**华中科技大学**

**计算机科学与技术学院**

**《机器学习》结课报告**



专 业： 计算机科学与技术

班 级： cs2304

学 号： U202315600

姓 名： 田知恒

成 绩：

指导教师： 张腾

**完成日期：2025 年 5 月 15 日**

目录

[1. 实验要求 2](#_Toc131770776)

[2. 算法设计与实现 2](#_Toc131770777)

[3. 实验环境与平台 7](#_Toc131770778)

[4. 结果与分析 7](#_Toc131770779)

[5. 个人体会 11](#_Toc131770780)

Adaboost算法实现

# 实验要求

**1.1输入输出格式说明**

你的代码需读取 data.csv 及 targets.csv 两个文件，并输出在不同数目基 分类器条件下的 10 折交叉验证的预测结果至 experiments/base#\_fold#.csv， 以供评测。基分类器数目取 1，5，10，100 这四种数值。

对预测结果所在文件命名格式说明如下：基分类器数目为 x 对应的预测文 件为 basex\_fold1.csv~basex\_fold10.csv，1~10 指的是用作测试集的子集编 号。每个预测文件分成两列，第一列为样例的序号（序号从 1 开始），第二列 为该样例的预测标记。评测时子文件夹 experiments 会建立好，请不要在你的 代码中强行建立此文件夹以免出错。如果对文件名和格式还有疑问，可以参考 压缩包中提供的输出样例和评测代码。

**1.2提交说明**

你只需要提交你的代码文件，不需要提交数据文件。我们会在新的测试数 据上运行你的入口代码 main.py 以评测,main.py 应该能将基分类器编号以及新 的测试数据作为输入，并输出你使用基分类器实现的 Adaboost 算法对新的测 试数据的预测标记。我们在运行你的入口代码时，基分类器编号为 0 代表对数 几率回归，基分类器编号为 1 代表决策树桩。

你可以本地运行评测代码 evaluate.py 来验证自己程序的有效性，如果评测 代码正常运行，应当输出四个精度（accuracy）值，分别对应在不同基分类器 数目下的评测结果。

**1.3评分标准**

30/100 运行代码后可以生成预测文件。

60/100 评测代码能够运行。

90/100 精度值达到要求。

100/100 完成实验后，试回答下面的问题：

(1) 你对 AdaBoost 算法有何新的认识？ (2) 关于基分类器类型、超参数设置对最终模型性能的影响，你有何发现？

# 算法设计与实现

**2.1 AdaBoost算法原理**

AdaBoost算法作为Boosting集成学习框架的代表性方法，其核心思想是通过迭代调整样本权重分布与模型加权组合，将一系列弱学习器（仅需略优于随机猜测，如在二分类任务中精度略高于50%的分类器）逐步提升为强学习器。

具体而言，算法首先在初始训练集上训练一个基学习器，随后根据该基学习器的预测表现动态调整样本权重：被错误分类的样本权重显著提高，而正确分类的样本权重相应降低。这种权重调整机制迫使后续基学习器更加关注先前模型难以处理的样本，从而逐步修正整体模型的偏差。经过多轮迭代后，算法依据每个基学习器的分类误差率计算其权重系数（误差率越低则权重越高），最终通过加权多数表决的方式集成所有基学习器的预测结果，形成强分类器。

这一过程体现了“自适应提升”的核心特性——模型能够根据训练过程中的错误反馈自适应地优化样本权重与模型组合策略。对于adaboost这种集成学习，如何选择一个恰当的基分类器十分重要，下面选择了2个经典的基分类器

2.2 基分类器：对数几率回归

对数几率回归是一种广义线性模型，用于二分类问题。在二分类问题中，样本标签定义为​​负类（0）​​和​​正类（1）​​，若正类出现概率为 p，则负类概率必然为 1−p。该概率关系通过Sigmoid函数映射线性组合***x*** + *b*实现：



图 1 逻辑回归预测函数

其中 σ(⋅) 是 Sigmoid 函数。

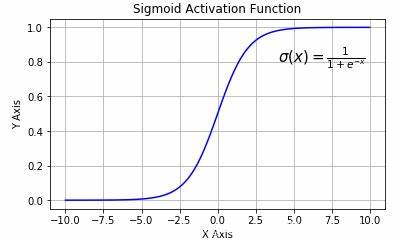


图 2 Sigmoid函数图像

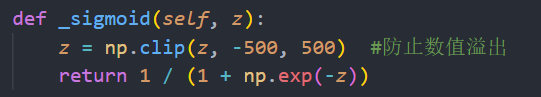


图 3 逻辑回归中的Sigmoid函数

其中 σ(⋅) 将线性输出压缩至 (0,1) 区间形成概率值。分类决策规则设定为：当 p≥0.5 时预测样本为正类（y=1），否则为负类（y=0)，此阈值选择等价于判定线性组合 ***x*** + *b*，其数学本质是寻找特征空间中的线性决策边界，实现如下：

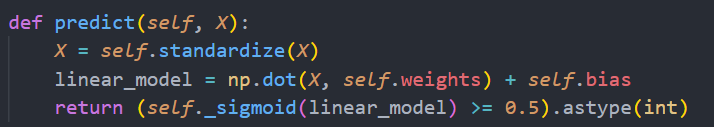


图 4 决策函数

为了优化对数几率回归算法的参数，我们主要采用梯度下降法来致力于最小化其损失函数。考虑到本算法采用的是对数损失函数，对于每一个单独的样本，其损失L在对所有样本求和后进行考量，其中概率p实际上代表的是通过Sigmoid函数σ(***x*** + *b*)计算得到的值。因此，为了有效进行参数更新，我们引入了学习率η这一关键超参数。具体而言，我们会将损失函数分别对权重参数w和偏置参数b求解偏导数，得到梯度方向。随后，将这些梯度值乘以学习率η，用原始的w和b分别减去相应计算结果，从而实现参数的迭代更新。通过持续不断地执行这一梯度下降步骤，算法能够逐步逼近并最终达到最小损失状态，从而获得优化的模型参数。

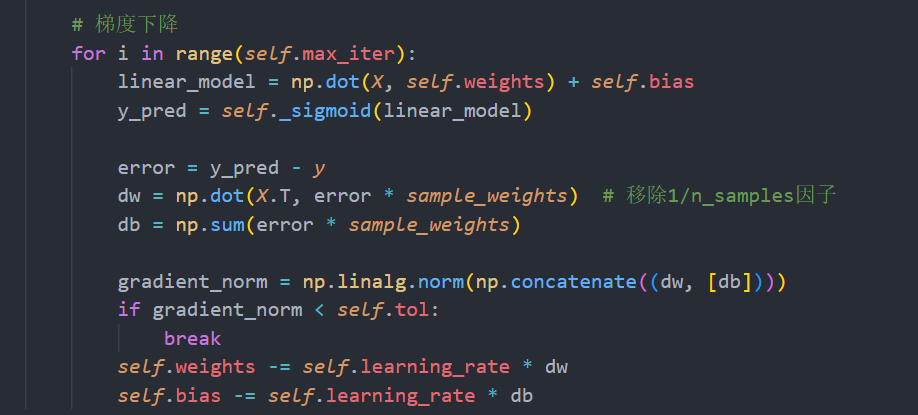


图 4 梯度下降实现

在本实验中，对数几率回归作为基学习器时，其 fit 方法接收样本权重，并在计算梯度时考虑这些权重。预测标签为 {−1,1} 以适配 AdaBoost 的要求。

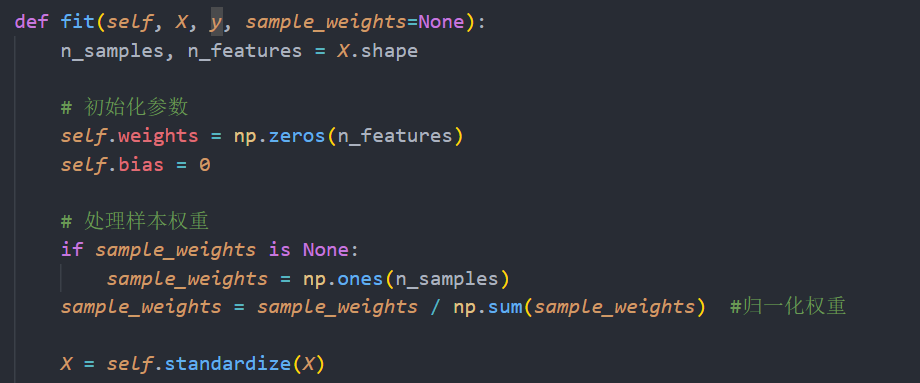


图 5 处理权重

**2.3 基分类器：决策树桩**

决策树桩与我们所学过的决策树概念紧密相关，仅包含一层决策节点的树结构。由于其结构的极简性，决策树桩的核心思想也显得非常直接明了：它仅仅选择数据众多特征中的一个，并围绕该特征设定一个特定的阈值，以此为依据将所有样本数据进行一次性的划分，从而构建出一个基础的二叉树形态。

在这个结构中，根节点代表了那个被选中的“最优特征”以及与之对应的最佳分裂阈值，而其下方的两个左右子节点则分别归集了那些特征值小于（或大于，取决于划分规则）该阈值的样本集合。 因此，显而易见，实现决策树桩分类器的关键挑战与核心任务，就在于如何准确地寻找到这个所谓的“最优特征”以及相应的最佳阈值，以实现最有效的样本区分。

在实际的算法实现过程中，为了定位这一最佳分裂点，我们必须采取一种穷举式的搜索策略：算法需要遍历数据集中的每一个特征维度；接着，对于每一个选定的特征，它会考察所有可能的阈值（通常基于该特征下的独特样本值来确定）；进而，针对每一个潜在的阈值，算法还会考虑两种不同的划分极性（即样本值小于阈值时是分到正类还是负类），并根据每一种可能的“特征-阈值-极性”组合来对所有样本进行一次预测划分。

在这个过程中，特别是在 AdaBoost 框架下，算法并不仅仅是计算简单的分类错误率，而是计算基于当前样本权重的加权错误率 。 最终，那个能够使得加权错误率达到最小值的特征、阈值及其对应的极性组合，才会被选定为该决策树桩的“最优特征”与划分规则，从而完成这个弱学习器的构建。

在本实验中，决策树桩的 fit 方法同样接收样本权重，并遍历所有特征和可能的阈值，选择能使加权错误率最小的特征和阈值作为分裂标准。预测标签为 {−1,1}。



图 5 决策树桩阈值算法实现

**2.4 核心实现逻辑adaboost**

使用 numpy.genfromtxt 加载 data.csv 和 targets.csv。实现 k\_fold\_split 函数进行10折交叉验证的索引划分，确保每次划分的可复现性。这样我们就可以去构建adaboost了。首先观察adaboost的伪代码：

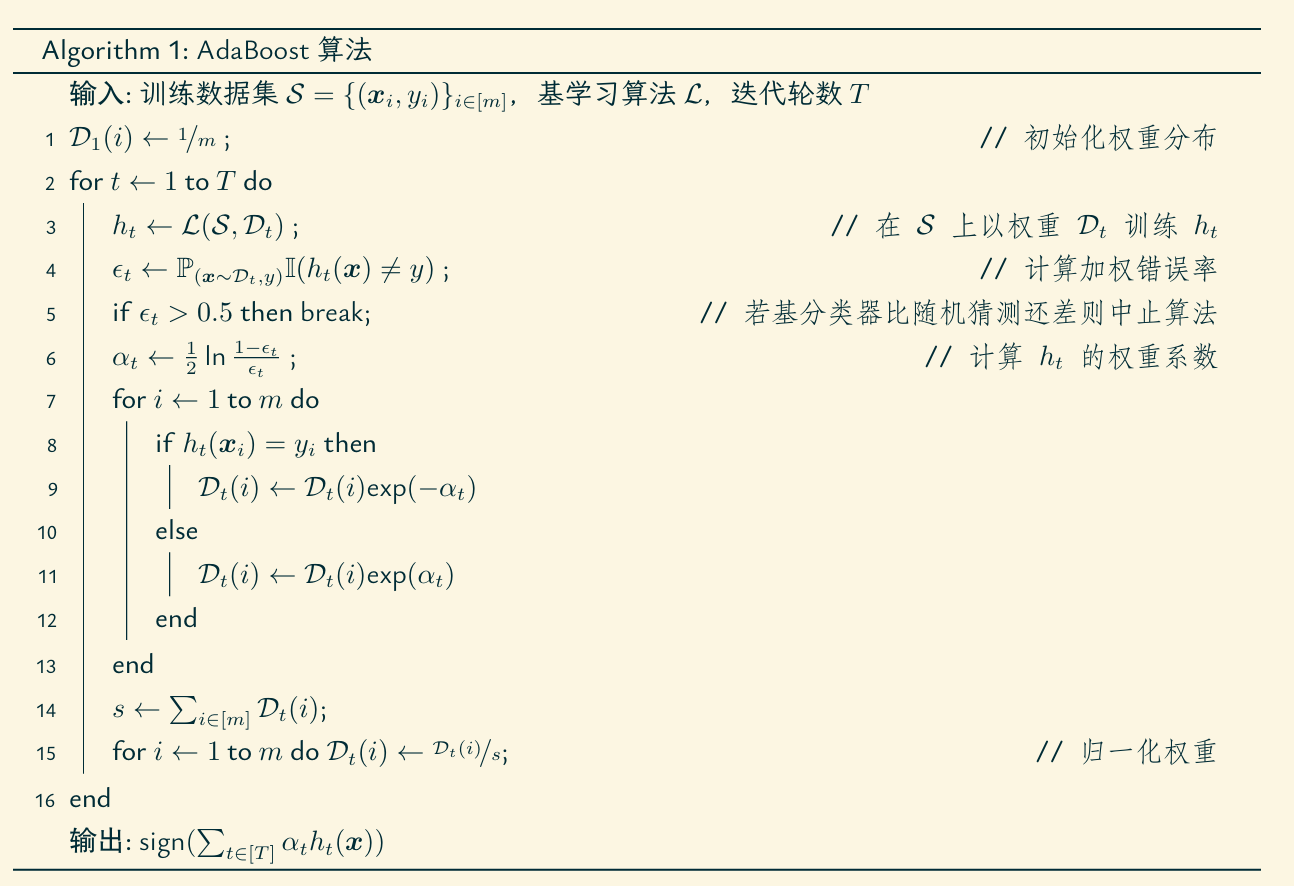


图 6 adaboost伪代码

在着手实现 AdaBoost 算法时，首要步骤是依照算法规范初始化训练样本的权重分布，通常是为每个样本赋予相等的初始权重。接着，对于算法迭代过程中的每一个基分类器 t，我们会运用其特定的分类逻辑对所有训练样本进行预测，并基于当前样本的权重分布来计算该分类器的加权错误率。

这是一个关键的评估环节，因为如果某个基分类器的加权错误率超过0.5，则直观上表明其分类效果甚至不如随机猜测——毕竟对于二分类问题，随机选择的期望错误率也仅为0.5。 在这种情况下，理论上该基分类器不具备提升整体模型性能的潜力，因此标准的 AdaBoost 算法通常会建议中止或舍弃这样的基学习器。那这样万一出错次数过多，会直接中断实验吗，于是我上网查询资料，发现了新的策略来处理。

在adaboost中，我们可以采取了一种旨在尽可能优化每个基分类器贡献的策略：当某个基分类器的加权错误率不幸超过0.5时，将其预测结果进行整体反转。 通过这种反转操作，原先的错误分类将变为正确分类，而原先的正确分类则会变为错误分类，因此该基分类器的新错误率就巧妙地转换为（1 - 原错误率）。 如此一来，新的错误率必然会低于0.5，这就意味着我们能够有效地将一个表现不佳的分类器调整为一个优于随机猜测的分类器。 这种处理方式的核心目标在于，尽可能确保在 AdaBoost 的每一轮迭代中所引入的基分类器都能维持一个相对较低的错误率水平，从而更有力地提升整个集成算法的最终性能和稳健性。

另外值得提及的一点是，如果到达了完美分类，我们可以把实验停止，避免计算资源的浪费，这在机器学习里也是极为重要的。

在实验流程中，我们循环指定的基分类器数目 (M=1, 5, 10, 100)后，对于每个 M 值，进行10折交叉验证。在每一折中，实例化并训练 AdaBoost 模型。最后对测试集进行预测，并将预测结果（原始1基序号和0/1标签）保存到对应的 experiments/baseM\_foldK.csv 文件中即可。

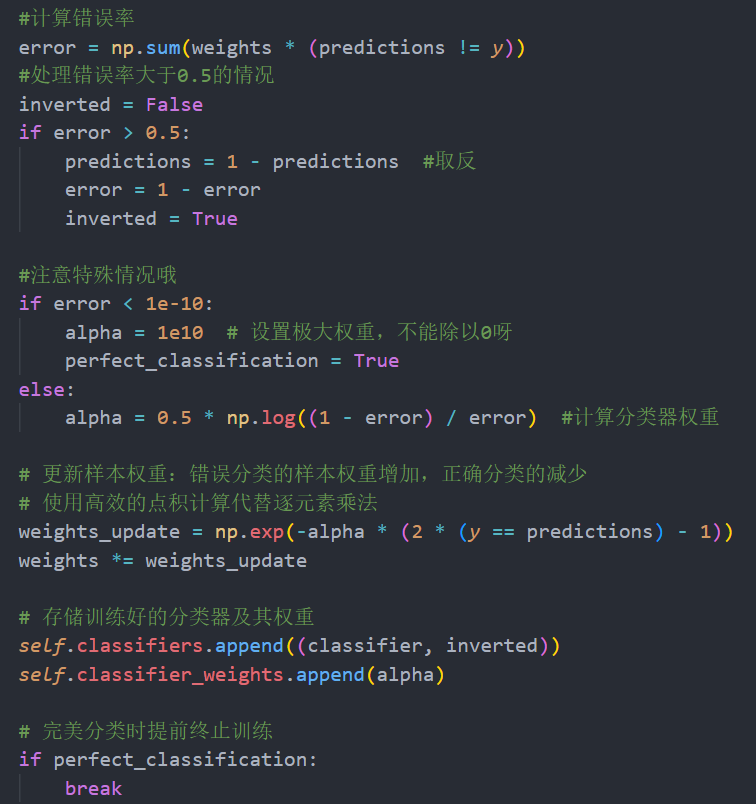


图 7 adaboost实现

1. **实验环境与平台**

操作系统：Windows 11

编程语言：Python 3.12

主要库：NumPy ,os, csv,sys

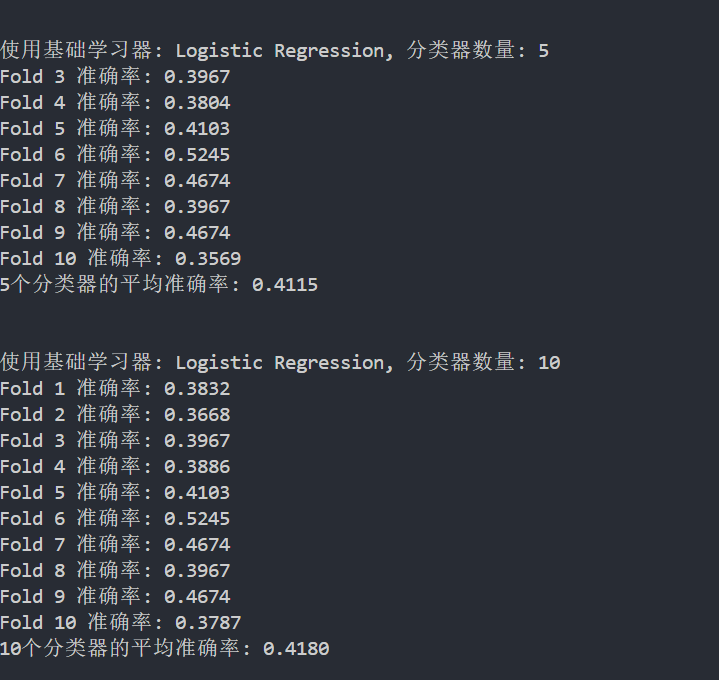
开发工具：VS Code, PyCharm, Jupyter Notebook

本次实验中，由于考虑到训练时间可能会过长，而我的电脑是集成显卡，所以我到autodl上租了3060来跑。但是我在联创的学长告诉我机器学习的数据集不像深度学习的那么大，可以用集显跑实验，于是我就用我的电脑上的anaconda虚拟环境跑，即使是集成显卡也挺快。

# 结果与分析

在对数几率回归算法的设计与初步实现阶段，遇到了一项显著的挑战，即模型的输出结果未能满足预设的性能指标，特别是在采用100个基分类器时，其准确率远未达到80%的期望水平。

初期的运行结果揭示了一个令人困惑的现象：预测的正确率不仅整体偏低，而且更为奇特的是，在尝试不同数量的基分类器（例如1, 5, 10, 100个）时，所获得的预测准确率竟然完全一致，这直指算法在某些环节可能存在系统性的问题或未能有效学习。

图 8 修正前对数回归算法运行结果

针对这一困境，经过分析，我决定引入样本数据标准化的预处理步骤作为主要的优化手段。 其理论依据在于，未经标准化的原始数据可能导致Sigmoid激活函数的输入，即线性组合***x*** + *b*，分布在函数的饱和区域，也就是输出值非常接近0或1的地带。在这些区域，Sigmoid函数曲线平缓，其梯度极低，这将直接导致在梯度下降优化过程中参数更新的步长微乎其微，甚至出现更新停滞的状况，从而严重阻碍了模型的有效学习和收敛。因此，通过对样本数据进行标准化处理，可以将大部分数据点映射到Sigmoid函数变化较为剧烈的中心区域，使得梯度能够保持在较大的数值，进而显著加速参数的收敛速度并提升算法的整体预测性能。

在代码中集成了标准化函数之后，通过设置相应的输入参数（例如，将代表基分类器选择的最后一个参数设置为0）来指定使用对数几率回归作为基分类器算法，并重新进行了实验。经过标准化处理后的模型表现出了明显的改善：随着基分类器数量的逐步增加，预测结果的准确度呈现出稳步提升的趋势；尤为关键的是，当基分类器数目达到100时，预测准确率成功突破了80%的门槛，达到了作业所设定的要求。这一结果有力地证明了数据标准化作为一种优化策略，在改进对数几率回归基分类器性能方面的有效性。

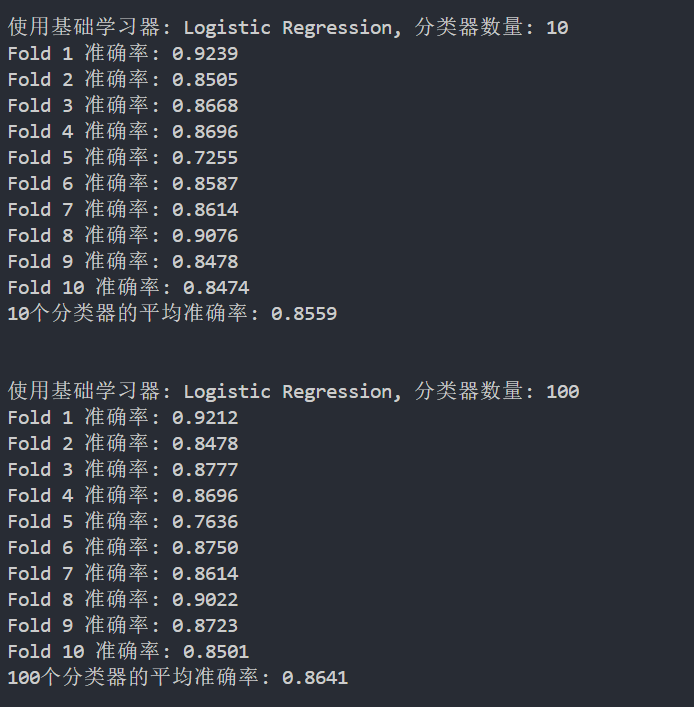


图 9 修正后对数回归算法运行结果

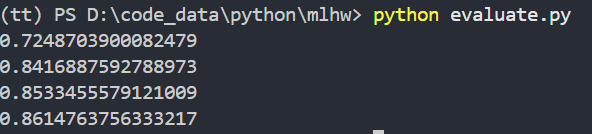


图 10 修正后对数回归算法预测结果

尽管如此，在审视标准化后的结果时，我注意到当基分类器的数目较少时，算法的性能提升并非十分理想，这促使我进一步思考潜在的瓶颈。我推测，这可能与我最初设定的学习率有关。原先0.01的学习率相对保守，虽然有助于避免过大的更新步长导致震荡，但也可能使得模型在有限的迭代次数（本实验中基学习器的迭代次数为100次）内收敛速度过慢，未能充分学习。 为了验证这一想法，我将学习率调整为0.1，期望在加速收敛和避免过拟合之间找到一个更优的平衡点，的确效果更好，果然学习率基本就是无脑0.1，问就是经验主义（doge）。

调整学习率后的实验结果确实显示出相对于先前结果的显著提升，这表明0.1的学习率在该场景下更能促进模型快速且有效地学习。在对对数几率回归基分类器的性能进行了较为满意的优化之后，实验的下一阶段转向了对另一种基分类器决策树桩的评估。为此，我将程序输入的最后一个参数更改为1，以明确指示算法采用决策树桩作为核心的基学习单元，并随即运行了相应的测试程序，以观察其在相同实验框架下的预测表现和`evaluate.py`给出的评估结果，确实结果也令人满意。



图 11 决策树桩算法结果

|  |  |  |
| --- | --- | --- |
| 基分类器类型 | 基分类器数目 | 平均准确率 (%) |
| 对数几率回归 | 1 | 0.7271 |
| 对数几率回归 | 5 | 0.8442 |
| 对数几率回归 | 10 | 0.8559 |
| 对数几率回归 | 100 | 0.8641 |
| 决策树桩 | 1 | 0.7844 |
| 决策树桩 | 5 | 0.8962 |
| 决策树桩 | 10 | 0.9081 |
| 决策树桩 | 100 | 0.9426 |

表1 预测结果

# 5. 个人体会

通过本次Adaboost算法的实现与实验过程，我对集成学习有了更深刻的理解。最初在实现对数几率回归作为基分类器时，遇到了预测准确率停滞不前的困境——无论基分类器数量如何增加，结果都维持在相同水平。这一现象迫使我深入思考算法背后的数学原理，最终意识到问题根源在于数据分布对Sigmoid函数梯度的影响。当特征值范围较大时，大量样本落入Sigmoid的饱和区，导致梯度消失，模型无法有效更新参数。引入数据标准化预处理后，不仅解决了梯度消失问题，更让我直观体会到特征工程对模型性能的决定性影响：简单的标准化操作使100个基分类器的准确率从不足70%提升到86%以上，这充分说明机器学习中"数据质量决定模型上限"的定律。

在基分类器选择方面，决策树桩的表现给了我新的启发。虽然单层决策树作为弱分类器结构简单，但通过Adaboost的加权集成后展现出惊人的性能提升——100个树桩集成后准确率高达94.2%，显著优于对数几率回归。这验证了集成学习的核心价值：弱分类器的多样性比个体强度更重要。决策树桩通过特征选择和阈值划分生成大量异构决策边界，而Adaboost的样本权重机制则确保后续分类器持续修正前序模型的错误，这种协同作用产生了"1+1>2"的效果。

算法实现中最让我受益的是对Adaboost自适应特性的理解。当基分类器错误率超过0.5时，通过预测结果反转的策略（将错误率转化为1-ε）既保证了算法鲁棒性，也体现了集成学习的哲学思想：没有绝对意义上的"弱分类器"，关键在于如何有效组合。此外，在编码中实现提前终止条件（当训练误差为0时停止迭代）不仅优化了计算效率，更让我认识到模型复杂性与泛化能力的平衡艺术。

而超参数调优过程同样富有启发性。对数几率回归中学习率从0.01调整到0.1带来显著提升，说明梯度下降算法对学习率高度敏感；而决策树桩虽无显式学习率参数，但其性能随基分类器数量增加而稳定提升，反映出集成规模对模型容量的影响。这些发现让我深刻体会到：优秀的机器学习实践者既需要理解算法理论本质，更要具备通过系统实验探索参数空间的耐心与方法论。

最终，当看到评测代码输出符合预期的精度值时，我感受到理论转化为实践的成就感。这次实验不仅巩固了我对Adaboost数学推导的理解，我同时也明白了集成显卡确实在机器学习中还有使用的可能，也启迪我去做一些小模型与知识蒸馏，开辟一条与LLM不一样的新赛道。