姓名：毛九弢　学号：221900175　2024 年 10 月 31 日

零. 实验环境

- 操作系统：Windows 11
- Python 版本：3.9.19
- 依赖库：见 requirements.txt

一. (30 points) 类别不平衡 [本题题面对正负样本的描述描述反了]

信用卡欺诈检测数据集 (Credit Card Fraud Detection) 包含了 2013 年 9 月通过信用卡进行的欧洲持卡人的交易。这是一个非常典型的类别不平衡数据集，数据集中正常交易的标签远多于欺诈交易。请你根据附件中提供的该数据集完成以下问题：

参考链接: `https://zhuanlan.zhihu.com/p/134091240`, `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`, `https://www.cnblogs.com/linjingyg/p/15708635.html`.

代码说明

- 代码见压缩包 code 中的 Prob1, 详细说明见 Prob1 中的 README 文件;
- 详细输出见 Prob1 中的 out 文件夹。

Task1

该数据集共有 284807 个样本，其中只有 492 个负样本。请按照训练集和测试集比例 4：1 的方式划分数据集（使用固定的随机种子）。在训练集上训练 SVM 模型，并计算该模型在测试集上的精度（如准确率、召回率、F1 分数，AUC 等）。请展示 SVM 模型训练过程的完整代码，并绘制 ROC 曲线（8 points）；

代码见压缩包 code 中的 Prob1, 详细说明见 Prob1 中的 README 文件

表 1: SVM 模型性能

指标	ACC	PRE	REC	F1	AUC
SVM	0.999350	0.906667	0.693878	0.786127	0.968045

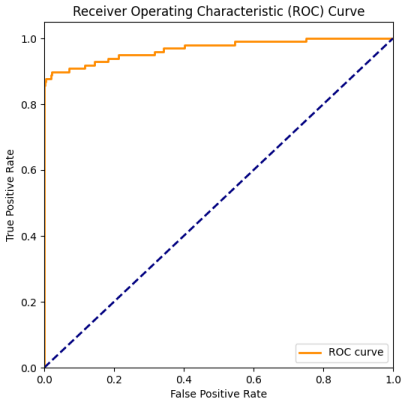


图 1: ROC 曲线

Task2

请从上述训练集中的正样本中分别随机剔除 2000，20000，200000 个样本，剩余的正样本和训练集中原本的负样本共同组成新的训练集，测试集保持不变。请参照上一小问方式在这三个新的训练集上训练 svm 模型，并记录每个模型的精度。观察并比较这几组实验的结果，结合准确率与召回率的定义，请说明不平衡数据集对模型的影响（9 points）；

2.1 结果

表 2: 剔除不同数目样本的 SVM 模型性能

指标	ACC	PRE	REC	F1	AUC
SVM	0.999350	0.906667	0.693878	0.786127	0.968045
SVM-rm2k	0.999350	0.906667	0.693878	0.786127	0.968553
SVM-rm2w	0.999368	0.907895	0.704082	0.793103	0.969577
SVM-rm20w	0.999298	0.802083	0.785714	0.793814	0.973112

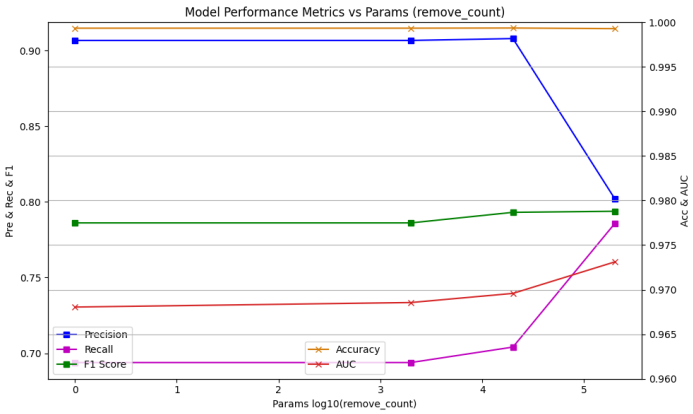


图 2: 变化曲线

剔除不同数目样本的 ROC 曲线

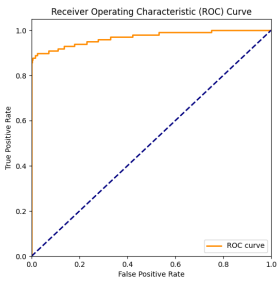


图 3: 剔除 2000 个样本

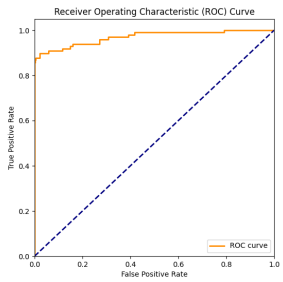


图 4: 剔除 20000 个样本

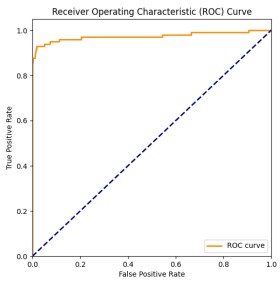


图 5: 剔除 200000 个样本

2.2 回答和结论

PRE 和 REC 的定义： $\text{PRE} = \frac{\text{TP}}{\text{TP+FP}}, \text{REC} = \frac{\text{TP}}{\text{TP+FN}}$

以这个负样本更多的数据集来说，在不进行任何优化处理的情况下，模型在模型没把握确定的时候，更加偏向预测为负样本（因为这样误差期望小），所以 REC 会低；换句话来说，如果模型预测了一个样本为正样本，那么模型一定是有很大把握的，所以 PRE 会高。

简单来说：

- 一个数据集中负样本更多，模型更容易预测为负样本，这样会导致 PRE 高，REC 低。
- 一个数据集中正样本更多，模型更容易预测为正样本，这样会导致 PRE 低，REC 高。

Task3 实现 SMOTE 算法中的 over sampling 函数, 以代码块的形式附于下方即可.

除了上述第 2 问的随机欠采样的方式以外，对小类样本的“过采样”也是处理不平衡问题的基本策略。一种经典的方法为人工合成的过采样技术 (Synthetic Minority Over-sampling Technique, SMOTE), 其在合成样本时寻找小类中某一个样本的近邻, 并在该样本与近邻之间进行差值, 作为合成的新样本。（8 points）;

```python
def over_sampling(self):
    N = self.N # N 为每个样本生成的合成样本数量
    n_synthetic_samples = N * self.n_sample # 计算需要生成的合成样本数量
    synthetic_samples = np.zeros((n_synthetic_samples, self.n)) # 初始化合成样本数组

    # 使用 K 近邻算法找到每个样本的 K 个最近邻
    neigh = NearestNeighbors(n_neighbors=self.K)
    neigh.fit(self.sample)
    neighbors = neigh.kneighbors(self.sample, return_distance=False)

    for i in mytqdm(range(self.n_sample), desc="Generating synthetic samples"):
        for n in range(N):
            nn = np.random.choice(neighbors[i][neighbors[i] != i])
            diff = self.sample[nn] - self.sample[i]
            gap = np.random.rand()
            synthetic_samples[i * N + n] = self.sample[i] + gap * diff
    return synthetic_samples, np.ones(n_synthetic_samples) * self.label
```

Listing 1: 过采样函数实现

Task4 请说明 SMOTE 算法的缺点并讨论可能的改进方案（5 points）。

我的实现中，设置 K 近邻的 K 为 7，并调整不同的 N 值，分别为 3, 5, 7, 15, 30, 50, 100, 200, 400，观察并比较不同 N 值对模型性能的影响.

4.1 结果如下:（N 代表每个样本合成的样本数目）

表 3: 不同 N 的 SVM 模型性能

指标	ACC	PRE	REC	F1	AUC
SVM	0.999350	0.906667	0.693878	0.786127	0.968045
SVM-N3	0.999456	0.913580	0.755102	0.826816	0.968986
SVM-N5	0.999544	0.918605	0.806122	0.858696	0.975347(感觉不正常)
SVM-N7	0.999544	0.918605	0.806122	0.858696	0.969755
SVM-N15	0.999438	0.851064	0.816327	0.833333	0.976791
SVM-N30	0.999192	0.728070	0.846939	0.783019	0.980940
SVM-N50	0.998912	0.636364	0.857143	0.730435	0.981739
SVM-N100	0.998227	0.491329	0.867347	0.627306	0.975618
SVM-N200	0.995558	0.265861	0.897959	0.410256	0.975457
SVM-N400	0.988115	0.116556	0.897959	0.206331	0.969106

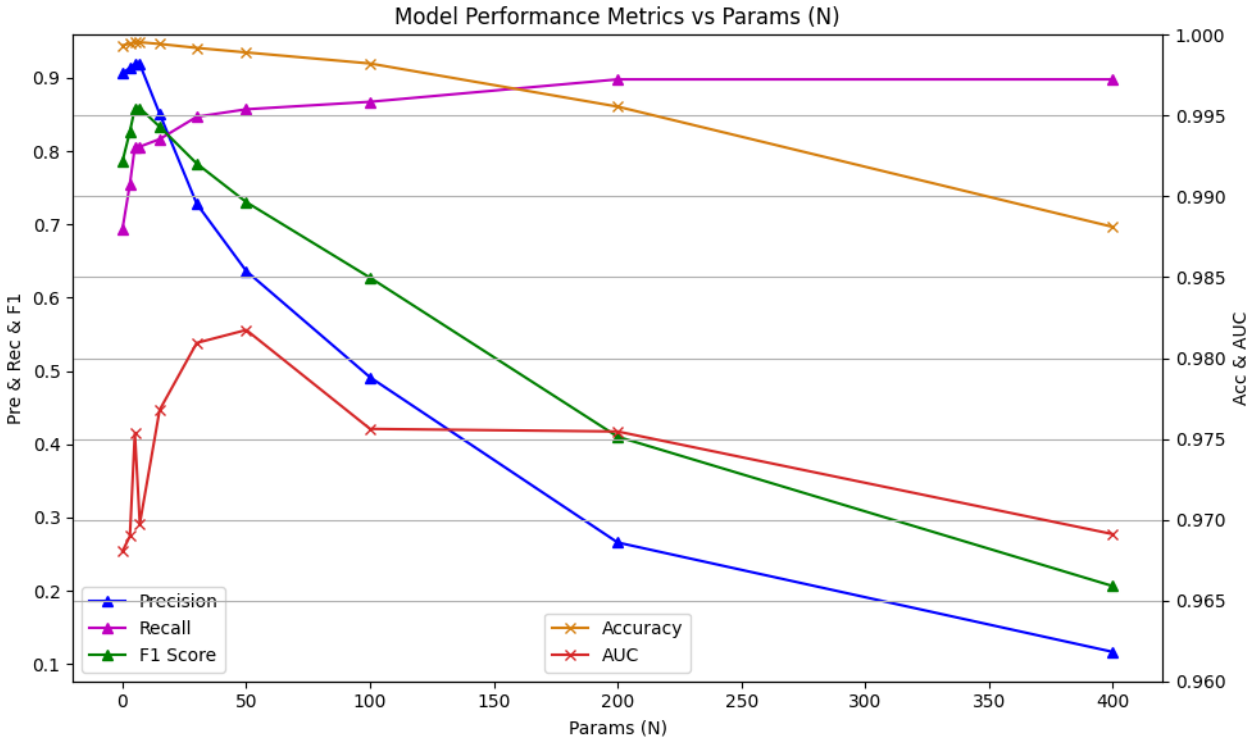


图 6: 变化曲线

不同典型 N 的 ROC 曲线

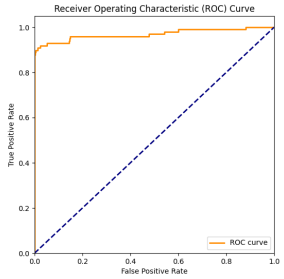


图 7: N=3

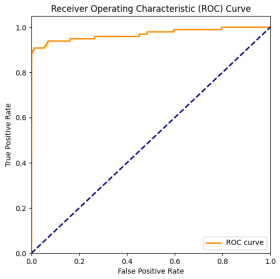


图 8: N=7

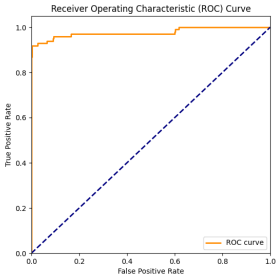


图 9: N=15

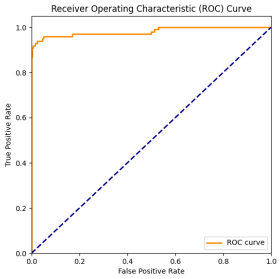


图 10: N=30

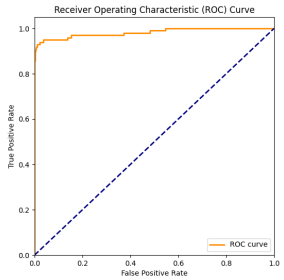


图 11: N=50

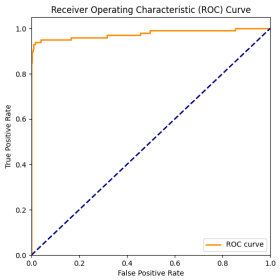


图 12: N=100

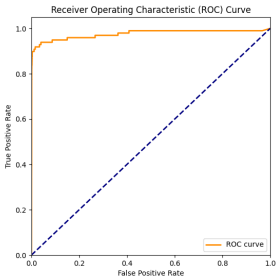


图 13: N=200

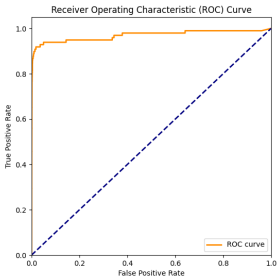


图 14: N=400

4.2 回答和结论

随着 N 的增大，模型的性能逐渐下降，合成样本的质量下降，过近的合成样本造成过拟合，跨边界合成样本成为噪声，导致模型的泛化能力下降。

4.2.1 缺点

- N 过大 K 过小的合成样本可能会引入冗余信息 (全部合成在某个负样本周围)，导致模型泛化能力下降。
- 合成样本可能会引入错误信息和噪声 (比如跨边界合成样本，即两个负样本之间夹着正样本区域, 但是合成样本的 label 却是负的)，导致模型性能下降。
- 将原本就比较大的训练集变得更大，导致训练时间增加。

4.2.2. 可能的改进方案

- 检查是否是跨边界合成样本，如果是则不合成，但是这样计算量会增加。
- 对于 K 近邻的” 邻” 进行限制，比如设置最大距离和最小距离，避免合成样本过于集中造成过拟合或过于分散造成跨类别合成。
- 使用 ENN 删除噪声样本, 以减少基于噪声样本的合成样本。
- 混合采样方法，即同时使用过采样和欠采样方法。
- 调整 K, N 的大小，使得合成样本的数量适中, 质量更高。

二. (20 points) 机器学习中的过拟合现象

本题以决策树和多层感知机为例, 探究机器学习中的过拟合现象。在教材 2.1 节中提到, 机器学习希望训练得到的模型在新样本上保持较好的泛化性能. 如果在训练集上将模型训练得“过好”, 捕获到了训练样本中对分类无关紧要的特性, 会导致模型难以泛化到未知样本上, 这种情况称为过拟合。

代码与数据集的选择

- 代码见压缩包 code 中的 Prob2, 详细说明见 Prob2 中的 README 文件;
- 数据集加载见 Prob2 中的 dataloader.py 文件，详细输出见 Prob2 中的 out 文件夹。

Task1 请简要总结决策树和多层感知机的工作原理及其对应的缓解过拟合的手段（5 points）

1.1 决策树

1.1.1 工作原理:

- 决策树是一种基于树结构的监督学习算法。它将数据递归地划分为多个子集，目的是通过询问一系列的条件问题来实现分类或回归。每个内部节点表示一个属性，边表示属性的划分条件，叶节点表示决策结果。
- 而在选择划分属性时，常用的划分标准包括信息增益、基尼系数等。

1.1.2 缓解过拟合:

- **剪枝 (Pruning)**：剪枝可以分为预剪枝（构建过程中停止增长）和后剪枝（构建后回溯修剪）。
- **设置最大深度**：限制树的最大深度，防止过深的树对训练数据过拟合。
- **最小样本数**：设置分裂节点所需的最小样本数，避免过度划分。
- **随机森林 (Random Forest)**：通过集成多个决策树并对其结果取平均，可以降低单棵树的过拟合风险。

1.2 多层感知机

1.2.1 工作原理:

- 多层感知机是一种前馈神经网络 (不存在同层/跨层连接)，包含至少一个隐藏层。每一层中的神经元通过权重和偏置与下一层的神经元相连，网络通过非线性激活函数（如 ReLU、Sigmoid）将输入映射到输出。
- 网络通过反向传播算法不断调整权重，以最小化损失函数的误差，从而实现分类或回归任务。

1.2.2 缓解过拟合:

- **早停 (early stopping)**:
  - 若训练误差连续 a 轮的变化小于 b, 则停止训练
  - 使用验证集：若训练误差降低、验证误差升高, 则停止训练
- **正则化 (regularization)**：在损失函数中加入惩罚项，描述网络复杂度。例如 $(1-\lambda)\sum_i w_i^2$
- **Dropout**：在训练过程中随机“丢弃”部分神经元，防止网络过度依赖某些节点，增强泛化能力。
- **数据增强 (Data Augmentation)**：通过对训练数据进行随机变换，增加数据的多样性，从而提升模型的泛化性能。

Task2 请使用 scikit-learn 实现决策树模型，并扰动决策树的最大深度 max_depth。一般来说，max_depth 的值越大，决策树越复杂，越容易过拟合，实验并比较测试集精度，讨论并分析观察到的过拟合现象等（5 points）；

2.1 实验过程和结果

我们选取了 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 18, 22, 26, 30] 作为 max_depth 的值，并发现在 max_depth 较大的时候，模型在不少数据集上出现了过拟合现象，全部图像见 Prob2/out/task2 文件夹中。我们选取一部分代表性的图像如下：

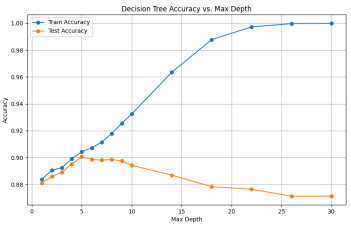


图 15: bank-full dataset

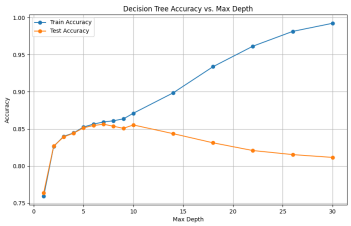


图 16: income dataset

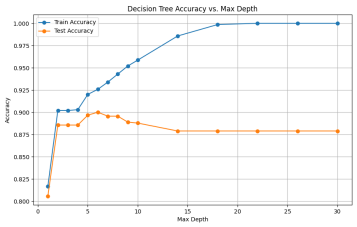


图 17: moon dataset

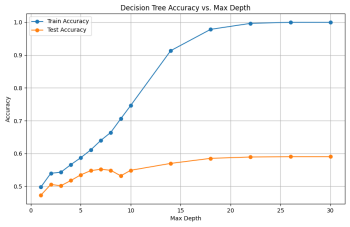


图 18: wine-quality dataset

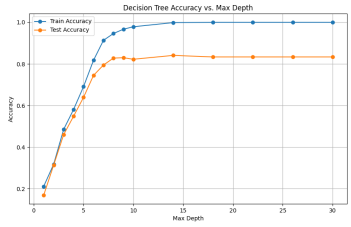


图 19: digits dataset

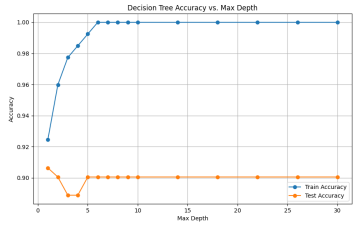


图 20: breast cancer dataset

2.2 解释

前三张图片较好的观察到了过拟合现象。而后三张图出现了一些奇怪的现象。个人猜测是图 18 可能是由于数据集的划分不好（因为我在之后跑的时候发现训练集中的某些特征很少无法 5 折交叉验证）；图 19 可能是 digits 数据集分布比较适合决策树或者是可能还未达到过拟合显示的最大深度（但是由于训练集上的正确率已经稳定达到 1.0 了所以第二种可能不大）；图 20 的出现一定程度上说明预剪枝的“贪心”有时候并不好。

Task 3 对决策树算法的未剪枝、预剪枝和后剪枝进行实验比较，并进行适当的统计显著性检验（5 points）

3.1 代码

预剪枝和后剪枝的代码见 Prob2/DTmodel.py 中，我们使用的 K 折交叉验证的方式进行实验比较，并选取综合得分最高的作为预剪枝/后剪枝的模型。

3.2 输出结果

全部结果见 Prob2/out/task3 文件夹中，我们选取了一部分代表性的结果如下：

```
# adult_income 40k Instances
未剪枝的决策树准确率: 0.811068116209078
预剪枝:
```

```
Best params: {'max_depth': 10, 'max_leaf_nodes': 28, 'min_samples_leaf': 1,'min_samples_split': 2}, Best score: 0.855010716612513
预剪枝的决策树准确率: 0.860573674835698
未剪枝 vs 预剪枝 p值: 1.2692101690951682e-78
后剪枝:
Best params: {'ccp_alpha': 0.00013663364519085}, Best score: 0.8599861513035167
后剪枝的决策树准确率: 0.862600577360113
未剪枝 vs 后剪枝 p值: 6.655682551121489e-23

# allwine 4.9K Instances 12 Features
未剪枝的决策树准确率: 0.5902564102564103
预剪枝:
Best params: {'max_depth': 2, 'max_leaf_nodes': 10, 'min_samples_leaf': 1,'min_samples_split': 2}, Best score: 0.5328845851618129
预剪枝的决策树准确率: 0.5056410256410256
未剪枝 vs 预剪枝 p值: 5.281555486312566e-10
后剪枝:
Best params: {'ccp_alpha': 0.007908052002760235}, Best score: 05328845851618129
后剪枝的决策树准确率: 0.5056410256410256
未剪枝 vs 后剪枝 p值: 5.281555486312566e-10

# cancer 569 Instances 30 Features
0.9271835443037976
预剪枝的决策树准确率: 0.9005847953216374
未剪枝 vs 预剪枝 p值: 0.8175564519377121
后剪枝:
Best params: {'ccp_alpha': 0.004745951982132888}, Best score: 09246518987341773
后剪枝的决策树准确率: 0.8947368421052632
未剪枝 vs 后剪枝 p值: 0.9083907372539946
```

Listing 2: 输出结果

3.3 个人理解

在较大的数据集上，剪枝的效果较为明显，而在较小的数据集上，剪枝的效果可能不太明显。

但是我们发现在 wine-quality 数据集上，预剪枝和后剪枝的结果并不理想，甚至比未剪枝的结果还要差，这可能是由于两种葡萄酒相互是对方的噪声、未经剪枝的决策树恰好具备足够的容量来拟合数据，而剪枝后容量不足，导致性能下降。

此外，在小数据集上，预剪枝和后剪枝的效果可能并不明显。

Task4 请使用 PyTorch 或 scikit-learn 实现一个简单的多层感知机，并通过层数、宽度或者优化轮数控制模型复杂度，实验并比较测试集精度，讨论并分析观察到的过拟合现象等（5 points）

我们选择了不同的层数、宽度、轮数的值，并发现在轮数、层数、宽度较大的时候，模型在不少数据集上出现了过拟合现象，全部图像见 Prob2/out/task4 文件夹中。我们选取一部分代表性的图像如下：

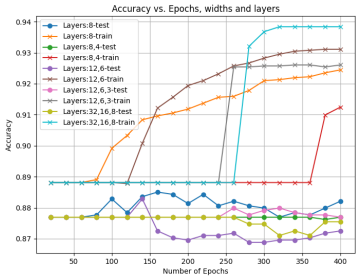


图 21: bank dataset

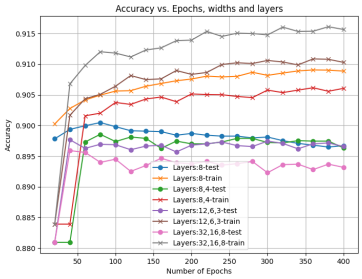


图 22: full-bank dataset

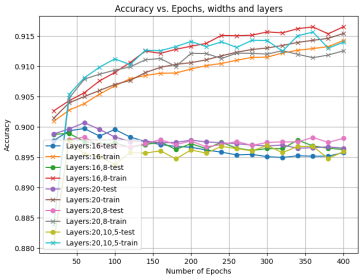


图 23: full-bank dataset

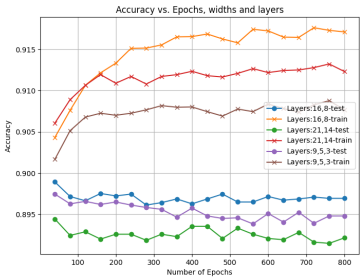


图 24: full-bank dataset

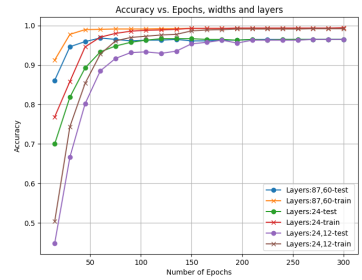


图 25: digits dataset

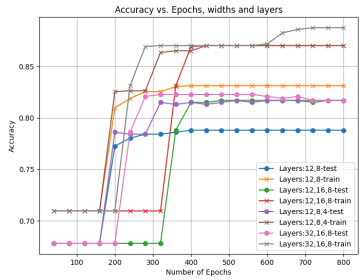


图 26: car-evaluation dataset

2.2 解释

图 21-23 可以较好地观察到了因为优化轮数过多、宽度过大、层数过多而导致得过拟合现象，以及轮数小、宽度较低、层数较少时的欠拟合现象。

图 24 可以观察到，宽度不是越大越好，层数也不是越多越好；

图 25 可以观察到增加轮数或宽度，可能会导致模型收敛所需的优化轮数变长；

图 26 可以观察到因为轮数小、宽度较低、层数较少时的欠拟合现象。

2.3 分析

- 层数（Depth）：
  - 过多的层数可能导致模型过于复杂，容易记忆训练数据中的噪声，从而引发过拟合，导致在测试集上的性能下降。
  - 过少的层数可能有欠拟合的风险，但是在我们的数据集出现得不是很明显。
- 宽度（Width）：
  - 过宽的网络同样会增加模型的复杂度，增加过拟合的可能性，尤其是在训练数据量不足时.
  - 过窄的网络可能限制模型的表达能力，导致无法充分学习数据中的有用特征。
- 优化轮数：
  - 训练过多的轮数容易使模型过度拟合训练数据，损害在未见数据上的泛化能力。
  - 训练轮数过少，模型可能未能充分学习数据特征，导致欠拟合

注：数据集选择

数据集的加载见 Prob2 中的 dataloader.py 文件, 数据集本身见 data 文件夹。主要选择了 adult income, car evaluation, bank marking, dights,wine quality, breast cancer, moon 等数据集。

三. (20 points) 激活函数比较

神经网络隐层使用的激活函数一般与输出层不同. 请画出以下几种常见的隐层激活函数的示意图并计算其导数（导数不存在时可指明）, 讨论其优劣（每小题 4 points）：

图像见该题目最后

1. Sigmoid 函数, 定义如下

$$f(x) = \frac{1}{1+e^{-x}}. \tag{1}$$

- 导数 $f'(x) = f(x)(1-f(x))$.
- 优缺点
  - 优点：连续且平滑，输出值范围在 (0, 1) 之间，容易求导。
  - 缺点：在极大值或极小值处梯度非常小，容易引发梯度消失问题，训练深层网络时表现不佳。

2. 双曲正切函数 (Hyperbolic Tangent Function, Tanh), 定义如下

$$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}. \tag{2}$$

- 导数 $f'(x) = 1 - f(x)^2$.
- 优缺点
  - 优点：输出范围为 (-1, 1)，相比于 Sigmoid 函数居中于零，表现更好，避免了 Sigmoid 的零中心问题。
  - 缺点：在极大值或极小值处梯度非常小，容易引发梯度消失问题，训练深层网络时表现不佳。

3. 修正线性函数 (Rectified Linear Unit, ReLU) 是近年来最为常用的隐层激活函数之一, 其定义如下

$$f(x) = \begin{cases} 0, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases} \tag{3}$$

- 导数 $f'(x) = \begin{cases} 0, & \text{if } x < 0; \\ 1, & \text{if } x > 0; \\ \text{undefined}, & \text{if } x = 0. \end{cases}$
- 优缺点
  - 优点：计算简单，在正值区域，m 导数为常数加速了训练过程，且在正向传播时避免了梯度消失问题，使得深层网络的训练更加高效。

  - 缺点：存在梯度未定义的情况；且在负值区域梯度为 0，导致神经元“死亡”，即当神经元在负值区域时，它的权重不再更新，可能导致模型不够灵活。

4. Softplus 函数, 定义如下

$$f(x) = \ln(1 + e^x). \tag{4}$$

   - 导数 $f'(x) = \frac{1}{1+e^{-x}} = Sigmoid(x)$.
   - 优缺点
     - 优点：和 ReLU 梯度相似，但其梯度任意点有定义；且在负值区域具有非零的梯度，可以避免 ReLU 的“死亡神经元”问题。
     - 缺点：相比 ReLU 计算代价较高。

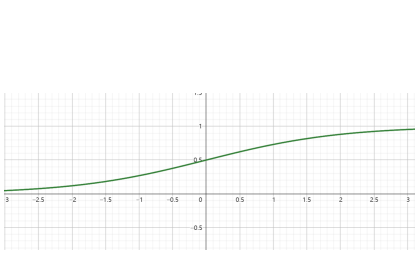
5. Leaky Rectified Linear Unit (Leaky ReLU) 函数, 定义如下

$$f(x) = \begin{cases} \alpha x, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases} \tag{5}$$

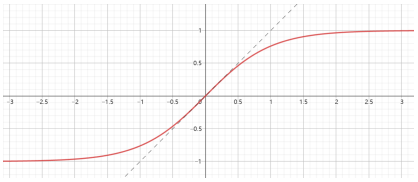
   其中 α 为一个较小的正数, 比如 0.01.

   - 导数 $f'(x) = \begin{cases} \alpha, & \text{if } x < 0; \\ 1, & \text{if } x > 0; \\ \text{undefined}, & \text{if } x = 0. \end{cases}$
   - 优缺点
     - 优点：和 ReLU 梯度相似，且计算效率高，在负值区域具有非零的梯度，可以避免 ReLU 的“死亡神经元”问题。
     - 缺点：负值区域的梯度仍然非常小，因此负值区域的学习速度较慢；且仍有未定义的梯度。

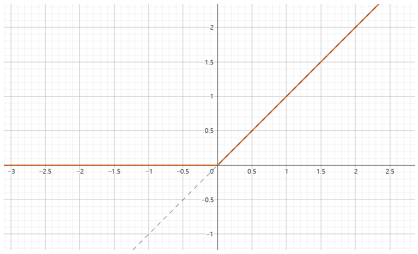
函数图像如下：



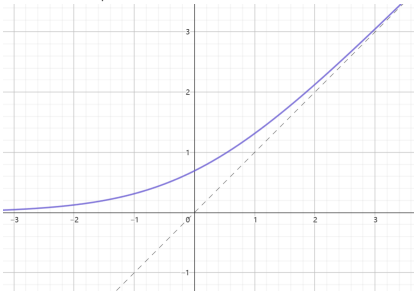
(a) $f(x) = \frac{1}{1+e^{-x}}$



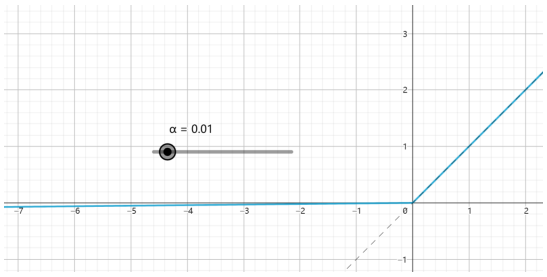
(b) $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(c) $f(x) = \ln(1 + e^x)$



(d) $f(x) = \begin{cases} \alpha x, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases}$



(e) $f(x) = \begin{cases} 0, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases}$

四. (30 points) 神经网络实战

moons 是一个简单的二分类数据集，请实现一个简单的全连接神经网络，并参考教材图 5.8 实现反向传播算法训练该网络，用于解决二分类问题。

1. 使用 NumPy 手动实现神经网络和反向传播算法。（15 points）
2. 实现并比较不同的权重初始化方法。（5 points）
3. 在提供的 moons 数据集上训练网络，观察并分析收敛情况和训练过程。（10 points）

提示：

1. 神经网络实现：
   - 实现一个具有一个隐藏层的全连接神经网络。
   - 网络结构：输入层 (2 节点) → 隐藏层 (4 节点) → 输出层 (1 节点)
   - 隐藏层使用 ReLU 激活函数，输出层使用 Sigmoid 激活函数。
   - 使用交叉熵损失函数。
2. 权重初始化方法。实现以下三种初始化方法，并比较它们的性能：
   - 随机初始化：从均匀分布 $U(-0.5, 0.5)$ 中采样。

- Xavier 初始化：根据前一层的节点数进行缩放。

$$W_{ij} \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$$

其中 n_{in} 是前一层的节点数，n_{out} 是当前层的节点数。

- He 初始化：He 初始化假设每一层都是线性的，并且考虑了 ReLU 激活函数的特性。

$$W_{ij} \sim N\left(0, \frac{2}{n_{in}}\right)$$

其中 n_{in} 是前一层的节点数。

3. 训练和分析：

- 使用提供的 moons 数据集。
- 实现小批量梯度下降算法。
- 记录并绘制训练过程中的损失值和准确率，训练结束后绘制决策边界。
- 比较不同初始化方法对训练过程和最终性能的影响并给出合理解释。
- 尝试不同的学习率，观察其对训练的影响并给出合理解释。

解:

4.1 代码

- 代码见压缩包 code 中的 Prob4, 详细说明见 Prob4 中的 README 文件;
- 详细输出见 Prob4 中的 out 文件夹。

4.2 记录并绘制训练过程中的损失值和准确率，训练结束后绘制决策边界

实验结果在 Prob4/out/out.log 与 info.log 中，绘制为表格如下：

表 4: 不同 lr,epoch,mehod 的 MLP 模型在测试集的 ACC

方法 / lr,epoch	random	xavier	he
0.01,500	0.87	0.87	1.0
0.01,200	0.87	0.87	0.965
0.1,100	0.865	0.865	1.0
0.1,40	0.88	0.88	0.995
1, 50	0.86	0.865	1.0
1, 20	0.875	0.895	1.0

所有图片见 Prob4/out 文件夹中，我们选取了一部分代表性的图片如下：

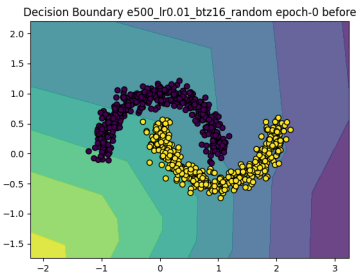


图 28: e500 lr0.01 rand init

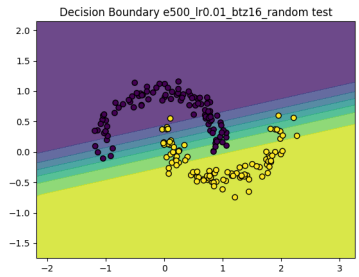


图 29: e500 lr0.01 rand test

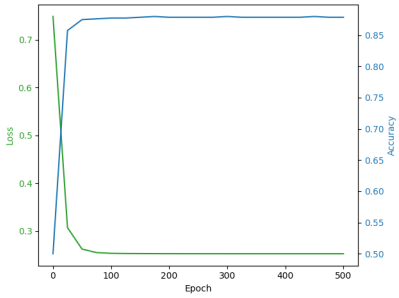


图 30: e500 lr0.01 rand train

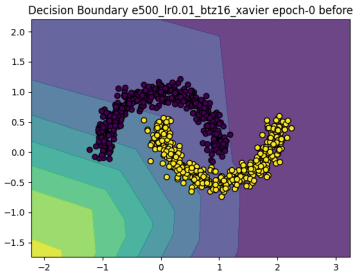


图 31: e500 lr0.01 xavier init

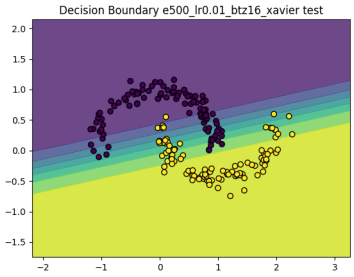


图 32: e500 lr0.01 xavier test

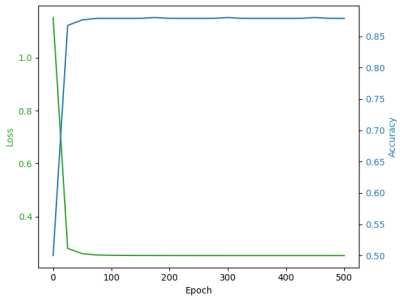


图 33: e500 lr0.01 xavier train

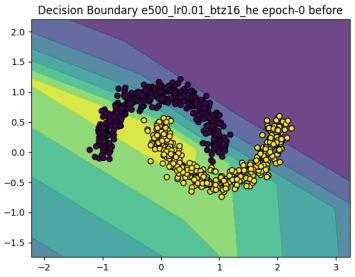


图 34: e500 lr0.01 he init

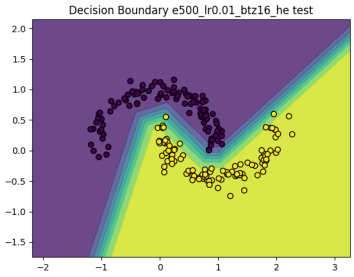


图 35: e500 lr0.01 he test

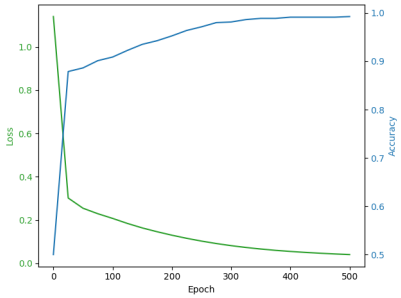


图 36: e500 lr0.01 he train

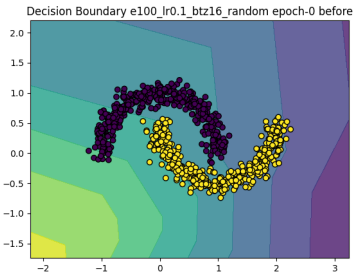


图 37: e100 lr0.1 rand init

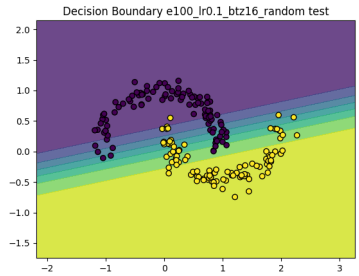


图 38: e100 lr0.1 rand test

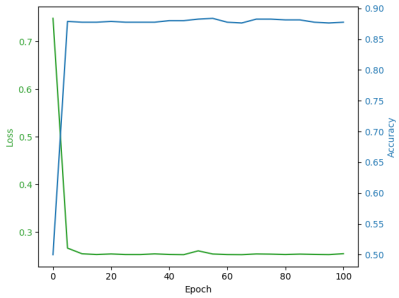


图 39: e100 lr0.1 rand train

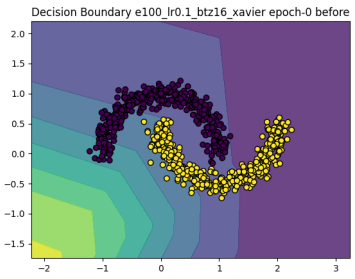


图 40: e100 lr0.1 xavier init

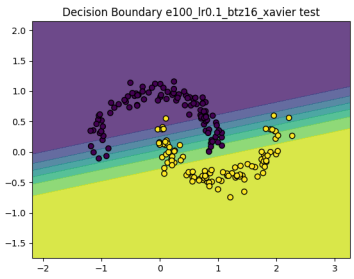


图 41: e100 lr0.1 xavier test

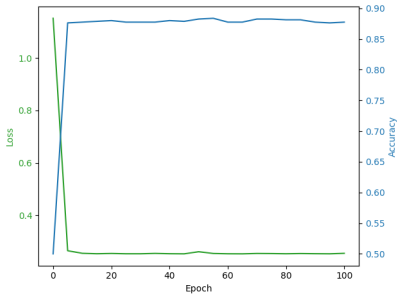


图 42: e100 lr0.1 xavier train

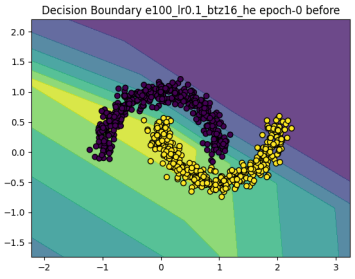


图 43: e100 lr0.1 he init

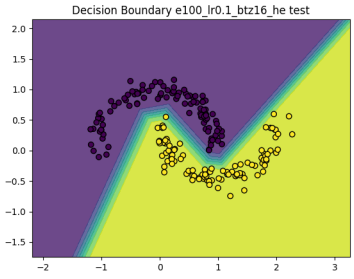


图 44: e100 lr0.1 he test

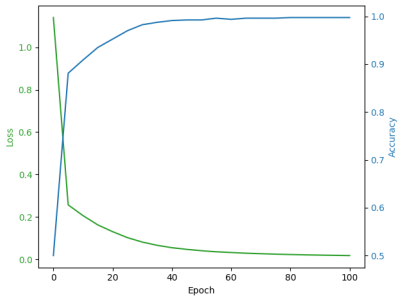


图 45: e100 lr0.1 he train

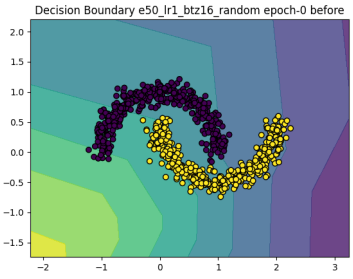


图 46: e50 lr1.0 rand init

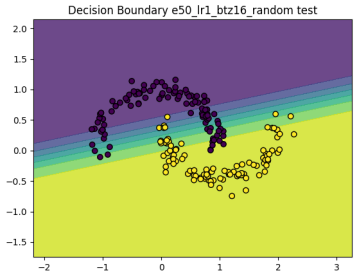


图 47: e50 lr1.0 rand test

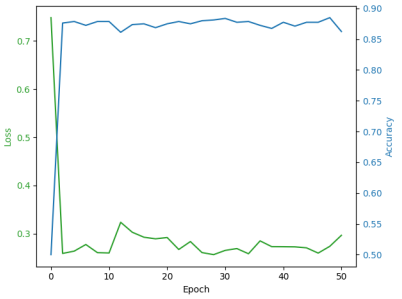


图 48: e50 lr1.0 rand train

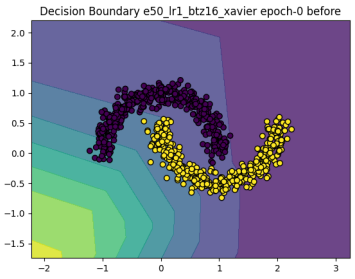


图 49: e50 lr1.0 xavier init

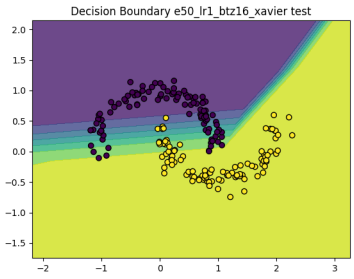


图 50: e50 lr1.0 xavier test

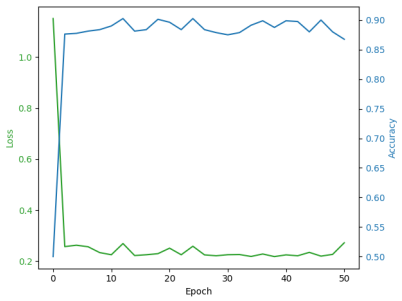


图 51: e50 lr1.0 xavier train

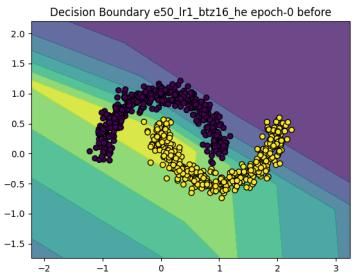


图 52: e50 lr1.0 he init

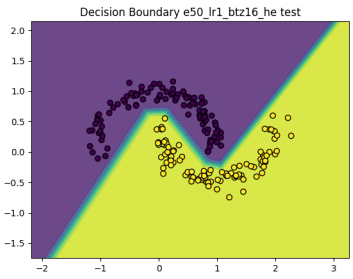


图 53: e50 lr1.0 he test

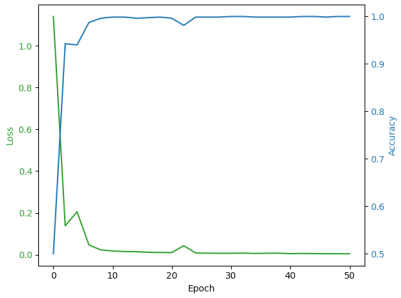


图 54: e50 lr1.0 he train

4.3 比较不同初始化方法对训练过程和最终性能的影响并给出合理解释

4.3.1 结果

在相同的学习率和训练轮数下，比较了随机初始化、Xavier 初始化和 He 初始化三种不同的权重初始化方法对神经网络模型的训练过程和最终性能的影响。通过实验结果的对比，可以得出以下结论：

在实验环境下，He 初始化方法在 moons 数据集上表现最好，其次是 Xavier 初始化，随机初始化的性能最差。

在学习率小于 1 的情况下，随机初始化和 xavier 初始化的收敛速度比 He 初始化快，但是最终在测试集的表现不如 He 初始化。

4.3.2 补充资料

在查阅相关资料：

- 随机初始化是最基本的方法，适用于浅层网络或作为其他初始化方法的基准。然而，由于没有考虑激活函数和网络层数对输出方差的影响，可能导致信号在传播过程中逐层衰减或爆炸，可能会出现方差不匹配的问题；同时在深层网络中，随机初始化可能导致梯度消失或爆炸，进而影响模型的训练效果和收敛速度。
- Xavier 初始化通过精心设计的方差保持机制，适用于对称激活函数，如 sigmoid 和 tanh。它在深层网络中表现良好，能有效缓解梯度消失和爆炸的问题，促进稳定的训练过程和良好的最终性能。但对于 ReLU 等非对称激活函数，Xavier 初始化没有充分考虑 ReLU 的特性，可能导致前向传播中部分神经元处于非激活状态，从而影响梯度的传播。
- He 初始化进一步优化了 Xavier 初始化，专为 ReLU 激活函数设计。
  - 更适合 ReLU：ReLU 激活函数在正区间线性增长，而在负区间为零，这种非对称性需要更高的初始化方差来保持信号的传播。He 初始化通过增加方差，确保在 ReLU 激活后信号不至于过于衰减。
  - 有效保持信号传递：在深层网络中，He 初始化能够更有效地保持前向传播和反向传播的信号强度，减少梯度消失的问题。提升训练效率：由于更好的信号传播，He 初始化通常能加快模型的收敛速度，提升最终的模型性能。
-

4.3.3 个人解释

实现的 MLP 中，虽然 Loss 层使用的是 sigmoid 函数，但是隐藏层使用的是 ReLU 函数，因此 He 初始化在这种情况下表现最好。而由于 Xavier 初始化没有考虑 ReLU 的特性，但是适合 sigmoid 对称激活函数，因此在这种情况下表现次之。随机初始化则表现最差。

4.4 尝试不同的学习率，观察其对训练的影响并给出合理解释。

在训练神经网络模型时，学习率（Learning Rate）是一个关键的超参数，决定了优化算法在参数空间中移动的步伐大小。选择合适的学习率对于模型的收敛速度和最终性能具有重要影响。

为了研究学习率对训练的影响，选择了 0.01, 0.1, 1.0 这几种不同的学习率值进行实验.

4.4.1 结果

- **学习率较低（0.01）**：导致模型收敛速度较慢，训练效率低下，需要更多的训练轮数 (这里是 500 轮) 才能达到较好的性能。
- **学习率适中（0.1）**：在保证训练稳定性的同时，加快了模型的收敛速度，提高了训练效率。
- **学习率过高（1）**：在上图中，明显发现 lr=1.0 时，在 Xavier 初始化和随机初始化的条件下，引发训练过程的不稳定，损失函数和训练集上的正确率出现剧烈波动，导致模型无法收敛。

4.4.2 合理解释

学习率决定了每次参数更新的步长大小。当学习率过低时，优化过程需要更多的迭代次数才能接近最优解，训练效率低。而学习率过高时，优化过程可能在最优解附近来回跳跃，可能无法稳定收敛，甚至导致参数发散，影响模型性能。