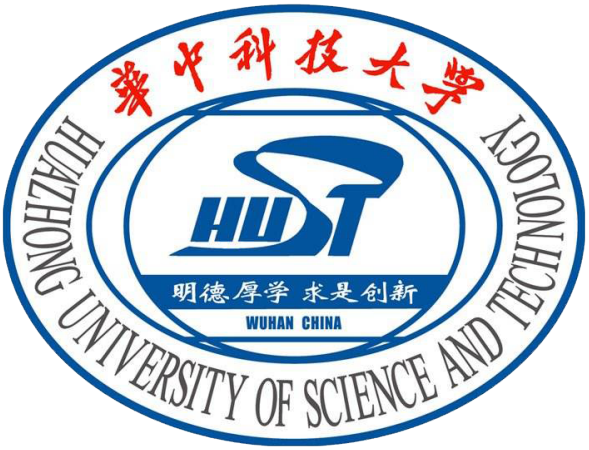
华中科技大学计算机科学与技术学院

《机器学习》结课报告



专　　业	计算机科学与技术
班　　级	本硕博 2301
学　　号	U202315752
姓　　名	陈宇航
成　　绩	
指导教师	张腾
时　　间	2025 年 4 月 11 日

目录

1 实验题目：以对数几率回归和决策树桩为基分类器的 AdaBoost 算法

1.1 摘要

本项目旨在通过实现 AdaBoost（Adaptive Boosting）算法来解决分类问题。AdaBoost 是一种强大的集成学习方法，通过结合多个弱分类器来构建一个强分类器，极大地提高分类精度。在本项目中，主要实现了两种基分类器：**决策树桩（Decision Stump）**和**逻辑回归（Logistic Regression）**。通过对比这两种基分类器，分析了 AdaBoost 在不同设置下的表现。

AdaBoost 通过迭代训练多个基分类器，并根据其在训练数据上的表现调整样本的权重，从而聚焦于分类错误的样本，逐步改进分类器的性能。每个基分类器的权重会根据其分类错误率进行调整，误差较小的分类器会获得更高的权重。最终，所有基分类器的预测结果会进行加权，得到最终的分类结果。

本项目的实验内容包括：基于 AdaBoost 的分类器训练、10 折交叉验证、结果评估及日志记录等。此外，还实现了对新数据的预测功能。该项目使用了 Python 语言，并通过模块化的方式实现了各个功能模块，便于进一步优化和扩展。

1.2 AdaBoost 算法介绍

AdaBoost 算法是一种基于加权投票的集成学习方法，旨在将多个弱分类器（单个分类器的性能略好于随机猜测）结合成一个强分类器。其核心思想是：通过将更多的关注放在那些被先前分类器错误分类的样本上，从而逐步提高分类性能。

AdaBoost 算法的工作流程如下：

1. **初始化权重**：为每个样本分配一个相等的初始权重。
2. **训练基分类器**：在每次迭代中，使用加权样本训练基分类器，并计算该分类器在训练集上的错误率。
3. **更新权重**：根据基分类器的错误率更新样本的权重，错误分类的样本权重将被增加，正确分类的样本权重将被减小。这样，下一轮训练将会更加关注那些难以分类的样本。
4. **加权预测**：所有基分类器的预测结果将根据其性能（错误率）进行加权，最终得到一个加权的决策结果。

通过反复迭代和加权，AdaBoost 不仅能提高分类精度，还能有效防止过拟合，特别是在弱分类器的性能相对较差时。

1.3 AdaBoost 算法实现

本项目中，AdaBoost 算法的实现包括了以下几个关键模块：

- **AdaBoost 类 (adaboost.py)**：主要负责训练过程，包含了 `fit` 方法用于训练基分类器，并根据错误率更新样本权重。`predict` 方法用于最终的预测，根据加权的基分类器输出结果。
- **基分类器**：实现了两种不同的基分类器——**决策树桩 (Decision Stump)** 和**逻辑回归 (Logistic Regression)**。其中，决策树桩通过选择最优的特征和阈值进行简单的二分类，逻辑回归则通过优化交叉熵损失进行训练，并利用梯度下降更新参数。
- **交叉验证**：为评估模型的性能，代码实现了 10 折交叉验证，通过将数据集划分为 10 个子集进行多次训练和验证，最终得出模型的平均准确率，确保算法的泛化能力。
- **数据加载与处理**：通过 `data_loader.py` 加载数据集，并通过 `cross_validation.py` 实现了数据的 10 折交叉验证划分，确保数据被充分训练和评估。

1.4 研究现状

AdaBoost 算法自 1995 年由 Yoav Freund 和 Robert Schapire 提出以来，已经成为集成学习领域的重要算法之一。AdaBoost 被证明能够通过多次迭代和加权集成多个弱分类器，显著提高分类精度。它特别适用于解决二分类问题，并在许多实际应用中取得了优异的性能，包括图像分类、文本分析、金融数据预测等。

1.4.1 过去的研究

AdaBoost 最初设计为用于弱分类器（通常为简单的分类器，如决策树桩）的加权组合。Freund 和 Schapire 在其原始论文中证明了 AdaBoost 在大多数情况下能够提供比单一分类器更好的性能，并且其具有良好的理论保证，即它能在适当条件下以指数级别减少训练误差。随着时间的推移，AdaBoost 已经被应用于各个领域，特别是在那些对分类精度要求较高且计算资源有限的任务中。

尽管 AdaBoost 在多种任务上表现出色，但也有其局限性。例如，在面对噪声数据和异常值时，AdaBoost 的表现可能会下降，因为其容易过度拟合噪声样本。为了克服这一缺陷，许多改进版本的 AdaBoost 被提出，如使用稳健的损失函数（如 LogitBoost 和 RobustBoost），以及基于加权的误差最小化策略的优化。

另外，AdaBoost 的基分类器的选择和算法的扩展也成为了研究的热点。除了传统的决策树桩，许多研究者还探讨了其他类型的基分类器，如支持向量机、神经网络等，以进一步提高其性能。

1.4.2 未来研究方向

随着深度学习和大数据时代的到来，AdaBoost 的应用场景和研究方向发生了新的变化。尽管深度学习方法在许多任务上取得了突破性进展，AdaBoost 作为一种简单高效的集成学习方法，依然在很多场合具有竞争力，特别是在数据量较小、计算资源受限的情况下。未来的研究可以集中在以下几个方面：

- **AdaBoost 与深度学习的结合**：尽管深度学习取得了显著的进展，但在某些应用中，AdaBoost 结合深度学习网络的特性，可能会提供更强大的性能。例如，利用 AdaBoost 强化神经网络的弱分类能力，或在深度学习的早期阶段应用 AdaBoost 来进行初步分类，再通过深度模型进行微调。

- **鲁棒性增强**：随着大数据应用的扩展，噪声数据和异常值问题变得愈加突出。未来的研究可以继续改进 AdaBoost，使其在处理不完整、缺失或噪声数据时仍能保持较高的性能。例如，研究如何通过改进样本权重更新机制，或在错误分类的基础上采用稳健的加权策略，以减少对噪声的敏感度。

- **大规模数据集上的应用**：随着计算能力的增强，AdaBoost 可以在更大规模的数据集上进行有效的训练。未来研究可能会重点解决 AdaBoost 在大规模数据集上的训练效率问题，并探索如何结合并行计算、分布式学习或流数据处理等技术，提升其在海量数据上的应用潜力。

- **算法优化与自适应机制**：针对传统 AdaBoost 算法在不同数据集和任务中的性能波动，研究者们提出了一些自适应机制。未来可以进一步研究自适应 AdaBoost 的变种，使其在面对不同的任务时能够自动选择最合适的学习策略和超参数设置。

此外，AdaBoost 还可以扩展到多分类问题、回归问题等其他任务中，尤其是在需要快速训练且能够解释模型决策的应用场景中，AdaBoost 依然是一种具有潜力的算法。

总之，AdaBoost 作为经典的集成学习算法，已经经历了多个发展阶段，并且仍然是机器学习研究中的一个活跃领域。随着新的算法优化和应用需求的不断涌现，AdaBoost 在未来的研究中将继续发挥其独特的作用，成为解决复杂问题的有力工具。

2 实验要求

2.1 总体要求

本次大作业的目标是实现一个基于 AdaBoost 算法的模型，其中需要使用对数几率回归（Logistic Regression）和决策树桩（Decision Stump）作为基分类器，进行模型的训练、参数调优以及最终的评估。要求手动实现这些模型，而不是调用已有的机器学习库。通过实验来探究 AdaBoost 算法如何在不同基分类器数量下表现，进行预测结果的交叉验证，并评估模型的性能。**实现以上算法时不允许调用已实现好的任何机器学习模型库**

2.2 训练数据集说明

2.2.1 数据集文件

该文件包含多个特征列，总共有 57 列，每列代表一个不同的特征。每行数据表示一个样本的特征值，这些特征是数值型数据。

示例数据中的每一行表示一个样本，每一列表示该样本的一个特征，特征值在每行中不同的列上有所体现。

在 AdaBoost 算法中，data.csv 提供了每个样本的特征数据，这些数据被用于训练每一轮的基分类器。每一轮训练后，样本的权重会根据分类器的错误分类情况进行调整，从而在下一轮的训练中更加关注错误分类的样本。最终，通过加权投票的方式，AdaBoost 将多个弱分类器组合成一个强学习器。

2.2.2 标签文件

在 AdaBoost 算法中，targets.csv 提供了每个样本的标签（目标值）。这些标签用于监督学习过程中的目标值，与特征数据（来自 data.csv）配对，告诉算法每个样本的实际类别。

targets.csv 包含每个样本对应的标签，每一行代表一个样本。在本次大作业中为二分类任务，标签为 0 和 1。在 AdaBoost 算法中，标签用于监督每一轮基分类器的训练。基分类器通过对 data.csv 中的特征进行学习，旨在预测与 targets.csv 中的标签一致的类别。

2.3 数据操作要求

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#，以供评测。基分类器数目取 1，5，10，100 这四种数值。

基分类器数目为 x 对应的预测文件为 `basex_fold1.csv` 到 `basex_fold10.csv`，其中 $1 \leq \text{fold} \leq 10$ 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。

3 算法设计与实现

本项目使用 AdaBoost 算法结合两种基分类器（决策树桩和逻辑回归）解决二分类问题。对于 AdaBoost 算法，其核心思想是通过加权组合多个弱分类器，逐步提高模型的分类精度。具体实现过程包括：初始化样本权重，然后在每次迭代中训练基分类器（如决策树桩或逻辑回归），根据分类错误率更新样本权重，并最终通过加权投票的方式输出最终的预测结果。在本项目中，使用了决策树桩作为一种简单且高效的基分类器，并通过逻辑回归进一步增强分类性能。通过调整基分类器的数量（即 AdaBoost 中的迭代次数）和样本权重的更新策略，可以在测试集上获得较高的分类准确率。

决策树桩 (Decision Stump) 是一种单层的决策树，通过选择一个特征和对应的阈值进行简单的二分类。逻辑回归 (Logistic Regression) 则通过优化交叉熵损失函数进行训练，使用梯度下降法更新参数。两种基分类器在 AdaBoost 框架下结合后，能够有效提升模型的分类精度，特别是在复杂的分类任务中。通过多轮迭代训练，AdaBoost 算法在测试集上最终达到了较高的分类准确率。

具体来说，我们的任务分为三步：

1. 数据集分析，读入和 k 折交叉处理的实现

2. 基分类器实现

3. 基于基分类器的 AdaBoost 算法的实现

这三步中，基分类器的实现极其关键。只要实现基分类器，AdaBoost 算法只需要接入其函数而算法内容可以通过多渠道获得

3.1 数据集的分析与数据处理

数据的读取和预处理：从 data.csv 文件中读取数据，此时可以打印输出观察一下文件里面的数据组成，见图3.1，并使用 np.genfromtxt 函数将其转换成 Array 的 data。然后我们可以输出里面的格式化数据和样式 (shape)，增加我们的理解。见图3.2

	0	1	2	3	4	5	6	7	8	9	...	47	48	49	50	51	52	53	54	55	56
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61.0	278.0
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101.0	1028.0
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485.0	2259.0
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40.0	191.0
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.0	0.00	0.223	0.0	0.000	0.000	0.000	3.000	15.0	54.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3675	0.00	0.00	1.19	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.000	0.0	0.000	0.000	0.000	1.000	1.0	24.0
3676	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.0	0.00	0.232	0.0	0.000	0.000	0.000	1.142	3.0	88.0
3677	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.000	0.0	0.353	0.000	0.000	1.555	4.0	14.0
3678	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.057	0.0	0.000	0.000	0.000	1.147	5.0	78.0
3679	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.000	0.0	0.125	0.000	0.000	1.250	5.0	40.0

3680 rows × 57 columns

图 3.1: 文件内容打印输出

```
array([[2.100e-01, 2.800e-01, 5.000e-01, ..., 5.114e+00, 1.010e+02,
        1.028e+03],
       [6.000e-02, 0.000e+00, 7.100e-01, ..., 9.821e+00, 4.850e+02,
        2.259e+03],
       [0.000e+00, 0.000e+00, 0.000e+00, ..., 3.537e+00, 4.000e+01,
        1.910e+02],
       ...,
       [0.000e+00, 0.000e+00, 0.000e+00, ..., 1.555e+00, 4.000e+00,
        1.400e+01],
       [9.600e-01, 0.000e+00, 0.000e+00, ..., 1.147e+00, 5.000e+00,
        7.800e+01],
       [0.000e+00, 0.000e+00, 6.500e-01, ..., 1.250e+00, 5.000e+00,
        4.000e+01]])
```

图 3.2: 部分数据可视化

可以看到，数据共有 3680 行和 57 列，代表有 3680 个数据和 57 个特征值。在 k 折交叉验证中（k_fold_validation），根据要求，我们取 k 为 10。交叉验证通过将数据集分成若干个子集，并交替使用每个子集进行测试，剩余部分用于训练，从而评估模型的泛化能力。当 k=10 时，数据集被分为十个子集，每次迭代选取一个子集作为测试集，其余九个子集作为训练集。这个过程通过 indices 来打乱数据顺序（shuffle=True），确保数据划分的随机性。日志记录了数据集折数和每一折大小，具体代码如下：

Code Listing 1: k-fold Split Algorithm

```
Input: Data matrix X, labels y, number of folds k, shuffle flag, random
    seed
n_samples = len(X)
indices = np.arange(n_samples)

if shuffle:
    np.random.seed(random_seed)
    np.random.shuffle(indices)

fold_size = n_samples // k
folds = []

for i in range(k):
    start = i * fold_size
    end = n_samples if i == k-1 else start + fold_size

    test_idx = indices[start:end]
    train_idx = np.concatenate((indices[:start], indices[end:]))

    X_train, y_train = X[train_idx], y[train_idx]
    X_test, y_test = X[test_idx], y[test_idx]

    folds.append((X_train, y_train, X_test, y_test, test_idx))

return folds
```

3.2 基分类器算法实现

在本次大作业中，我们使用了两种基分类器：

1. 对数几率回归（Logistic Regression）：对数几率回归是一种用于分类问题的线性模型，通过学习特征与标签之间的关系，模型输出每个类别的概率。其优点在于计算效率较高，且易于解释，但对于非线性数据的拟合能力较弱。

2. 决策树桩（Decision Stump）：决策树桩是一种非常简单的决策树模型，通常只有一层，用于处理简单的二分类问题。它通过选择一个特征并根据该特征的阈值进行数据划分，从而实现分类。决策树桩的优点在于训练速度快，适合于 AdaBoost 等集成学习方法，但其分类能力较弱，通常需要多个决策树桩的组合。

下面逐个描述

3.2.1 对数几率回归（Logistic Regression）

对数几率回归（Logistic Regression）是一种常用的二分类模型，尽管它的名字中有“回归”二字，但实际上它是一种分类算法。它的核心思想是通过拟合一个线性模型，并通过逻辑函数（Sigmoid）将线性输出映射到 $[0,1]$ 之间的概率值，进而进行二分类决策。对数几率回归在实际应用中广泛用于医学诊断、金融风控等领域。

模型描述：

在对数几率回归中，我们使用特征向量 $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ 来预测二分类标签 $y \in \{0,1\}$。模型通过一个线性组合来计算预测值，并使用 Sigmoid 函数将预测结果映射到概率空间。

$$P(y=1|\mathbf{x}) = \frac{1}{1+\exp(-(\mathbf{w}^T\mathbf{x}+b))}$$

其中，$\mathbf{w} = [w_1, w_2, \ldots, w_n]$ 是模型的权重，b 是偏置项，$\mathbf{x}$ 是特征向量，$P(y=1|\mathbf{x})$ 是样本属于类别 1 的概率。

损失函数：

对数几率回归通过最大化似然函数来训练模型。为了简化计算，通常对数似然函数的负对数称为损失函数。对于给定的样本集，损失函数为：

$$L(\mathbf{w},b) = -\sum_{i=1}^{n}[y_i \log(P(y_i=1|\mathbf{x}_i)) + (1-y_i)\log(1-P(y_i=1|\mathbf{x}_i))]$$

其中，n 是样本数，y_i 是样本 i 的真实标签，$\mathbf{x}_i$ 是样本 i 的特征向量，$P(y_i=1|\mathbf{x}_i)$ 是模型预测该样本属于类别 1 的概率。

梯度下降优化：

为了训练模型，我们使用梯度下降法来最小化损失函数。具体的权重和偏置更新公式如下：

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}, b)$$

$$b \leftarrow b - \eta \nabla_{b} L(\mathbf{w}, b)$$

其中，η 是学习率，$\nabla_{\mathbf{w}} L(\mathbf{w}, b)$ 和 $\nabla_{b} L(\mathbf{w}, b)$ 分别是损失函数对于权重和偏置的梯度。

正则化：

为了防止模型过拟合，我在损失函数中添加正则化项（L2 正则化），使得损失函数变为：

$$L(\mathbf{w}, b) = -\sum_{i=1}^{n} [y_i \log(P(y_i = 1|\mathbf{x}_i)) + (1 - y_i) \log(1 - P(y_i = 1|\mathbf{x}_i))] + \lambda \|\mathbf{w}\|^2$$

其中，λ 是正则化强度的超参数，$\|\mathbf{w}\|^2$ 是权重的平方和（即 L2 范数）。正则化项有助于控制模型复杂度，防止过拟合。

预测：

在训练过程中，模型学到的权重 $\mathbf{w}$ 和偏置 b 将用来预测新样本的类别。给定一个新的特征向量 $\mathbf{x}$，我们通过计算 $P(y = 1|\mathbf{x})$，并将结果与阈值 0.5 进行比较来做出分类决策，从而实现了回归任务向分类任务的转变：

$$\hat{y} = \begin{cases} 1, & \text{if } P(y = 1|\mathbf{x}) \geq 0.5 \\ 0, & \text{if } P(y = 1|\mathbf{x}) < 0.5 \end{cases}$$

伪代码如下：

Code Listing 2: Logistic Regression Algorithm

```
Input: Data matrix X, labels y, learning rate lr, maximum iterations
    max_iter, regularization strength reg_strength
Initialize: weights w and bias b randomly
For each iteration in range(max_iter):
    # Linear combination of inputs and weights
    linear_output = X.dot(w) + b

    # Sigmoid function to get  probabilities
    probs = sigmoid(linear_output)

```

```
        # Calculate gradient for weights and bias
        gradient_w = X.T.dot(probs - y) + reg_strength * w
        gradient_b = np.sum(probs - y)

        # Update weights and bias using gradient descent
        w -= lr * gradient_w
        b -= lr * gradient_b

        # Calculate and log the loss for monitoring
        loss = -np.sum(y * np.log(probs + 1e-10) + (1 - y) * np.log(1 -
            probs + 1e-10)) + reg_strength * np.sum(w ** 2)
        If iteration % 10 == 0:
            log loss
    Return: weights w, bias b
```

3.2.2 决策树桩基分类器（Decision Stump）

决策树桩（Decision Stump）是一种简单的决策树模型，通常只有一层（即单个节点），用于处理二分类问题。它通过选择一个特征并使用该特征的某个阈值来对数据进行划分，决定样本的类别。在 AdaBoost 集成学习方法中，决策树桩作为弱分类器，参与多轮迭代训练，组合成一个强分类器。

模型描述：

决策树桩选择一个特征和一个阈值，并基于该特征的值来划分数据。具体地，假设我们有一个数据集 $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$，其中 $\mathbf{x}_i$ 是样本 i 的特征向量，$y_i \in \{+1, -1\}$ 是标签。决策树桩通过选择一个特征 x_j 和一个阈值 t 来进行分类：

$$h(\mathbf{x}) = \begin{cases} +1, & \text{if } x_j \geq t \\ -1, & \text{if } x_j < t \end{cases}$$

其中，x_j 是特征向量中的第 j 个特征，t 是阈值。

训练过程：

决策树桩的训练过程是遍历所有特征和特征的可能阈值，并选择使分类误差最小的特征和阈值。训练的目标是找到一个特征和阈值，使得根据该特征划分后的分类误差最小。分类误差定义为：

$$\text{Error} = \sum_{i=1}^{n} \text{weight}_i \cdot I(h(\mathbf{x}_i) \neq y_i)$$

其中，weight_i 是样本 i 的权重，$I(\cdot)$ 是指示函数，当 $h(\mathbf{x}_i)$ 与 y_i 不同（即分类错误）时，$I(h(\mathbf{x}_i) \neq y_i) = 1$，否则为 0。

在训练过程中，决策树桩还可以考虑通过反转类别的符号（即 polarity）来最小化误差。如果某个特征和阈值的分类误差大于 0.5，则选择反转类别标签，使得误差变为 $1 - \text{error}$，从而降低总误差。

模型预测：

在预测时，给定一个新的样本 $\mathbf{x}$，决策树桩会根据训练时选定的特征和阈值进行分类：

$$h(\mathbf{x}) = \begin{cases} +1, & \text{if } x_j \geq t \\ -1, & \text{if } x_j < t \end{cases}$$

最终的分类预测结果是该特征和阈值对应的分类结果，可能还会根据训练时选择的 polarity 来反转预测结果。

伪代码如下：

Code Listing 3: Decision Stump Pseudocode

```
Input: Data matrix X, labels y, sample weights sample_weight (optional)
Initialize: feature_idx = None, threshold = None, polarity = 1, min_error = inf
For each feature f in X:
    Get unique values in feature f
    For each threshold t in unique values of feature f:
        Initialize predictions: preds = 1 for all samples
        For each sample, if feature value < threshold, set preds = -1
        Calculate misclassification error: error = sum(sample_weight * (preds != y))

        If error > 0.5:
            Set error = 1 - error, set polarity = -1

        If error < min_error:
            Update min_error, feature_idx, threshold, and polarity

Output: feature_idx, threshold, polarity
```

3.3 基于基分类器的 AdaBoost 实现

AdaBoost（Adaptive Boosting）是一种集成学习方法，将多个弱分类器（此处为 `Logistic Regression` 和 `Decision Stump`）组合成一个强分类器，在保持每个基分类器简单的同时显著提高整体性能。本节介绍基于逻辑回归或决策树桩作为基分类器的 AdaBoost 实现.

模型描述：

AdaBoost 的基本思想是以加权的方式组合多个弱分类器 $h_t(\mathbf{x})$，形成最终的强分类器 $H_T(\mathbf{x})$：

$$H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}),$$

其中，T 表示迭代轮数，α_t 表示第 t 个弱分类器的权重，$h_t(\mathbf{x})$ 是第 t 轮训练得到的弱分类器，其输出为 $\{-1, +1\}$。

训练过程：

AdaBoost 的目标是最小化以下指数损失函数：

$$\mathcal{L}(H) = \sum_{i=1}^{n} \exp(-y_i H_T(\mathbf{x}_i)),$$

其中 $y_i \in \{-1, +1\}$ 是标签，$H_T(\mathbf{x}_i)$ 是当前组合分类器对样本的预测值。

在每一轮迭代 t，模型执行以下步骤：

1. 根据样本的当前权重分布 $D_t(i)$（初始化为均匀分布）训练一个弱分类器 h_t。2. 计算该弱分类器的加权错误率：

$$\epsilon_t = \sum_{i=1}^{n} D_t(i) \cdot \mathbb{I}(h_t(\mathbf{x}_i) \neq y_i).$$

3. 计算该分类器的权重 α_t（加入学习率 η）：

$$\alpha_t = \eta \cdot \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right).$$

4. 更新样本的权重：

$$D_{t+1}(i) \propto D_t(i) \cdot \exp(-\alpha_t y_i h_t(\mathbf{x}_i)),$$

并进行归一化，使 $\sum_i D_{t+1}(i) = 1$。

5. 保存当前弱分类器和其对应的权重。

模型预测：

预测阶段，AdaBoost 使用所有训练得到的弱分类器按照其权重进行加权投票。具体地，预测值为：

$$H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}),$$
$$\hat{y} = \text{sign}(H_T(\mathbf{x})).$$

如果我们将标签从 $\{-1, +1\}$ 映射回 $\{0, 1\}$，则有：

$$\hat{y}_{\text{final}} = \begin{cases} 1, & \text{if } \hat{y} = +1 \\ 0, & \text{if } \hat{y} = -1 \end{cases}$$

伪代码如下：

Code Listing 4: AdaBoost Pseudocode

```
Input: Training data (X, y), base model factory get_base_model,
    iterations T, learning_rate eta
Initialize sample weights: w = np.ones(n_samples) / n_samples
Initialize models = [], alphas = []

for t in range(T):
    model = get_base_model()
    model.fit(X, y, sample_weight=w)
    pred = model.predict(X)

    mis = (pred != y)
    error_t = np.sum(w * mis)

    # Compute alpha with learning rate
    alpha_t = 0.5 * np.log((1 - error_t + eps) / (error_t + eps)) * eta

    # Update weights
    w = w * np.exp(-alpha_t * y * pred)
    w /= np.sum(w)

    models.append(model)
    alphas.append(alpha_t)

Output: sign(sum(alpha_t * model.predict(X))) for all models
```

AdaBoost 的基分类器兼容任何支持 `sample_weight` 参数的分类器（如 sklearn 的 `DecisionTreeClassifier` 或 `LogisticRegression`），并且通过引入学习率参数 η 控制每一轮学习器对最终模型的影响，防止过拟合，提高泛化能力。

4 实验环境与平台

表 4.1: **硬件信息**

项目	信息
设备名称	华硕无畏 16pro
CPU	13th Gen Intel Core i9-13900H (24) @ 5.00GHz
GPU	NVIDIA GeForce RTX 4050 Mobile
RAM	32GB DDR5
存储	1TB NVMe SSD + 2TB HDD（机械硬盘）
系统类型	x86_64
显示器	16 英寸 QHD (2560x1600) IPS 显示屏，165Hz 刷新率
操作系统	Windows 11

表 4.2: **软件信息**

项目	信息
操作系统	Windows 11 Home
版本	22000.556
IDE	PyCharm Professional 2024.1.1
Python	Python 3.12.3
Jupyter	JupyterLab v3.2.1
Python library	numpy pandas

5 代码框架与测试分析

5.1 代码框架

1. 数据加载与准备 (data_loader.py): 首先，通过 `load_data` 函数读取数据文件（`data.csv`）和标签文件（`targets.csv`）。该函数返回特征矩阵 X 和标签向量 y。

2. 交叉验证划分 (cross_validation.py): 然后，使用 `k_fold_split` 函数将数据划分为 $k = 10$ 折交叉验证数据集。每一折生成训练集和测试集，数据根据需要进行洗牌。该函数的输出是包含训练集和测试集的折列表 `folds`。

3. 训练过程 (main.py): 在主程序中，基于命令行输入的基分类器编号（Logistic Regression 或 Decision Stump）选择相应的基分类器。接着，使用 AdaBoost 类进行模型训练:

$$\text{AdaBoost}\left(\text{get\_base\_model}, n\_estimators = 10\right)$$

在 `fit` 方法中，基分类器通过多次迭代进行训练，逐步调整样本权重，最终得到加权组合模型。

4. 预测 (main.py): 在预测阶段，使用训练好的 AdaBoost 模型对新数据进行预测。每个基分类器的预测结果会加权汇总，最终得到最终预测标签 $\hat{y}$:

$$\hat{y} = \text{sign}\left(\sum_{t=1}^{n\_estimators} \alpha_t \cdot \text{model}_t(x)\right)$$

其中 α_t 是第 t 个基分类器的权重，$\text{model}_t(x)$ 是基分类器对样本 x 的预测。

5. 交叉验证评估 (evaluate.py): 最后，使用 `evaluate.py` 脚本对每个折的预测结果进行评估。计算每个折的准确率，并输出所有折的平均准确率。每个折的准确率 accuracy_i 计算公式为:

$$\text{accuracy}_i = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{\hat{y}_j = y_j\}}}{n}$$

其中 $\hat{y}_j$ 是第 j 个样本的预测标签，y_j 是真实标签。

具体框架如下图5.1，所有的代码日志都在 `log` 文件夹下，评估日志在 `eva_log` 目录下（注：`eva_log_Before_the_final_version_of_the_code` 和 `log_Before_the_final_version_of_the_code` 是代码调试定稿前的日志）。

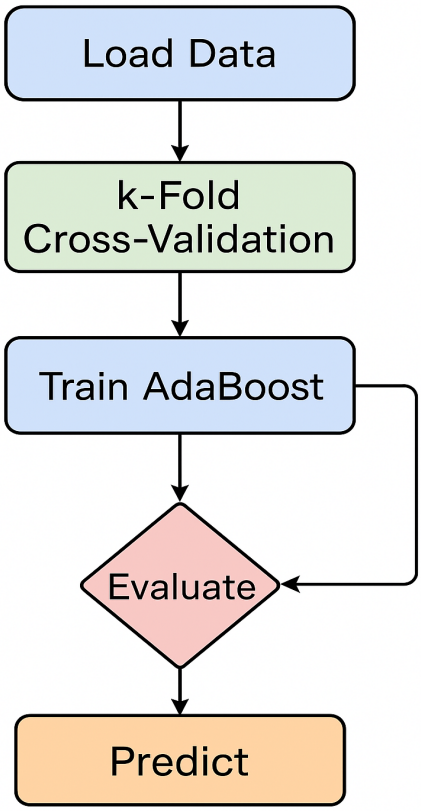


图 5.1: 代码框架图

5.2 测试结果

5.2.1 测试 1：在固定学习率为 0.2，max_iter 为 300 的 LogisticRegression 下的精确度测试

使用 LogisticRegression 模型作为基模型，进行 1,5,10,100 为基分类数目的集成学习，日志如下图5.2, 测试结果如下图5.3, 可以看到精度在 0.6 左右，不是很高

```
2025-04-10 23:31:53 [INFO] __main__:37 - 基分类器编号: 0
2025-04-10 23:31:53 [INFO] __main__:64 - 使用基分类器: LogisticRegression 进行10折交叉验证，并输出到 experiments/ 文件夹
2025-04-10 23:31:53 [INFO] utils.data_loader:9 - load_data: 读取 data.csv 和 targets.csv, 样本数=3680, 特征数=57
2025-04-10 23:31:53 [INFO] utils.cross_validation:26 - k_fold_split: 已拆分为 10 折, each fold size~368 (最后一折=368)
2025-04-10 23:31:53 [INFO] adaboost.adaboost:14 - n_estimators is 1, learning_rate is 0.2
2025-04-10 23:31:53 [INFO] base_models.logistic_regression:14 - learning_rate is 0.02,max_iter is300,L2 reg_strength is0.01
2025-04-10 23:31:53 [INFO] adaboost.adaboost:25 - model is <base_models.logistic_regression.LogisticRegressionModel object at 0x000001D3D7BFD220>
2025-04-10 23:31:53 [INFO] adaboost.adaboost:14 - n_estimators is 1, learning_rate is 0.2
2025-04-10 23:31:53 [INFO] base_models.logistic_regression:14 - learning_rate is 0.02,max_iter is300,L2 reg_strength is0.01
2025-04-10 23:31:53 [INFO] adaboost.adaboost:25 - model is <base_models.logistic_regression.LogisticRegressionModel object at 0x000001D3D799AF40>
2025-04-10 23:31:53 [INFO] adaboost.adaboost:14 - n_estimators is 1, learning_rate is 0.2
2025-04-10 23:31:53 [INFO] base_models.logistic_regression:14 - learning_rate is 0.02,max_iter is300,L2 reg_strength is0.01
2025-04-10 23:31:53 [INFO] adaboost.adaboost:25 - model is <base_models.logistic_regression.LogisticRegressionModel object at 0x000001D3D7BFD220>
2025-04-10 23:31:53 [INFO] adaboost.adaboost:14 - n_estimators is 1, learning_rate is 0.2
2025-04-10 23:31:53 [INFO] base_models.logistic_regression:14 - learning_rate is 0.02,max_iter is300,L2 reg_strength is0.01
```

图 5.2: 学习日志

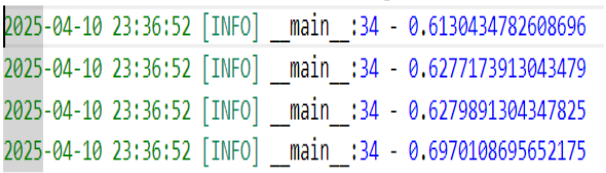


图 5.3: 测试结果

5.2.2 测试 2：在固定学习率为 0.2，max_iter 为 300 的 DecisionTreeClassifier 下的精确度测试

使用 LogisticRegression 模型作为基模型，进行 1,5,10,100 为基分类数目的集成学习，日志如下图5.4, 测试结果如下图5.5, 可以看到精度最初在 0.7 左右，随着基分类器数目增加，增加到 0.92 以上，增加效果较好

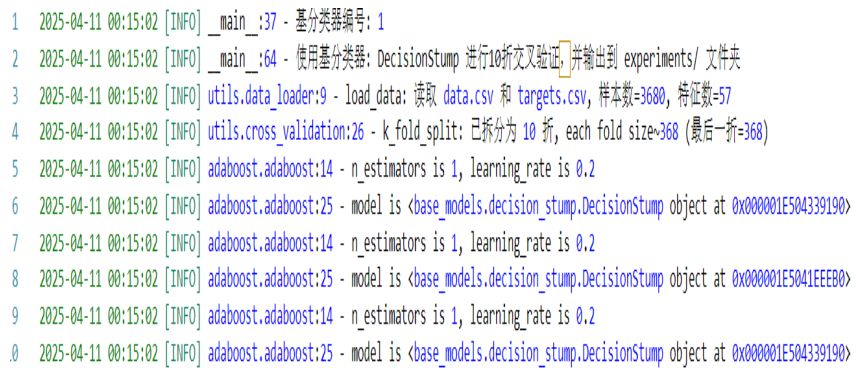


图 5.4: 学习日志

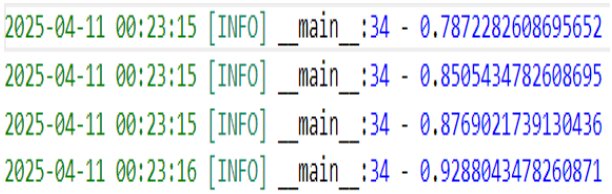


图 5.5: 测试结果

5.2.3 测试 3：在不同学习率下，max_iter 为 300LogisticRegression 下的精确度测试

使用 LogisticRegression 模型作为基模型，进行 1,5,10,100 为基分类数目的集成学习，为了探究哪种学习率下训练效果最好，我们在尝试之后得到数据精确度在学习率在 0.05 到 0.2 之间出现最大值，然后设置学习率在此之间进行训练测试，数据如图5.6所示，我们使用 `matplotlib` 进行绘图分析5.7

学习率	0.0071	0.0068	0.0069	0.00685	0.00707	0.00663	0.00625	0.0075	0.015	0.005	0.01	0.02	0.05	0.1	0.2
1个分类器	0.599728	0.597554	0.614946	0.613315	0.569022	0.594565	0.608152	0.566304	0.585054	0.572283	0.622554	0.596196	0.592935	0.605163	0.6130434
5个分类器	0.628533	0.633967	0.633424	0.625543	0.634239	0.613859	0.613859	0.629891	0.625815	0.615217	0.6125	0.632609	0.642663	0.636141	0.627717
10个分类	0.631793	0.639674	0.640761	0.643478	0.630978	0.620652	0.629891	0.644293	0.63913	0.640217	0.627174	0.628804	0.620109	0.634239	0.637989
100个分类	0.69538	0.693478	0.700815	0.696467	0.70788	0.707065	0.69837	0.703261	0.68587	0.691848	0.698641	0.690489	0.689674	0.6875	0.69701

图 5.6: 数据内容

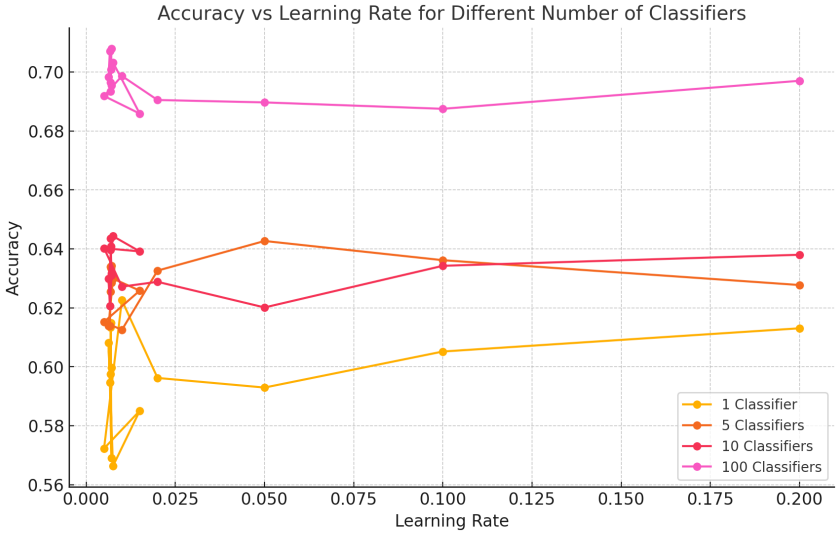


图 5.7: 图示内容

学习率的影响

图表显示，随着学习率从 0.005 增加到 0.1，模型的准确率通常有所提升。对于非常低的学习率（例如 0.005），模型的准确率相对较低，这可能是由于训练过程中的收敛速度过慢。随着学习率的增加，准确率也逐渐提高，在大多数分类器数量下，准确率在 0.00707 附近达到峰值。超过 0.00707 后，准确率开始趋于平稳或略有下降，表明学习率可能过高，导致训练过程中的过拟合或不稳定。

基分类器数量的影响

对于 **1 个分类器**，在所有学习率下，准确率相对较低，表明使用单一弱分类器并未取得最佳效果。**5 个分类器**的准确率较 1 个分类器有所提升，在所有学习率下的准确率普遍更高。**10 个分类器**继续提高准确率，并在较高学习率范围（0.05-0.2）内保持稳定。**100 个分类器**的准确率是最高的，特别是在较高学习率范围（约 0.1），这表明增加更多的分类器可以提高模型捕捉更复杂数据模式的能力。

最佳学习率

对于大多数分类器配置，最佳学习率似乎是在 **0.1** 左右，在该学习率下准确率达到峰值，特别是在 10 个和 100 个分类器的情况下。在 **0.05 到 0.2** 之间的学习率能为所

有分类器数目带来最高的性能，而 **0.0707** 是大多数分类器的最佳学习率。

结论

该图表表明，对于使用 AdaBoost 的逻辑回归模型，适中的学习率（约 0.1 左右）能带来最佳的性能，尤其是在使用更多分类器时。此外，增加分类器的数量能提高准确率，然而超过 10 个分类器后，准确率的提升逐渐减缓。

5.2.4 测试 4：在不同学习率下的 DecisionTreeClassifier 下的精确度测试

使用 DecisionTreeClassifier 模型作为基模型，进行 1,5,10,100 为基分类数目的集成学习，为了探究哪种学习率下训练效果最好，我们在尝试之后得到数据精确度在学习率在 0.1 到 1.5 之间出现最大值，然后设置学习率在此之间进行训练测试，数据如图5.10所示，我们使用 `matplotlib` 进行绘图分析5.9

学习率	0.2	0.1	0.3	0.4	0.5	0.8	1.5	1.2	1	1.15	1.05	1.1	1.075	1.025	1.015	1.01	1.0105	1.0125	1.011	1.01075	1.01025	1.01063	1.01054	1.01045	1.01048
1个分类器	0.787228	0.787228	0.787228	0.787228261	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228	0.787228261
5个分类器	0.850543	0.826087	0.866576	0.867663043	0.874457	0.886957	0.869837	0.895652	0.898641	0.897011	0.9	0.9	0.9	0.9	0.9	0.898641	0.898641	0.9	0.9	0.9	0.898641	0.898641	0.898641	0.898641	0.9
10个分类	0.876902	0.854348	0.891033	0.900271739	0.905435	0.905435	0.90625	0.907065	0.908967	0.904076	0.910598	0.9125	0.911685	0.911141	0.911957	0.911957	0.911957	0.911957	0.911957	0.911957	0.911957	0.911957	0.911957	0.911957	0.911956522
100个分类	0.928804	0.919565	0.93288	0.936956522	0.939946	0.942663	0.937228	0.939402	0.945109	0.943207	0.945109	0.944837	0.944565	0.946196	0.946467	0.947554	0.948913	0.947011	0.947011	0.94837	0.947554	0.94837	0.94837	0.94837	0.950113043

图 5.8: 数据内容

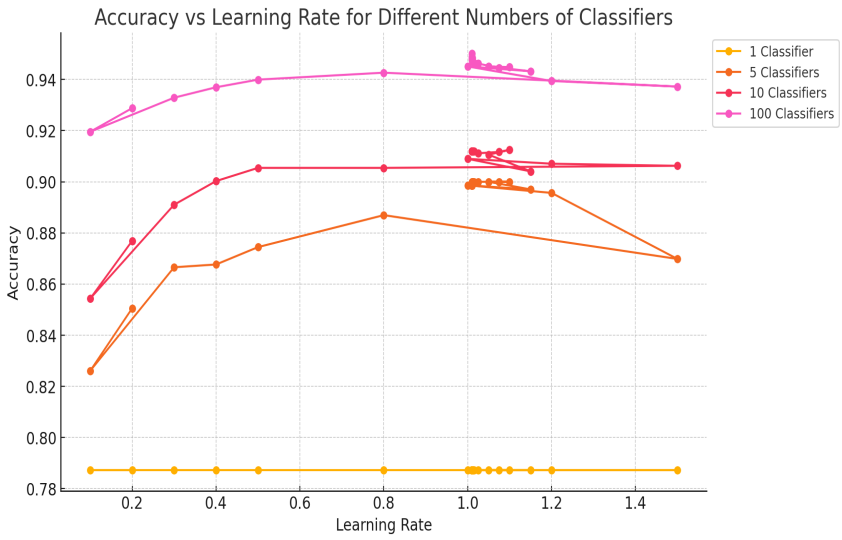


图 5.9: 图示内容

数据分析

1 个分类器的影响

无论学习率如何变化，1 个分类器的精确度始终保持在 0.7872，这表明单一分类器的学习能力有限，调整学习率对其精确度几乎没有影响。

5 个分类器的影响

随着学习率的增加，精确度表现出波动。尤其在学习率为 0.8 和 1.0 时，精确度达到了 0.9，显示出较强的性能。此外，较高的学习率（如 1.5）似乎导致精确度下降，说明较大学习率可能导致模型的过拟合或不稳定。

10 个分类器的影响

10 个分类器的精确度进一步提高，尤其在学习率 1.0 附近，精确度达到了 0.91。整体来看，随着分类器数量的增加，模型能更好地捕捉数据的复杂性，学习率对性能的影响更加明显。

100 个分类器的影响

当基分类器数量达到 100 个时，精确度显著提高，特别是在学习率 1.0 到 1.5 范围内，精确度接近 0.95。这表明随着分类器数量的增加，集成方法的表现大幅提升，特别是在适当的学习率下。

结论

基分类器数量的影响：随着基分类器数量的增加，模型的精确度显著提高。100 个分类器的精确度最高。

学习率的影响：学习率对较少分类器的影响较小，但在分类器数量较多时（如 10 和 100 个分类器），学习率对精确度的提高起到了至关重要的作用。最佳学习率在 1.0 到 1.5 范围内。

思考：不同学习率对不同分类器数目效果不同步的原因

- **模型复杂度与学习率**：当基分类器数量较少时，模型较简单，较小的学习率可能导致训练过于缓慢，而较大的学习率则可能导致训练不稳定。随着分类器数量的增加，模型变得更加复杂，较大的学习率可能加速收敛，提高性能。
- **集成学习的效果**：在集成学习中，较少的分类器对学习率更敏感，因为单一分类器的能力有限；而当分类器数量增多时，分类器之间的互补性增强，较大的学习率可能导致过拟合，较小的学习率则可能导致欠拟合。

5.2.5 测试 5：在最佳学习率为 0.0707 下，max_iter 为在 50 至 1000 之间 LogisticRegression 下的精确度测试

使用 LogisticRegression 模型作为基模型，进行 1,5,10,100 为基分类数目的集成学习，为了探究哪种迭代数下训练效果最好，在最佳学习率 0.0707 下，50 到 1000 每隔 50

设置步长进行训练测试，数据如图??所示，我们使用 `matplotlib` 进行绘图分析5.11：

max_iter	50	100	150	200	250	300	350	400	450	500	550	600	700	650	750	800	850	900	950	1000
1个分类器	0.583967	0.578533	0.584239	0.57962	0.547283	0.569022	0.576902	0.58125	0.584783	0.61712	0.597283	0.588587	0.61413	0.590217	0.613043	0.613043	0.614946	0.619293	0.608967	0.631522
5个分类器	0.597554	0.6125	0.590217	0.610054	0.626359	0.625815	0.638859	0.646739	0.662228	0.67663	0.65788	0.666848	0.675543	0.684239	0.675	0.658967	0.707609	0.654076	0.700272	0.699728
10个分类	0.60625	0.62663	0.631793	0.65163	0.627989	0.661685	0.679891	0.665761	0.695109	0.678261	0.67038	0.708696	0.699457	0.680163	0.710326	0.713315	0.698913	0.701359	0.706793	0.709783
100个分类	0.614674	0.665217	0.676087	0.704891	0.708152	0.734239	0.726087	0.749185	0.755707	0.791576	0.760598	0.792935	0.768207	0.775	0.795924	0.776902	0.816033	0.798641	0.769022	0.791304

图 5.10: 数据内容

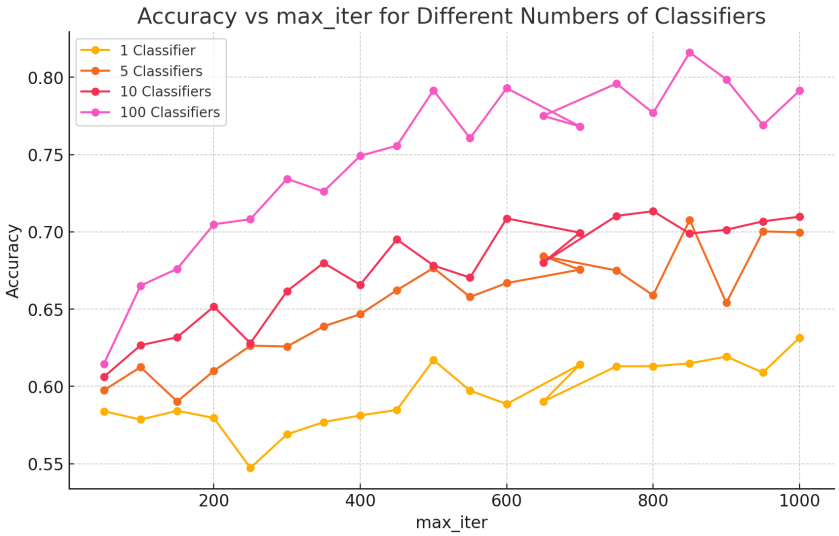


图 5.11: 图示内容

数据分析

1 个分类器（1 Classifier）

随着 max_iter 增加，精确度有所提升，但精确度变化较小，始终保持在较低水平，约为 0.58 到 0.63。这表明单一分类器的学习能力较弱，调整迭代次数对其精确度影响有限。

5 个分类器（5 Classifiers）

精确度从约 0.59 提升至 0.70 左右，表明增加分类器数目能提高模型的学习能力，且迭代次数增加后，模型逐步收敛，精确度逐渐提高。

10 个分类器（10 Classifiers）

精确度持续提高，尤其在 max_iter 较大的情况下（约 600 及以上）。精确度达到接近 0.71，表明更多的分类器在适当的迭代次数下能进一步提升模型性能。

100 个分类器（100 Classifiers）

精确度显著提高，在较大的迭代次数下（如 900 和 1000），精确度接近 0.75。这表明随着分类器数量的增加，模型的学习能力增强，更多的迭代能帮助优化模型，提升精确度。

结论

- **基分类器数量的影响：**随着基分类器数量的增加，精确度显著提升。100 个分类器的精确度最高。
- **max_iter 的影响：**随着 max_iter 的增加，精确度持续提升，尤其在 10 和 100 个分类器的情况下，更多的迭代次数显著改善了模型的性能。

思考：max_iter 不断增大，精确度出现摇摆的原因

1. 过拟合和欠拟合的影响

过拟合：当 max_iter 增加时，模型可能开始过拟合训练数据。尤其是在迭代次数增加时，模型可能会开始记住训练集的噪声而不是学习到数据的真实模式。特别是对于较少分类器（如 1 或 5 个分类器），过多的迭代可能导致模型的表现不稳定，从而导致精确度在某些迭代中上升，而在其他迭代中下降。

欠拟合：当 max_iter 过小或者在某些学习率下迭代不够时，模型的表现也可能因为训练不足而出现波动，表现为精确度的上下起伏。

2. 学习率与迭代次数的相互作用

学习率与 max_iter 的关系：虽然在最佳学习率下，增加 max_iter 应该会使得模型逐渐收敛并提高精确度，但学习率的设定也影响着训练的稳定性。如果学习率与 max_iter 结合不当，可能会导致训练过程中模型参数更新不稳定，从而表现为精确度的波动。

震荡现象：如果学习率较高，可能在一定的迭代次数下，模型无法稳定收敛，导致精确度在不同的迭代中出现较大波动。

3. 集成学习与迭代次数

集成学习的复杂性：集成学习方法（如 AdaBoost 或 Bagging）通过组合多个弱分类器形成一个强分类器。对于较少分类器数目的情况下，增加 max_iter 会让每个分类器的学习过程更加充分，可能导致稳定性问题（过拟合）。但是，随着基分类器数量的增加，集成方法的稳定性有所增强，总体上 max_iter 增加仍然有助于提升最终的性能。。

4. 优化过程中的波动

梯度下降的步长问题：每次迭代都会根据梯度更新模型的权重。较大的 max_iter 可以让模型进行更多的优化步骤，但如果在训练过程中模型的更新步长较大，可能会导致震荡，使得精确度在训练过程中出现波动。

学习过程不稳定：虽然 max_iter 增加，模型依然没有完全收敛，导致训练过程的不稳定，特别是在对训练集的拟合过程中，精确度出现了来回波动。

5. 数据分布和噪声

数据的随机性和噪声：数据的分布和特征存在噪声，某些学习率和迭代次数组合可能导致模型在特定的数据集上表现更好，而在其他数据集上表现较差，造成精确度的波动。

5.3 改进算法思考

Boosting 算法升级：Boosting 是一种集成学习方法，通过将多个弱学习器（通常是简单模型）结合起来，得到一个强学习器。最早的 Boosting 算法是 AdaBoost（Adaptive Boosting），它通过加权组合多个弱分类器来提高整体的分类性能。AdaBoost 通过调整样本权重，使得模型在每次迭代时关注难以分类的样本，从而逐步减少训练误差。

然而，随着 Boosting 算法的广泛应用，人们发现其存在一定的局限性，比如对于噪声数据的敏感性、过拟合等问题。为了解决这些问题，许多改进版的 Boosting 算法应运而生，诸如 Gradient Boosting Machines (GBM)、XGBoost、LightGBM 和 CatBoost 等，它们在 AdaBoost 的基础上，使用了不同的损失函数、优化方法以及正则化策略，大大提升了模型的性能与鲁棒性。

Boosting 算法的超参数调优：对于 Boosting 系列算法，超参数的选择和调优可以优化算法效果。常见的超参数包括：

- **基分类器选择（Base Learner）**：AdaBoost 最初使用的是决策树桩（Decision Stump）作为基分类器，但在后来的研究中，发现使用更复杂的基分类器（如完全决策树、逻辑回归等）可以进一步提高模型的性能。因此，选择合适的基分类器是提高模型性能的关键。

- **学习率（Learning Rate）**：学习率决定了每次基分类器对最终模型贡献的大小。较高的学习率可以使模型快速收敛，但也容易导致过拟合；较低的学习率可以增加模型训练的稳定性，但可能导致训练时间变长。因此，学习率通常需要根据数据的特点进行调优。

- **弱分类器数量（n_estimators）**：弱分类器的数量直接影响模型的复杂度和训练时间。过多的基分类器可能导致过拟合，而过少则可能导致欠拟合。常见的做法是使用交叉验证（cross-validation）来选择合适的基分类器数量。

- **正则化（Regularization）**：在一些改进版的 Boosting 算法中，如 XGBoost、LightGBM 等，正则化项被引入来防止模型过拟合。L1 和 L2 正则化常用于控制模型的复杂度，避免过度依赖某些特征。

- **最大深度（Max Depth）**：在使用决策树作为基分类器时，树的最大深度是一个重要的超参数。过深的树可能会导致过拟合，过浅的树则可能导致欠拟合。因此，需要根据数据集的复杂度来调整此参数。

- **子采样（Subsampling）**：一些 Boosting 算法（如 XGBoost）允许对每一轮训练使用不同的训练子集，这种技术被称为子采样。通过引入噪声和随机性，子采样可以有效减少过拟合，提高模型的泛化能力。

算法的调优策略：通常，超参数调优采用两种常见策略：

1. **网格搜索（Grid Search）**：通过指定一组可能的超参数值，遍历所有组合并评估模型的表现。网格搜索的优势是可以全面搜索超参数空间，但计算量较大，尤其在超参数空间较大时。

2. **随机搜索（Random Search）**：与网格搜索不同，随机搜索在超参数空间中随机选择组合进行评估。这种方法在计算资源有限的情况下能够较为高效地找到近似最优的超参数组合。

3. **贝叶斯优化（Bayesian Optimization）**：贝叶斯优化是一种基于概率模型的优化方法，它通过对超参数空间的概率建模，逐步选择最有可能提高模型性能的超参数。贝叶斯优化相较于网格搜索和随机搜索在调优效率和计算资源利用上具有优势。

改进思路与挑战：尽管 Boosting 算法已被广泛应用并取得了显著的成果，但仍有改进的空间。以下是一些潜在的改进方向：

- **鲁棒性提高**：Boosting 算法对于噪声和异常值较为敏感。未来的研究可以集中在设计更加鲁棒的损失函数或基分类器，来减少噪声的影响。

- **并行化与加速**：由于 Boosting 算法的训练过程具有一定的顺序性，这使得其训练速度较慢。为了解决这一问题，可以考虑并行化或分布式计算框架（如 XGBoost、LightGBM 已经在这一方向上有所突破）。

- **自动化超参数调优**：近年来，自动化机器学习（AutoML）技术迅速发展，能够自动搜索最佳超参数配置。结合 AutoML 技术与 Boosting 算法，未来有可能进一步提高调优效率和模型性能。

。

6 机器学习课程的学习体会与建议

6.1 项目总结

本次大作业的任务是分别实现以对数几率回归（Logistic Regression）和决策树桩（Decision Stump）为基分类器的 AdaBoost 算法。该任务不仅要求我们深入理解 AdaBoost 的算法机制，还需要掌握两种基分类器的数学模型与实现细节。

在深入学习和查阅文献的基础上，我逐步构建了解决方案，并将其细化为清晰的模块。在系统架构设计方面，我采用了面向对象的方式，将每种基分类器封装成一个函数工厂，使得 AdaBoost 能够灵活地组合不同类型的弱分类器。这种设计不仅使系统具有较好的扩展性，也提高了整体的可维护性和清晰度。

在实现的过程中，我还特别增加了日志模块，便于观察模型在不同轮数，不同学习率和不同选择基数下的预测变化，有助于更直观地理解 AdaBoost 如何逐轮优化模型表现。此外，我尝试对不同学习率参数进行调节，观察其对最终模型性能的影响，这也加深了我对 AdaBoost 控制收敛速度与过拟合风险之间平衡机制的理解。

通过本次项目，我对 AdaBoost 算法中最小化指数损失、权重更新机制、弱学习器加权组合等关键步骤有了深入体会。同时，对数几率回归作为一个可微可优化的概率模型，其在 AdaBoost 中的表现也使我更直观地理解了“线性弱分类器在非线性集成中的强大表现”。

6.2 反思

虽然本次项目实现了指定的功能目标，但在回顾过程中，我也发现了多个可以进一步优化的方向。

首先，由于实现重点放在了单变量特征空间（二维或低维可视化数据集）上，目前系统对高维数据的支持仍不充分。虽然 Logistic Regression 和 AdaBoost 都理论上适用于高维空间，但为了便于调试和可视化，本次实验未进行高维数据的系统测试。

其次，在决策树桩的训练中，我采用的是遍历所有特征及可能的划分阈值的方法。尽管这种方式较为精确，但计算复杂度仍可优化，例如采用排序后的中点分割策略，以减少不必要的重复判断。

此外，我在调参过程中观察到，当基分类器使用 Logistic Regression 时，AdaBoost 的学习率对最终性能影响较大，但目前系统未集成自动调参模块。未来可以通过交叉验证自动选取最优参数，从而增强泛化能力。

最后，由于时间限制，本项目未能深入探索模型在不平衡数据、带噪标签等真实场景下的鲁棒性问题。AdaBoost 在这类情况下可能产生“过度关注异常点”的风险，这也是我未来希望深入研究的方向。

6.3 学习体会

通过本次机器学习课程的学习，尤其是在亲自实现以对数几率回归和决策树桩为基分类器的 AdaBoost 算法的过程中，我对机器学习中“弱学习器提升为强学习器”的基本思想有了深刻的理解。相比于单一模型的构建与调参，AdaBoost 更强调模型之间的协同与迭代优化，这种设计思想让我感受到集成学习的强大威力。

对数几率回归是我在这门课中第一次系统性地学习的一种“概率解释明确、推导严谨”的线性模型。通过对第 5.3 节课件中推导的学习，我理解了逻辑函数（sigmoid）的由来、对数似然的优化目标，以及如何将其引入 AdaBoost 中作为一个可以拟合样本权重的弱学习器。

另一方面，决策树桩虽然模型结构简单，却是集成学习中最常见的弱分类器。它让我体会到“简单模型 + 智能组合”的思路往往胜过单一复杂模型。尤其是结合课件第 12 章对树模型的讲解后，我更理解了其分裂准则、信息增益、剪枝策略等背后的直觉与理论。

此外，在编程实践中我也进一步熟练掌握了 Python 中的 NumPy、sklearn 等工具的使用，这让我更加感受到“工具 + 理论”在机器学习中的双轮驱动。尤其是 sklearn 的弱学习器能以统一接口快速被 AdaBoost 框架调用，这种设计思维也对我未来进行模块化建模有所启发。

机器学习课程就像是揭开“魔法”的旅程，让我明白了很多生活中智能应用背后的奥义。从图像识别、文本分类到推荐系统，这些看似“不可思议”的系统，其实都有一套严谨、数学化的基础。通过本课程，我不仅学会了这些算法的使用方式，更重要的是学会了如何思考、设计和改进这些算法。

我深知这次大作业只是我迈入机器学习世界的一小步。今后，我希望能不断深入理论，提升工程能力，真正实现将知识转化为现实中“可落地”的人工智能系统。

6.4 建议

从学生的角度出发，我对本门课程提出以下几点建议，以期对课程未来的优化与改进有所帮助：

- **增加项目选题参考范围：**本次大作业题目固定，但由于同学们之间对于机器学习的认识程度不同，仅仅考虑两种基分类器实现方法可能限制了一定发挥。如果能在底层逻辑基础上，提供多个可选项目方向，例如“用 XBoost 算法改进 AdaBoost 算法”、“选择其他基分类器算法提高 AdaBoost 算法能力”等，或者给予自选任务加分项，将有助于激发同学们的探索热情。

- **适当增加项目引导过程：**建议在项目中期安排一次汇报/讨论课，或设立 office hour 供学生就具体模型设计或实现细节提问交流。这样既能防止部分同学陷入错误方向，又能增强课堂的互动性和参与感。

- **补充更多工程实践内容：**课程主要讲授了机器学习算法的原理，但实际工业应用中还涉及如模型评估、调参、部署等一系列工程问题。建议在课程末尾安排一节“实践专题”内容，介绍如 cross-validation、pipeline 构建、超参数搜索等内容，将理论与工程更加紧密结合。

- **强化数学与代码的连接：**对于像 Logistic Regression 这种模型，如果课程中能同时展示数学推导与代码实现对应关系，例如推导的每一项在代码中如何体现，会更有助于理解“从公式到代码”的完整过程。

总之，《机器学习》是一门极具挑战性和收获感的课程。它不仅让我学到了数学原理和编程技巧，更重要的是引导我进入了人工智能的真实世界。我期待未来能够在老师的指导下，持续学习、研究并实践这个领域中更广阔的内容。

参考文献

[1] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119–139.

[2] Schapire, R. E. (1999). A brief introduction to boosting. *IJCAI*, 1401–1406.

[3] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.

[4] 周志华. (2021). 机器学习（第 2 版）. 清华大学出版社.

[5] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[6] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

[7] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106.

[8] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[9] Kleinbaum, D. G., & Klein, M. (2010). *Logistic Regression: A Self-Learning Text* (3rd ed.). Springer.

[10] Beygelzimer, A., Langford, J., & Ravikumar, P. (2009). Error-correcting tournaments. *International Conference on Algorithmic Learning Theory*, 247–262.

[11] Storn, R., & Price, K. (1997). Differential evolution –A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341–359.

[12] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD* (pp. 785–794).

[13] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

[14] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123–140.

[15] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems* (pp. 1–15). Springer.

A 附录 I

A.1 问题回答

1. 对 AdaBoost 算法的认识

AdaBoost 是一种强大的集成学习算法，通过将多个弱分类器组合成一个强分类器来提高模型的准确性。通过对分类器赋予不同的权重，AdaBoost 能够有选择地关注那些容易被分类器错误分类的样本，进而提高模型的性能。通过查阅资料，他有几个很好的特性：

- **自适应性：** AdaBoost 在训练过程中自适应地调整分类器的权重。通过增加对分类困难样本的权重，AdaBoost 能够逐步提高模型对这些困难样本的识别能力。这种自适应性使得 AdaBoost 在面对复杂数据时表现优异，能够有效处理分类难度较大的样本。
- **敏感性：**尽管 AdaBoost 在大多数情况下效果显著，但在面对噪声数据或异常值时，AdaBoost 可能会过拟合。随着训练的进行，分类错误的样本会被赋予更大的权重，这可能导致模型更加关注异常数据。因此，在数据预处理和清洗上需要小心，以避免数据中的噪声对模型性能的负面影响。
- **叠加性：** AdaBoost 通过逐步训练多个弱分类器，并根据每个分类器的性能分配权重，最终将它们组合成一个强分类器。随着基分类器数量的增加，AdaBoost 的表现逐渐得到改善，注意到，过多的基分类器可能会导致计算开销增加，同时可能发生过拟合现象。因此，需要在模型复杂度和计算效率之间找到平衡。
- **弱解释性：**虽然 AdaBoost 是一个强大的模型，但由于其基于多个弱分类器的组合，其最终模型的解释性较差。整体上，AdaBoost 属于“黑箱”模型。

2. 基分类器类型、超参数设置对最终模型性能的影响

AdaBoost 的性能受到基分类器类型、超参数设置以及数据本身的高度影响。以下是我对基分类器和超参数设置影响的发现：

- **基分类器类型：**
  - **决策树桩与逻辑回归比较**决策树桩作为弱分类器，通常表现较差，但是当与 AdaBoost 结合时，由于 AdaBoost 的迭代加权机制，弱分类器的性能得以大幅提升。基于决策树桩的 AdaBoost 通常能够在大量分类器组合的情况下取得较好的效果。逻辑回归作为基分类器时，表现一般。虽然它可以很好地进行概率估计，但由于其线性性质，在面对复杂的非线性数据时表现不佳。通

过 AdaBoost 加强弱分类器的能力，逻辑回归的性能得到了一定提升，但提升相当有限，面对高度非线性问题时受到限制。

- **其他基分类器：** AdaBoost 也可以与其他类型的基分类器（如支持向量机、K 近邻等）结合。如果基分类器本身的表现较好，那么集成后的模型将获得更好的效果。

- **超参数设置：**
  - **学习率：** 学习率对 AdaBoost 的影响较大。较小的学习率会使得训练过程更加稳定，但可能需要更多的基分类器才能达到较好的效果。较大的学习率虽然可以加速收敛，但可能导致模型过拟合或不稳定。学习率的调整需要结合数据和模型的实际表现进行选择。
  - **基分类器数量：** 基分类器的数量是影响 AdaBoost 模型性能的重要超参数。增加基分类器的数量通常会提高模型的性能，尤其是在面对复杂数据时。但过多的基分类器可能导致计算开销增加，并且在某些情况下可能会导致过拟合。因此，基分类器的数量需要根据具体问题进行合理选择。

A.2 文件夹内容说明

AdaBoost_Project.txt

包括结构说明

data.csv

测试 data

targets.csv

标签值

data

data 等数据样例

data_record.xlxs

测试三至五数据

demand.pdf and learn.pdf

课程参考要求与参考资料

main.py

主函数

evaluate.py

评估文件，其中包含修改过的日志函数

img

用图

log and eva_log

训练日志和测试日志

log_Before_the_final_version_of_thecode and eva_log_Before_the_final_version_of_

debug 用的日志

base_model

基分类器文件

adaboost

AdaBoost 算法实现

A.3 Python 库使用

代码提供了 cpu 和 gpu 版本，对于 cpu 版本，只需要 numpy 和 pandas 包，对于 gpu 版本需要额外 gpu 版本下的 pytorch 包，xgboost 包和 sklearn 包。实验 1,2 全部使用 cpu 版本，实验 3,4，5, 由于需要多次测试，使用 gpu 版本加速

A.4 算法处理说明

为了在不同数据上有更好的鲁棒性，已经设置为实验得到的最好学习率和最大迭代数。对于逻辑回归, 设置学习率为 0.0707, 最大迭代数 850; 对于决策树桩, 设置学习率为 1.01048。可以尝试调整二者获得更多结果(逻辑回归的调整/base_model/logistic_regression.py, 决策树桩调整在 adaboost/adaboost.py)