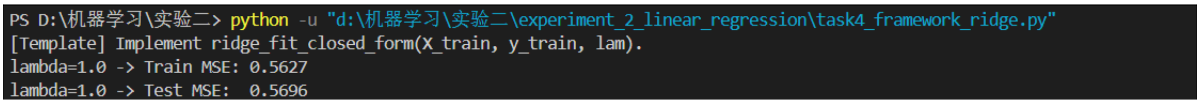


图 6.7: 代码输出结果

从结果我们可以看出 (当前 $\lambda = 1$ 的情况下):

1. 训练 MSE: 0.5627
2. 测试 MSE: 0.5696

与任务 2 的普通线性回归对比：

模型	训练 MSE	测试 MSE	差异
普通线性回归	0.5644	0.5766	0.0122
岭回归 ($\lambda = 1.0$)	0.5627	0.5696	**0.0069**

表 2: 岭回归与普通线性回归的误差比较

我们可以看出：测试误差从 0.5766 降到 0.5696，下降了约 1.2%，并且训练误差与测试误差的差异从 0.0122 降到 0.0069，泛化差距缩小。这说明岭回归确实起到了改善泛化能力的作用。

6.4 扩展：尝试不同的正则化参数值

尝试不同的正则化参数值，观察它对模型参数（权重）大小以及测试误差的影响。

我们修改 main 函数中对于 λ 的设置以及后续的 MSE 计算，修改为如下的多个不同正则化参数 λ 值。

岭回归的闭式解算法采用不同的正则化参数值

```python
def main():
    df = pd.read_csv(CSV, sep=';')
    X = df.iloc[:, :-1].to_numpy().astype(float)
    y = df.iloc[:, -1].to_numpy().astype(float)

    X_train, y_train, X_test, y_test = train_test_split(X, y, test_ratio=0.2)
    X_train, X_test = normalize(X_train, X_test)

    print('[Template] Implement ridge_fit_closed_form(X_train, y_train, lam).')

    # 测试不同的正则化参数
    lambda_values = [0, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

    # 追加常数列用于偏置
    X_train_ext = np.column_stack([X_train, np.ones(X_train.shape[0])])
    X_test_ext = np.column_stack([X_test, np.ones(X_test.shape[0])])

    print(" 值\t训练MSE\t测试MSE\t参数范数\t泛化差距")
    print("-" * 50)

    train_errors = []
    test_errors = []
    param_norms = []

    for lam in lambda_values:
        theta = ridge_fit_closed_form(X_train_ext, y_train, lam)
        w, b = theta[:-1], float(theta[-1])

        # 计算预测和误差
        y_train_pred = X_train @ w + b
        y_test_pred = X_test @ w + b
        train_mse = mse(y_train, y_train_pred)
        test_mse = mse(y_test, y_test_pred)

        # 计算参数范数（不包括偏置项）
        param_norm = np.linalg.norm(w)
        generalization_gap = test_mse - train_mse

```

```
        train_errors.append(train_mse)
        test_errors.append(test_mse)
        param_norms.append(param_norm)

        print(f"{lam}\t{train_mse:.4f}\t{test_mse:.4f}\t{param_norm:.4f}\t{generalization_gap:.4f}")
```

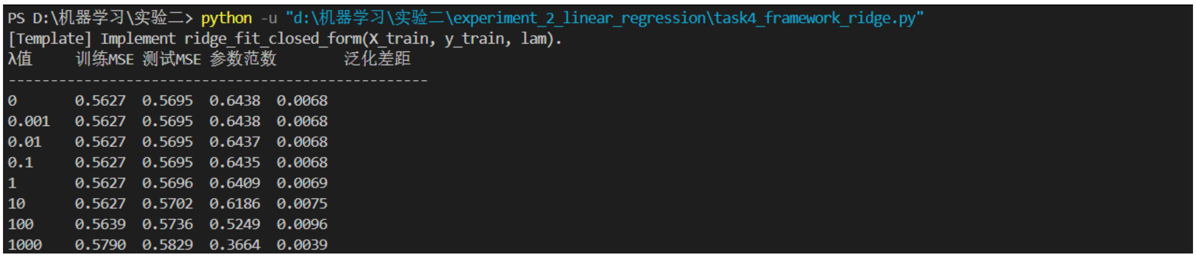


图 6.8: 代码输出结果

λ 值	训练 MSE	测试 MSE	参数范数	泛化差距
0	0.5627	0.5695	0.6438	0.0068
0.001	0.5627	0.5695	0.6438	0.0068
0.01	0.5627	0.5695	0.6437	0.0068
0.1	0.5627	0.5695	0.6435	0.0068
1	0.5627	0.5696	0.6409	0.0069
10	0.5627	0.5702	0.6186	0.0075
100	0.5639	0.5736	0.5249	0.0096
1000	0.5790	0.5829	0.3664	0.0039

表 3: 不同正则化参数 λ 对岭回归性能的影响

从输出结果我们可以看出，岭回归所产生的正则化效果可以分为三种情况:

6.4.1 弱正则化区域（λ = 0 - 10）

在 λ = 0 - 10 的情况下，训练/测试误差几乎不变，参数范数轻微下降，从 0.6438 下降到 0.6186，说明在这个范围内，正则化对模型性能影响很小，原始模型已经比较优化。

6.4.2 中等正则化区域（λ =100）

在 λ = 100 的情况下，误差开始上升，训练误差上升到 0.5639，测试误差上升到 0.5736，并且范数降至 0.5249，参数明显压缩。且泛化差距增大，变为 0.0096，说明开始出现轻微欠拟合迹象。

6.4.3 强正则化区域（λ = 1000）

在 λ = 1000 的情况下，误差显著上升，训练误差 0.5790，测试误差 0.5829，参数大幅压缩，范数降至 0.3664，泛化差距减小为 0.0039，但绝对误差很高，说明明显欠拟合，模型过于简单。

因此对于最优 λ 值的分析，最佳性能应该是 (λ = 0 - 0.1) 范围内，在这种情况下，测试 MSE 最低，训练 MSE 稳定，且泛化差距最小。