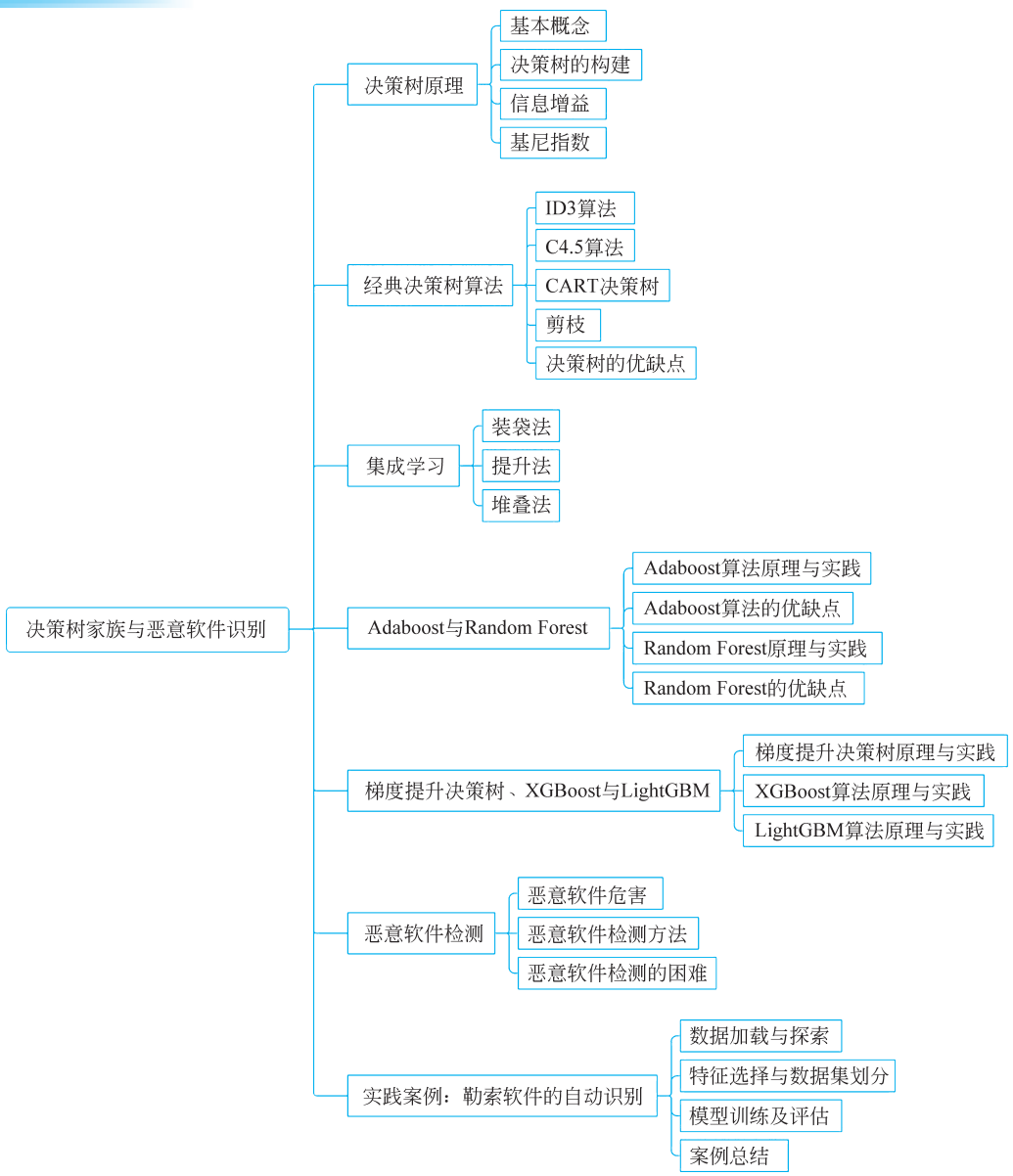
第 3 章　决策树家族与恶意软件识别

本章思维导图



学习目标

- 理解决策树的基本概念与构建过程，掌握信息增益与基尼指数等核心理论。
- 掌握经典决策树算法（如 ID3、C4.5、CART）的原理。
- 了解决策树剪枝方法，理解其在优化决策树性能中的作用。
- 掌握 Adaboost 与 Random Forest 算法的原理、实现、优化策略以及优缺点。
- 了解梯度提升树、XGBoost、LightGBM 等前沿方法，掌握其核心算法原理并编程实现。
- 理解恶意软件的原理、危害及识别方法，并通过实践案例学习自动识别勒索软件的实际应用。

决策树家族的发展经历了从单一决策树到集成学习的演变（图 3.1）。最早的决策树算法包括 ID3、C4.5 和 CART，它们用于构建树状结构以进行分类或回归。随后，集成学习方法引入了 Adaboost，它采用 Boosting 算法，重点关注分类错误的样本，以提高模型性能。随机森林通过 Bagging 算法随机选择特征并训练多个决策树，从而增强泛化能力。

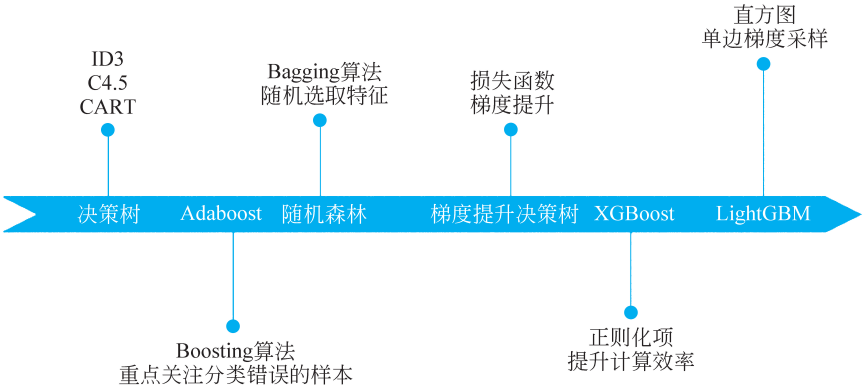


图 3.1　决策树家族发展史

在此基础上，梯度提升决策树（GBDT）进一步优化了损失函数，使模型在每一步都能通过梯度提升减小误差。XGBoost 通过正则化项来提高计算效率并降低过拟合风险，而 LightGBM 采用直方图算法和单边梯度采样，进一步优化了计算效率，使其在大规模数据集上表现更优。

3.1　决策树原理

决策树是一种简单而有效的机器学习算法，广泛应用于分类和回归任务。决策树的思想源于人类的决策过程，在已知各种情况发生概率的前提下，通过构建树状结构来评估风险和判断其可行性。这种方法利用概率分析的图解方式，以树状图的形式直观地展示决策过程，因此得名“决策树”。决策树通过递归地分裂数据集，将数据划分为更小的子集，以便能够更好地预测目标变量。其主要思想是通过一系列的判断条件，将数据集分成不同的分支，最终形成一个树状结构，其中每个叶节点代表一个最终的分类或回归结果。

3.1.1 基本概念

决策树是一种树状结构的模型，如图 3.2 所示，它由根节点、内部节点和叶节点组成，其中，根节点和内部节点用于做出决策，并且能够拥有多个分支，而叶节点则是这些决策的最终输出，不能继续分裂出新的子树。

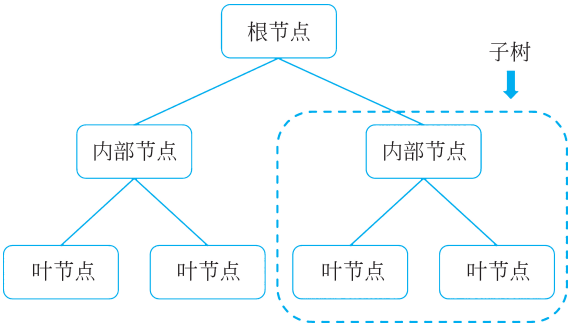


图 3.2 决策树模型

(1) 根节点(root node)：根节点是决策树的起点，包含整个数据集，同时代表树状结构的最高层次。在构建决策树的过程中，根节点通过选择一个最优的特征(或属性)进行第一次划分，将数据集分成若干个子集。

(2) 内部节点(internal node)：内部节点位于根节点和叶节点之间，每个节点代表一次决策或判断。每个内部节点使用一个特征及其相应的阈值或分类标准来分裂数据，进而向下传递数据到更细的子节点。

(3) 叶节点(leaf node)：叶节点是决策树的终点，不再进行进一步的划分。每个叶节点对应一个类别标签(在分类问题中)或一个连续值(在回归问题中)。

(4) 子树(sub tree)：子树是决策树中的一部分，包含一个内部节点及其所有的子节点和分支。子树本身也是一棵决策树，可以看作一棵较大树中的子结构。

(5) 分支(branch)：分支是决策树中的路径，连接节点之间的通道。在决策树中，数据从根节点开始，通过一系列的分支(基于特征的判断条件)传递到最终的叶节点，而每个分支则代表一个特定的条件或规则的满足。

(6) 父/子节点(parent/child node)：如果一个节点直接连接到另一个节点并从它分支出来，那么这个节点就是后者的父节点，而父节点分裂产生的节点则称为子节点。

(7) 深度(depth)：树的深度指从根节点到最深的叶节点所经过的边的数量。

此外，可以将决策树近似为一个 if-then 规则的集合。具体而言，从根节点到叶节点的每一条路径都可以看作一个 if-then 决策规则。在每个内部节点中，分支条件都可以用“如果(if)某个特征的值满足特定条件，则(then)沿着某个分支继续”来描述。例如，在决策树的某个节点上，如果特征值小于某个阈值，那么数据将沿着该条路径继续向下分支，而这一路径的每一步都是一个具体的 if-then 判断。这种 if-then 规则的结构能够清晰地展示决策树的决策过程，使决策树的决策逻辑变得直观且易于理解。

3.1.2 决策树的构建

基本构建方法是一个自顶向下的递归过程，从数据集中选择一个分裂属性创建根节点，

并按属性取值划分数据集，为每个子集创建新的节点。通过对每个子节点重复上述步骤直到满足停止条件，最终使叶节点表示决策结果。为便于读者理解，下面将通过一个具体的天气数据集实例详细说明决策树的构建。

假设分析目标是预测某天是否适合出门玩耍，其相应的样本属性包括天气、温度、湿度以及风速。带标签的数据如表 3.1 所示。

表 3.1　天气数据集

序　号	天　气	温　度	湿　度	风　速	是否适合出门玩耍
1	晴天	炎热	高	弱	否
2	晴天	炎热	高	强	否
3	阴天	炎热	高	弱	是
4	雨天	温暖	高	弱	是
5	雨天	寒冷	正常	弱	是
6	雨天	寒冷	正常	强	否
7	阴天	寒冷	正常	强	是
8	晴天	温暖	高	弱	否
9	晴天	寒冷	正常	弱	是
10	雨天	温暖	正常	弱	是
11	晴天	温暖	正常	强	是
12	阴天	温暖	高	强	是
13	阴天	炎热	正常	弱	是
14	雨天	温暖	高	强	否

决策树的构建首先需要选择一个属性分裂数据集，通过这个属性将原始的数据集分裂成若干个子集。属性选择是构建决策树的第一步，也是最关键的一步，因为需要找到能够最好地分离数据的特征。此外，属性选择也会直接影响决策树的整体形状以及最终的分类结果。

属性选择的常用方法主要包括信息增益、基尼指数等，这将会在 3.1.3 节进行详细介绍。在上述数据集中，可以选择将“天气”作为分裂属性来创建决策树的根节点，将其作为第一个分裂属性的原因在于它可以将数据集分成较为均衡的 3 个子集，从而最大化区分度。

在选定分裂属性后，将会根据分裂属性的不同值将数据集划分成多个子集并创建新的子节点。在选择将天气作为分裂属性后，将得到 3 个不同的子集“晴天”“阴天”“雨天”。在此之后，可以在不同的子集上重新选择新的分裂属性，对子集进行划分。例如，在“晴天”子集中，可以选择“湿度”作为下一个分裂属性，如果湿度高，则“不出门玩耍”，否则“出门玩耍”；在“雨天”子集中，可以选择“风速”作为下一个分裂属性。通过重复选取分裂属性和划分子集这两个步骤，最终构建出一棵完整的决策树，如图 3.3 所示。

为防止决策树过度生长并避免过拟合现象的出现，在决策树的构建过程中需要设置合理的停止条件。常见的停止条件如下。

(1) 数据集被完全纯化：某个节点中的所有样本属于同一类别。在这种情况下，继续

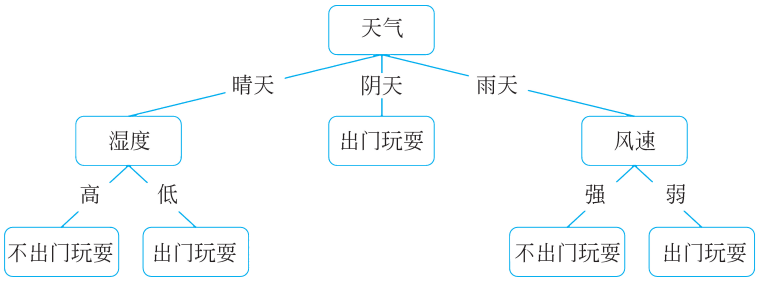


图 3.3　是否出门决策树

分裂已经没有意义，因为不会带来任何新的信息。例如，在“阴天”子集中，由于所有样本的结果都是“出门玩要”，因此在“阴天”这个子集上不需要继续分裂，这是一个典型的纯化数据集停止条件。

（2）没有更多的属性可供选择：所有可能的属性都已被用来分裂数据集，或者当前节点的所有样本在剩余属性上具有相同的值。此时，继续分裂也无法产生新的子节点，决策树应在此节点停止生长。

（3）样本数量小于预设的阈值：在某些情况下，节点中的样本数量可能过少，继续分裂可能导致过拟合。为避免这种情况，可以设置一个最小样本数阈值，当节点中的样本数量低于该阈值时，停止进一步分裂。

为进一步提高决策树的泛化能力，在决策树生成之后，可以通过剪枝去除决策树上的一些冗余分支，使决策树模型整体更为简洁。剪枝过程中通常可以借助交叉验证或独立的验证集验证剪枝效果，避免过度拟合已有数据。

3.1.3　信息增益

假设有两种数据，分别用正方形和三角形表示，其分布如图 3.4 所示。可以看到，图中有 4 个正方形点和 4 个三角形点。如果以 $x=3$ 为分界线对其进行分裂，如图 3.5 所示，可以观察到这个分裂将正方形点与三角形点完美地分裂成了两个相互独立的子区域，左侧包含 4 个正方形点，右侧包含 4 个三角形点。

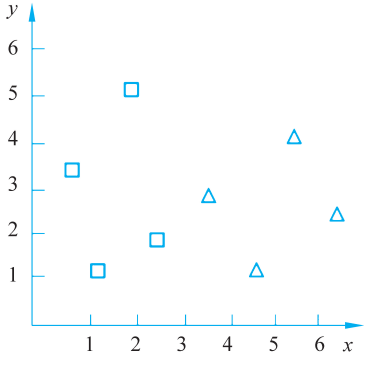


图 3.4　数据点分布图

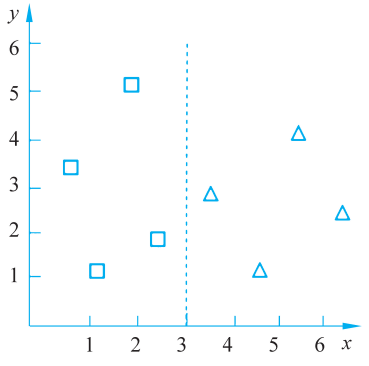


图 3.5　数据点完美分裂图

但是，如果以 $x=4$ 为分界线进行分裂，如图 3.6 所示，可以观察到左侧包含 4 个正方形点和 1 个三角形点，右侧包含 3 个三角形点。这种分裂方式显然是不合理的，那么应该如何

对分裂质量进行量化呢？

信息增益是一种在决策树构建中广泛使用的指标，它基于熵的概念来量化每个特征在数据集分裂中的贡献，从而确定最优的分裂特征。通过计算信息增益，可以衡量某个特征在将数据集分裂成子集后不确定性减少的程度，从而确定哪个特征在分裂数据集时提供了最多的信息。当选择一个特征进行分裂时，如果分裂后的子集熵值显著降低，则说明该特征在区分数据方面非常有效，因此信息增益较大。信息增益越大，说明使用该特征进行划分后，数据集的纯度提高得越多。

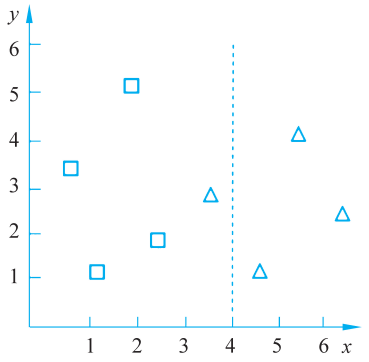


图 3.6　数据点不完美分裂图

假定选择一个特征 A 来分裂数据集 D，其中特征 A 有 v 个可能的取值 $\{a_1,a_2,\cdots,a_v\}$。根据特征 A 的不同取值，可以将数据集 D 划分为 v 个子集 $\{D_1,D_2,\cdots,D_v\}$。每个子集 D_i 中的样本数占原数据集 D 的比例为 $|D_i|/|D|$，其中 $|D_i|$ 代表子集 D_i 的样本数量。

那么，数据集 D 的信息熵 $H(D)$ 为

$$H(D)=-\sum_{i=1}^{n} p_i \log_2 p_i \tag{3.1}$$

其中，n 为数据集中的类别数，p_i 为第 i 类样本的比例。

条件熵 $H(D\mid A)$ 为

$$H(D\mid A)=\sum_{i=1}^{v}\frac{|D_i|}{|D|}H(D_i) \tag{3.2}$$

信息增益 $\mathrm{IG}(D,A)$ 定义为使用特征 A 分裂数据集 D 前后信息熵的差值：

$$\mathrm{IG}(D,A)=H(D)-H(D\mid A) \tag{3.3}$$

由于信息增益反映了某个特征在将数据集分裂成子集后减少了多少不确定性，因此信息增益越大，说明该特征在分裂数据集时带来的纯度提升就越大，那么在构建决策树时，将会倾向于选择信息增益大的特征进行分裂。

以表 3.1 为例，该数据集包含 14 个训练样本，用来学习一棵能预测是否适合出门玩耍的决策树。显然，在这棵决策树中，类别数 n 为 2。在决策树开始学习时，根节点包含 D 中的所有样本，其中正例占 $p_1=\frac{9}{14}$，反例占 $p_2=\frac{5}{14}$。于是，根据公式可以计算出根节点的信息熵为

$$H(D)=-\left(\frac{9}{14}\log_2\frac{9}{14}+\frac{5}{14}\log_2\frac{5}{14}\right)\approx 0.94$$

然后，分别计算当前特征集合{天气，温度，湿度，风速}中每个属性的信息增益。以特征“天气”为例，按照该特征对数据集进行划分，可以得到 3 个子集，分别记为 D_1（天气＝晴天）、D_2（天气＝阴天）、D_3（天气＝雨天）。子集 D_1 中包含编号为{1,2,8,9,11}的 5 个样本，其中正例占 $p_1=\frac{2}{5}$，反例占 $p_2=\frac{3}{5}$；子集 D_2 中包含编号为{3,7,12,13}的 4 个样本，其中正例占 $p_1=\frac{4}{4}$，反例占 $p_2=\frac{0}{4}$；子集 D_3 中包含编号为{4,5,6,10,14}的 5 个样本，其中

正例占 $p_1=\frac{3}{5}$，反例占 $p_2=\frac{2}{5}$。根据公式可计算出用特征“天气”划分后的 3 个分支的信息熵为

$$H(D_1)=-\left(\frac{3}{5}\log_2\frac{3}{5}+\frac{2}{5}\log_2\frac{2}{5}\right)\approx 0.971$$

$$H(D_2)=-\left(\frac{4}{4}\log_2\frac{4}{4}+\frac{0}{4}\log_2\frac{0}{4}\right)=0$$

$$H(D_3)=-\left(\frac{2}{5}\log_2\frac{2}{5}+\frac{3}{5}\log_2\frac{3}{5}\right)\approx 0.971$$

于是根据公式可计算出特征“天气”的条件熵为

$$H(D\mid \text{天气})=\frac{5}{14}H(D_1)+\frac{4}{14}H(D_2)+\frac{5}{14}H(D_3)\approx 0.693$$

因此特征“天气”的信息增益为

$$\text{IG}(D,\text{天气})=H(D)-H(D\mid \text{天气})\approx 0.247$$

通过这种方式，可以计算出其他特征的信息增益为 $\text{IG}(D,\text{温度})\approx 0.029$，$\text{IG}(D,\text{湿度})\approx 0.152$，$\text{IG}(D,\text{风速})\approx 0.048$。显然，特征“天气”的信息增益最大，因此在构建决策树时首先选用该特征进行划分。

3.1.4 基尼指数

CART 决策树使用了基尼指数选择分裂特征。基尼指数是一种度量数据集纯度的指标，基尼指数越小，数据集的纯度越高。通过计算基尼指数，可以确定哪个特征在分裂数据时能够最有效地提高数据集的纯度，从而实现合理的决策树构建。

假定选择一个特征 A 来分裂数据集 D，其中特征 A 有 v 个可能的取值 $\{a_1,a_2,\cdots,a_v\}$。那么，数据集 D 的纯度可以使用基尼不纯度来度量：

$$\text{Gini}(D)=\sum_{i=1}^{n}\left(p_i\sum_{k\neq i}p_k\right)=\sum_{i=1}^{n}p_i(1-p_i)=\sum_{i=1}^{n}p_i-\sum_{i=1}^{n}p_i^2=1-\sum_{i=1}^{n}p_i^2 \tag{3.4}$$

当数据集中的所有样本属于同一类别时，基尼不纯度为 0，表示数据集纯度最高。当数据集中的样本类别均匀分布时，基尼不纯度最大，表示数据集纯度最低。

根据特征 A 的不同取值，可以将数据集 D 划分为 v 个子集 $\{D_1,D_2,\cdots,D_v\}$，$|D_i|$ 代表子集 D_i 的样本数量。那么，根据特征 A 进行数据集划分时，对应的基尼增益的计算公式为

$$\text{GG}(D,A)=G(D)-\sum_{i=1}^{v}\frac{|D_i|}{|D|}\text{Gini}(D_i) \tag{3.5}$$

由于在选择最佳分裂特征时面对的是同一个数据集，尽管这个数据集可能是最初的完整数据集，也可能是经过分裂后形成的子集。但无论是何种情况，$G(D)$ 都是相同的，因此针对基尼增益的计算公式进行变化，便得到了基尼指数的计算公式为

$$\text{GI}(D,A)=\sum_{i=1}^{v}\frac{|D_i|}{|D|}\text{Gini}(D_i) \tag{3.6}$$

以表 3.1 为例，分别计算当前特征集合{天气，温度，湿度，风速}中每个属性的基尼指数，以此找出最佳分裂特征。以特征“天气”为例，按照该特征对数据集进行划分可以得到 3

个子集，分别记为 D_1（天气＝晴天）、D_2（天气＝阴天）、D_3（天气＝雨天）。按照式(3.4)分别计算 D_1、D_2 和 D_3 的基尼不纯度为

$$\mathrm{Gini}(D_1)=1-\left(\left(\frac{3}{5}\right)^2+\left(\frac{2}{5}\right)^2\right)=0.48$$

$$\mathrm{Gini}(D_2)=1-\left(\left(\frac{4}{4}\right)^2+\left(\frac{0}{4}\right)^2\right)=0$$

$$\mathrm{Gini}(D_3)=1-\left(\left(\frac{2}{5}\right)^2+\left(\frac{3}{5}\right)^2\right)=0.48$$

因此特征“天气”的基尼指数为

$$\mathrm{GI}(D,\text{天气})=\frac{5}{14}\mathrm{Gini}(D_1)+\frac{4}{14}\mathrm{Gini}(D_2)+\frac{5}{14}\mathrm{Gini}(D_3)\approx 0.343$$

通过这种方式，可以计算出其他特征的基尼指数为 $\mathrm{GI}(D,\text{温度})\approx 0.440$，$\mathrm{GI}(D,\text{湿度})\approx 0.367$，$\mathrm{GI}(D,\text{风速})\approx 0.428$。显然，特征“天气”的基尼指数最小，因此在构建决策树时首先选用该特征进行划分。

3.2　经典决策树算法

经典决策树算法包括 ID3、C4.5、CART 等，通过引入不同的分裂标准和剪枝技术来提高模型的准确性和鲁棒性，同时每种算法都有其独特的优点和应用场景。ID3 算法使用信息增益选择最优分裂属性，适用于处理离散数据；C4.5 算法改进了 ID3 算法，引入了增益率来处理连续数据，并加入了剪枝技术以防止过拟合；CART 算法使用基尼指数指导分裂属性的选择，在集成学习算法中，经常选择使用 CART 决策树。

3.2.1　ID3 算法

ID3(Iterative Dichotomiser 3)算法是由 Ross Quinlan 于 1986 年提出的一种决策树构建算法，是一种基于贪心策略的算法，其基本思想是通过选择最能提高数据分类准确性的特征来构建决策树。ID3 算法的步骤如下。

(1) 对当前样本集合，计算所有特征的信息增益。

(2) 选择信息增益最大的特征作为当前节点的分裂特征，并将数据集分裂成多个子集。

(3) 对每个子集递归应用 ID3 算法，直到满足递归停止条件(①没有划分特征可供继续划分；②给定的分支的数据集为空；③数据集属于同一类；④决策树达到设置的最大值)。

ID3 算法凭借直观的思想和简洁的实现而得到广泛应用，但也存在明显的不足。

(1) 偏好取值较多的特征：算法倾向于选择取值较多的特征进行分裂，即选择那些信息增益较大的特征，但这些特征未必具备良好的分类能力。这种特征选择偏向可能引起模型的过拟合，降低了对新数据的泛化能力。

(2) 无法处理连续特征：只能处理离散特征，对于连续特征，在分类前需要对其进行离散化。

(3) 无法处理缺失值：不能处理属性具有缺失值的样本。

(4) 容易过拟合：没有采用剪枝技术，决策树的结构可能过于复杂，导致模型过拟合。

为克服这些不足，后续的算法（如 C4.5 和 C5.0）进行了相应的改进。

3.2.2 C4.5 算法

Ross Quinlan 于 1993 年改进了 ID3 算法，在信息增益的基础上引入了多个优化策略，称为 C4.5 算法。算法的步骤如下。

（1）与 ID3 算法不同，C4.5 算法使用信息增益率（Gain Ratio）选择分裂特征，信息增益率通过对信息增益进行归一化处理，缓解了 ID3 算法中信息增益偏向于具有更多取值的属性的问题。信息增益率的定义如下：

$$\text{GainRatio}(A)=\frac{\text{IG}(D,A)}{\text{SplitInfo}(D,A)} \tag{3.7}$$

其中，$\text{SplitInfo}(D,A)$ 表示特征分裂所产生的信息量，计算公式如下：

$$\text{SplitInfo}(D,A)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\times\log_2\frac{|D_i|}{|D|} \tag{3.8}$$

其中，$|D|$ 表示数据集中的样本个数，n 表示数据集中特征 A 所具有的类别数量，i 表示数据集中特征 A 具有的类别的编号，$|D_i|$ 表示数据集中特征 A 的值为编号 i 的样本的个数。

（2）在每个节点上，选择信息增益率最高的特征作为分裂特征，并将数据集根据该特征的取值划分为不同的子集。

（3）对每个子集递归地应用 C4.5 算法，直到满足停止条件（①所有样本都属于同一类别；②没有更多的特征可用来分裂；③达到预设的树深度限制）。

C4.5 算法的改进主要如下。

（1）使用信息增益率选择特征。

（2）在决策树的构建过程中进行剪枝操作，缓解过拟合现象的出现。

（3）在构建决策树前不需要对连续特征进行离散化处理，可以直接处理连续特征。

（4）可以处理带有缺失值的样本。

3.2.3 CART 决策树

CART（Classification And Regression Tree）算法是 Breiman 等在 1984 年提出的一种应用广泛的决策树模型，可以用于分类任务（称为 CART 分类决策树）和回归任务（称为 CART 回归决策树）。不同于 ID3 算法以及 C4.5 算法，CART 算法构建的是一种二分决策树，每次对特征进行分裂后只会产生两个分支。而在 ID3 算法和 C4.5 算法中，决策树的分支是根据选定特征的取值决定的，分裂特征有多少种不同的取值，就有多少个分支。

CART 分类决策树的步骤如下。

（1）已知训练数据集为 D，计算数据集中现有的所有特征对该数据集的基尼增益（相关概念于 3.1.4 节介绍）。具体来说，对于每个属性特征 F 的所有可能取值 f，对于 $F=f$ 的测试为“是”或“否”，将数据集 D 划分为 $D_{是}$ 和 $D_{否}$ 两个子数据集，并计算 $F=f$ 时的基尼增益。

（2）在所有的特征 F 及其全部的可能取值 f 中，选择基尼增益最大的特征及其对应的取值 f 作为最优分裂点，按照该分裂点生成两个分支以及子节点，其中左子节点为 $D_{是}$，右子节点为 $D_{否}$。

（3）对两个子节点分别循环步骤（1）和步骤（2），直到满足停止条件。

3.2.4　剪枝

Breiman 等在 1984 年提出了一种方法，通过删除对泛化能力没有影响的子分支，将过拟合的复杂决策树裁剪成准确但精简的决策树。

剪枝过程主要可以分为以下两种。

1）预剪枝

在决策树构建过程中，通过提前停止子树分裂来降低决策树整体的复杂性。具体而言，在划分节点时会根据停止条件（如最大树深度、最小信息增益或最小样本数等）来判断是否继续创建子分支。如果满足停止条件，则停止在当前节点创建子分支，并将当前节点设置为叶节点。

预剪枝（pre-pruning）能够有效地控制决策树的整体规模，避免生成过于复杂的模型，从而降低出现过拟合现象的可能性，并在一定程度上降低计算复杂度，提升创建决策树的效率。然而，这一操作也存在着一定的局限性，由于依赖停止条件进行判断，因此可能会导致决策树在构建过程中忽略某些潜在的重要特征组合，从而影响决策树模型的性能。

2）后剪枝

与预剪枝不同，后剪枝（post-pruning）是在决策树构建完成后进行的，根据某些标准（如验证集的错误率、最小误差杂度等）利用叶节点代替内部节点和子树，删除那些对模型泛化能力贡献较小的分支。

后剪枝能够保留决策树构建过程中的所有潜在信息，通过对完整的决策树进行优化获得效果更好的模型，同时能更加灵活地调整决策树的大小和复杂性，使得模型更好地适应数据。然而，由于后剪枝需要先构建完整的决策树，这一过程可能涉及大量计算，会消耗更多的时间和资源。尽管后剪枝有助于缓解过拟合现象，但完全依赖后剪枝并不能彻底消除过拟合。

常见的后剪枝方法有代价复杂度剪枝（Cost-Complexity Pruning，CCP）、错误率降低剪枝（Reduced Error Pruning，REP）、悲观错误剪枝（Pessimistic Error Pruning，PEP）、最小错误剪枝（Minimum Error Pruning，MEP）等。

代价复杂度剪枝在 Breiman 的 CART 决策树算法中首次提出并使用，旨在通过逐步修剪决策树的子树来降低模型的复杂度，在尽可能保留模型性能的同时避免过拟合现象的发生。该剪枝方法为决策树定义了代价（cost）和复杂度（complexity），以及一个衡量代价与复杂度之间关系的参数 α。其中，代价指在剪枝过程中因子树 T_t 被叶节点替代而增加的错分样本，复杂度指剪枝子树 T_t 减少的叶节点数。参数 α 的定义如下：

$$\alpha = \frac{R(t) - R(T_t)}{|N_{T_t}| - 1} \tag{3.9}$$

$$R(t) = \sum_{t \in T} p(t) r(t) \tag{3.10}$$

$$p(t) = \frac{n(t)}{n} \tag{3.11}$$

$$r(t) = 1 - \max_k p(C_k - t) \tag{3.12}$$

其中，$|N_{T_t}|$ 代表子树 T_t 中的叶节点数，$R(t)$ 代表节点 t 错误分类的误差之和，$R(T_t)$ 代

表剪枝形成的子树 T_t 中每个叶节点所产生的错误分类的误差之和，$n(t)$ 代表节点 t 处理的样本数量，n 代表总的样本数量，$r(t)$ 代表节点 t 错误分类的比例。

代价复杂度剪枝的主要步骤如下。

(1) 对于完全决策树 T 的每个内部节点计算 α 值，循环剪掉具有最小 α 值的子树，直到剩下根节点。从这个过程中得到一系列的剪枝树 $\{T_0, T_1, \cdots, T_r\}$，其中，$T_0$ 为原有的完全决策树，T_r 为根结点。

(2) 根据交叉验证的结果，选择在验证集上表现最好的 α 值对应的剪枝子树，确保这个剪枝树在平衡训练误差和模型复杂度方面能够达到最优的效果。

3.2.5 决策树的优缺点

决策树作为一种广泛使用的算法，凭借简洁直观的结构和强大的解释能力而备受欢迎。这种算法不仅易于理解和实现，而且能直观地反映数据的特征和模式。然而，决策树并非完美无缺，在应用过程中仍然存在许多局限性。

决策树的优点如下。

(1) 具有一定的可解释性：决策树的结构直观、清晰，与人类的决策过程十分相似，每个内部节点都代表一个特征的决策，而叶节点则代表最终的预测结果。这种结构不仅易于理解，而且能够对树的结构进行可视化展示。

(2) 能处理多种数据类型：决策树能够处理数值型和类别型数据，并且在需要时可以自动对连续数据进行离散化处理，这使得决策树具有广泛的适用性。

(3) 无须过多的数据预处理：在构建决策树模型时，通常不需要对数据进行复杂的预处理，如特征缩放或标准化，同时决策树也能够处理数据集中存在的缺失值。

决策树的缺点如下。

(1) 过拟合：容易在训练过程中出现过拟合，尤其是当树的深度较大且没有适当剪枝时，这将导致模型尽管在训练集上表现良好，但在测试集上的性能却显著下降。

(2) 不稳定：对数据中的细微变化非常敏感，即使是数据集的微小变化，也可能导致树的结构发生显著改变。

(3) 对特征选择敏感：决策树算法在选择特征时具有一定的主观性，不同的特征可能对分类结果产生不同的影响，这可能导致决策树的结果不稳定。

3.3 集成学习

集成学习就是集成多个弱监督模型以期得到一个更好、更全面的强监督模型，以提高整体的预测性能和泛化能力。在处理复杂数据集时，集成学习训练出的模型往往比单一弱监督模型表现得更加出色，因为它们能够有效降低弱监督模型的偏差和方差。

常见的集成学习方法包括装袋法(Bagging)、提升法(Boosting)、堆叠法(Stacking)。装袋法通过在训练数据上构建多个模型，并对其结果进行平均，从而减少模型的方差；提升法则通过逐步调整错误预测样本的权重，迭代地提高模型的准确性；堆叠法结合了多种基础模型的预测，通过学习一个新的模型来融合这些预测结果，以提高整体性能。

3.3.1　装袋法

装袋法最初由 Leo Breiman 于 1996 年提出，它采用自助抽样（Bootstrapping），通过多轮有放回的随机抽样来生成多个训练数据，并训练出多个弱学习器（预测结果仅比随机分类好一点的分类器）。这些弱学习器之间不存在强依赖关系，将这些弱学习器的预测结果进行汇总，在一定程度上能够提高整体模型的性能。尽管装袋法是以有放回的自助抽样为基础的，但在多轮重复后，数据集中的样本仍然可能存在尚未被抽样的，每个样本未被抽样的概率为 $(1-1/N)^N$。

装袋法遵循以下步骤（图 3.7）。

（1）数据抽样：从原始数据集中随机抽取多个样本子集。每个子集都是通过有放回的抽样方法（Bootstrap 抽样）生成的，即每个样本可能被多次抽取，也可能没有被抽取。

（2）训练弱学习器：对每个抽样子集，训练一个相同类型的弱学习器（如决策树），由于训练数据集的不同，每个弱学习器在学习过程中的表现也会有所不同。

（3）汇总结果：对于分类问题，使用投票机制来决定最终的预测结果。即对所有弱学习器的预测结果进行统计，选择预测次数最多的类别作为最终结果。而对于回归问题，计算所有弱学习器预测结果的平均值作为最终预测值。

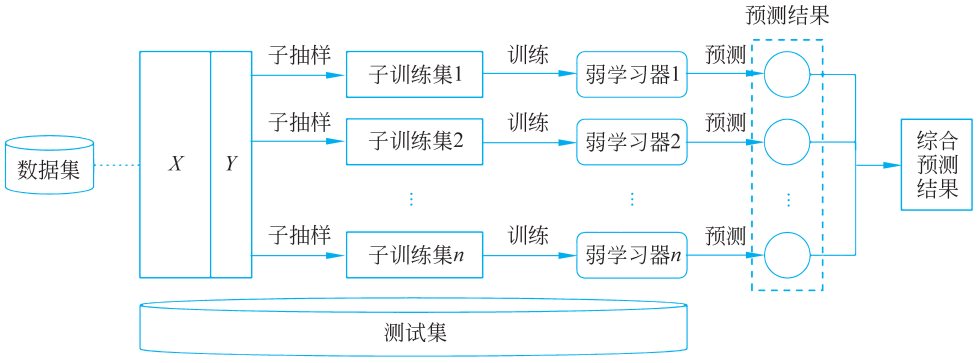


图 3.7　装袋法流程示意

装袋法的主要优势如下。

（1）减少方差：通过对多个弱学习器的预测结果进行汇总，能够显著减少模型的方差，使得最终模型在面对新的数据时更稳定。

（2）降低过拟合：由于每个弱学习器仅在部分数据上进行训练，因此可以有效降低模型的过拟合风险，提升模型的泛化能力。

（3）并行化训练：由于每个弱学习器的训练过程都是相互独立，因此每个弱学习器都可以在集成前并行训练。

装袋法面临的不足如下。

（1）计算开销大：需要训练多个弱学习器，尤其是在数据集和模型规模较大时，计算开销较大。

（2）可解释性较差：集成了多个弱学习器，其预测过程和结果往往难以直观理解，模型的可解释性不如单一模型。

（3）可能增大偏差：由于需要对数据进行有放回的随机抽样，生成多个不同的训练子集，因此如果数据集本身存在显著的偏差，那么这些偏差可能会在抽样过程中被进一步放大。

3.3.2 提升法

Kearns 和 Valiant 于 1989 年首次提出弱学习器与强学习器学习能力等价的猜想，随后 Robert Schapire 于 1990 年通过构造多项式的方法对这一猜想给出了肯定的回答，这也是最早的提升法。在提升法中，通过选择随机数据样本逐步构建并训练一系列的弱学习器，同时每个新学习器都试图修正前一个弱学习器的错误。通过这种逐步改进的方式，提升法能够有效地降低模型的偏差，提升预测精度。

提升法遵循以下步骤（图 3.8）。

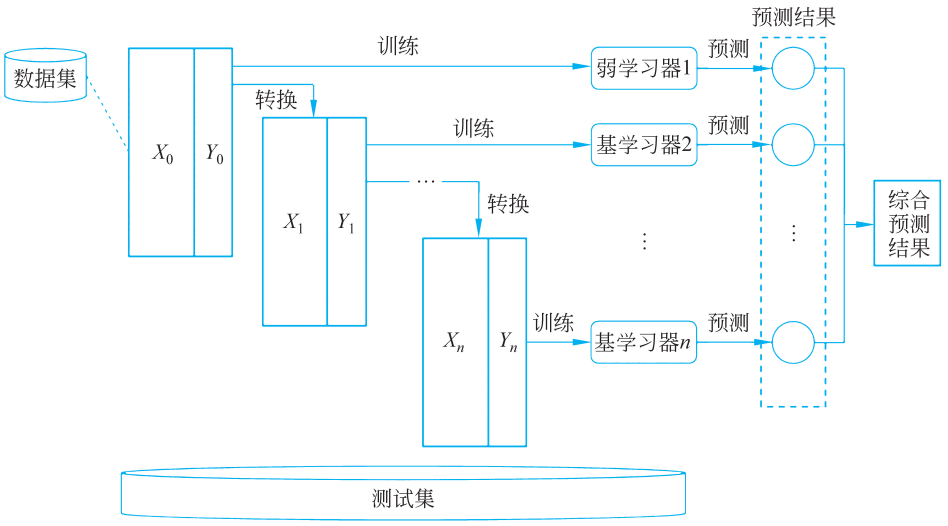


图 3.8 提升法流程示意

（1）初始化权重：给所有训练样本分配相等的初始权重，这些权重在每轮迭代训练中会根据样本被正确或错误地分类的情况进行调整。

（2）训练弱学习器：使用当前权重分布下的训练数据训练一个新的弱学习器。

（3）更新权重：根据弱学习器的表现，对训练样本的权重进行调整。具体来说，对于被正确分类的样本，降低其权重；对于被错误分类的样本，提高其权重。这样做的目的是让后续的弱学习器更加关注那些难以分类的样本。

（4）组合弱学习器：将所有弱学习器按其重要性（通常与其分类误差相关）加权组合起来，形成最终的强学习器。

提升法的主要优势如下。

（1）高准确性：通过组合多个弱学习器，使得最终模型的预测性能显著提升，尤其在处理复杂的分类或回归问题时，通常能取得较高的准确性。

（2）降低偏差：在迭代过程中按顺序训练多个弱学习器，通过修正前一轮学习器的错误，使得最终模型在训练数据上的拟合能力更强，从而降低模型训练过程中的偏差。

提升法面临的不足如下。

(1) 容易过拟合：在迭代过程中不断调整模型，使其对训练数据的拟合程度越来越高，因此如果没有适当的正则化措施，则可能会对噪声数据过度拟合，降低模型的泛化能力。

(2) 计算开销大：在迭代过程中需要多次训练弱学习器，并在每轮迭代中调整权重和更新模型参数，因此训练过程通常较为耗时，尤其在大规模数据集上。

3.3.3　堆叠法

David H. Wolpert 在 1992 年首次提出了堆叠泛化(Stacked Generalization)的概念，也就是堆叠法。作为一种常见的集成学习方法，堆叠法主要通过组合多个基础模型(也称为一级模型)来提升整体的预测性能，这些基础模型可以是不同类型的机器学习算法，也可以是同一种算法的不同超参数组合。与其他集成学习方法不同，堆叠法使用元模型(也称为二级模型)对多个基础模型的输出进行组合，作为新的特征输入元模型，元模型学习这些特征之间的关系来做出最终的预测，以此提高预测的准确性和鲁棒性。

堆叠法主要遵循以下步骤(图 3.9)。

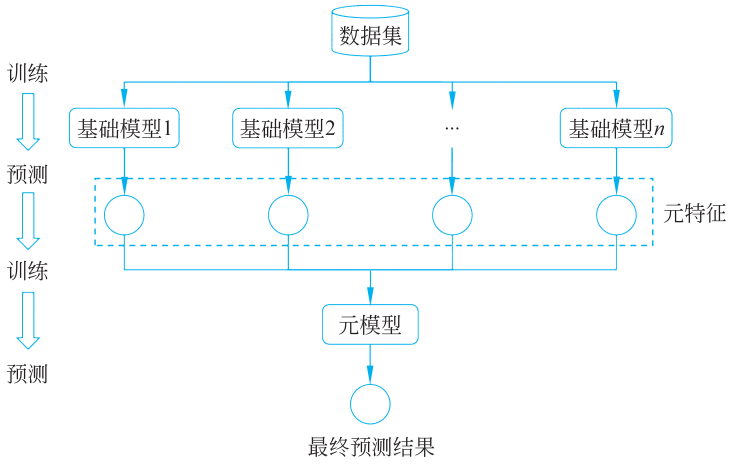


图 3.9　堆叠法流程示意

(1) 训练基础模型：在原始数据集上训练多个不同的基础模型，每个基础模型根据原始特征做出自己的预测。

(2) 生成元特征：将这些基础模型的预测结果作为新的特征，形成一个新的特征集，也称为元特征(Meta-features)，用于训练元模型。

(3) 训练元模型：使用生成的元特征训练一个元模型，该模型通过综合不同基础模型的预测结果来生成最终的预测，其中元模型可以是任何机器学习算法，包括线性回归、逻辑回归、梯度提升树等。

(4) 预测：在新数据上通过基础模型生成元特征，然后利用元模型进行最终的预测。

堆叠法的主要优势如下。

(1) 提高预测性能：堆叠法组合多个基础模型，利用其互补性，提高预测准确性和泛化能力，尤其适用于复杂数据。

(2) 灵活性高：堆叠法可以利用不同模型的优势来处理不同特征类型的数据，确保整

体模型更加灵活、适应性更强。

堆叠法存在的不足如下。

(1) 训练时间较长：堆叠法需要先训练多个基础模型，再训练元模型，计算开销较大，在处理大规模数据时更为显著。

(2) 过拟合：当基础模型过于复杂或训练数据不足时，堆叠法容易过拟合，影响测试集的性能。

装袋法、提升法和堆叠法作为 3 种主要的集成学习策略，在不同的场景中各有其适用性和局限性。选择合适的方法需要考虑数据特点、模型需求、计算资源和调参复杂度，在实际应用中可灵活选择或结合使用。

3.4 Adaboost 与 Random Forest

Adaboost 和 Random Forest 是两种极具代表性的集成算法，广泛应用于分类和回归等任务中，表现出强大的效果。

Adaboost(Adaptive Boosting)是一种 Boosting 算法，它通过多次迭代构建一系列弱分类器(如决策树)，并在每次迭代中根据模型的表现为数据样本赋予不同的权重。每轮训练重点关注之前分类错误的样本，使得新一轮的分类器能够更好地纠正前面的错误。最终，将多个弱分类器的加权组合形成一个强分类器，极大地提升了整体模型的准确性。Adaboost 的灵活性使其能够适应各种数据集，尤其是在噪声较少的场景中表现优异。

Random Forest 是一种基于 Bagging 的算法，通过构建大量的决策树并结合它们的预测结果来提高模型的鲁棒性。每棵树都在随机选择的数据子集和特征子集上进行训练，这种随机性减少了决策树之间的相关性，有效防止了过拟合。Random Forest 的主要优势在于它的稳定性和泛化能力，它在高维度和噪声数据上表现优异，且内置特征重要性评估功能。

3.4.1 Adaboost 算法原理

假设有一个数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$，每个样本 x_i 都包含一个相关的类 $y_i\in\{-1,1\}$，其中 $i\in\{1,2,\cdots,n\}$，此外还有一组弱分类器 $\{C_1(x),C_2(x),\cdots,C_m(x)\}$。在第 $m-1$ 次迭代后，Adaboost 分类器是所有弱分类器的线性组合，其形式为

$$f_{m-1}(x)=\alpha_1C_1(x)+\alpha_2C_2(x)+\cdots+\alpha_{m-1}C_{m-1}(x)=\sum_{j=1}^{m-1}\alpha_j\,C_j(x) \tag{3.13}$$

在第 m 次迭代时，希望通过增加一个弱分类器 $C_m(x)$ 并增加一个对应的权重 α_m，将其扩展成一个更好的分类器：

$$f_m(x)=f_{m-1}(x)+\alpha_m\,C_m(x) \tag{3.14}$$

在整个迭代过程中，希望这个模型在数据集上的经验误差最小，那么第 m 轮迭代的优化目标为

$$\min\sum_{i=1}^{n}L(y_i,f_{m-1}(x)+\alpha_m\,C_m(x)) \tag{3.15}$$

而在训练过程中，Adaboost 的损失函数为指数函数，即

$$\text{Loss}=\sum_{i=1}^{n}\exp(-y_i f_m(x))=\sum_{i=1}^{n}\exp(-y_i(f_{m-1}(x)+\alpha_m C_m(x))) \tag{3.16}$$

而由于 $f_{m-1}(x)$ 与优化变量 α_m 和 $C_m(x)$ 无关，因此令 $\omega_{m,i}=\exp(-y_i f_{m-1}(x))$，在这里 $y_i(f_{m-1}(x))$ 相当于已知的，此时损失为

$$\text{Loss}=\sum_{i=1}^{n}\omega_{m,i}\exp(-y_i\alpha_m H_m(x)) \tag{3.17}$$

为求解最优的分类器 $C_m^*(x)$，对式(3.17)求解最小值可以得到

$$C_m^*(x)=\arg\min_{H}\sum_{i=1}^{m}\omega_{m,i}I(y_i\neq C_m(x_i)) \tag{3.18}$$

其中，$I(\cdot)$ 为指示函数，错误分类时为 1，否则为 0。

此时将所求的 $C_m^*(x)$ 代入式(3.18)，对 α 求偏导并使偏导为 0，可以得到

$$\alpha_m^*=\frac{1}{2}\log\frac{1-e_m}{e_m} \tag{3.19}$$

其中，e_m 为分类器的误差率，即

$$e_m=\frac{\sum_{i=1}^{m}\omega_{m,i}I(y_i\neq C_m(x_i))}{\sum_{i=1}^{m}\omega_{m,i}}=\sum_{i=1}^{m}\omega_{m,i}I(y_i\neq C_m(x_i)) \tag{3.20}$$

而对于每轮的样本权重更新，根据式(3.16)以及 $\omega_{m,i}=\exp(-y_i f_{m-1}(x))$ 可得样本权重更新为

$$\omega_{m+1,i}=\omega_{m,i}\exp(-y_i\alpha_m C_m(x)) \tag{3.21}$$

此时，最终的强分类器为

$$H(x)=\text{sign}(f(x))=\text{sign}\left(\sum_{j=1}^{m}\alpha_j C_j(x)\right) \tag{3.22}$$

Adaboost 算法的伪代码如下。

输入：数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$，$y_i\in\{-1,1\}$，一组弱分类器 $\{C_1(x),C_2(x),\cdots,C_m(x)\}$

输出：最终的强分类器 $f(x)$

过程：

1. 初始化样本集权重：

$$D(1)=(\omega_{1,1},\omega_{1,2},\cdots,\omega_{1,n}),\omega_{1,i}=\frac{1}{n},i=1,2,\cdots,n$$

2. 对于 $j=1,2,\cdots,m$：

(1) 使用具有权重 $D(j)$ 的样本集进行训练，得到弱分类器 $C_j(x):X\to\{-1,1\}$

(2) 计算弱分类器的分类误差率：

$$e_j=P(C_j(x_i)\neq y_i)=\sum_{i=1}^{n}\omega_{j,i}I(y_i\neq C_m(x_i))$$

(3) 计算弱分类器的系数：

$$\alpha_j=\frac{1}{2}\log\frac{1-e_j}{e_j}$$

(4) 更新样本集的权重分布：

$$\omega_{j+1,i}=\frac{\omega_{j,i}}{Z_j}\exp(-y_i\alpha_j C_j(x))$$

其中 Z_j 为规范化因子：

$$Z_j = \sum_{i=1}^{n} \omega_{j,i} \exp(-y_i \alpha_j C_j(x))$$

3. 得到最终的分类器：

$$f(x) = \text{sign}\left(\sum_{j=1}^{m} \alpha_j C_j(x)\right)$$

3.4.2 模型评价指标

在机器学习的分类任务中，评价模型性能的指标至关重要。常见的分类任务评价指标包括准确率(Accuracy)、混淆矩阵(Confusion Matrix)、精度(Precision)、召回率(Recall)、F1-score 和支持度(Support)。

(1) 准确率：准确率是最基本的分类评价指标，表示模型预测正确的样本数占总样本数的比例。

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.23}$$

其中，TP(True Positive)指被正确分类为正类的样本数；TN(True Negative)指被正确分类为负类的样本数；FP(False Positive)指被误分类为正类的样本数；FN(False Negative)指被误分类为负类的正样本数。尽管准确率直观易懂，但在类别不均衡的数据集上，准确率可能会产生误导。例如，在极端情况下，如果 99%的样本属于某一类别，则一个简单的全零分类器(始终预测该类别)也能达到 99%的准确率，但它实际上毫无作用。

(2) 混淆矩阵：混淆矩阵提供了分类结果的详细分布，以矩阵的形式展现了真实类别和预测类别之间的关系。例如，二分类任务的混淆矩阵如表 3.2 所示。

表 3.2 二分类任务的混淆矩阵

	预测正类	预测负类		预测正类	预测负类
真实正类	TP	FN	真实负类	FP	TN

混淆矩阵能直观地展示模型在哪些类别上表现较好或较差，有助于针对特定类别优化模型。

(3) 精度：精度衡量的是被预测为正类的样本中实际为正类的比例。

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.24}$$

高精度意味着模型在预测正类时很谨慎，减少了误报(FP)，这在欺诈检测、医疗诊断等任务中尤为重要。

(4) 召回率：召回率衡量的是所有真实正类样本中被正确分类的比例。

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.25}$$

高召回率表示模型能识别更多的正类样本，但可能会带来更多的误报(FN)。在疾病检测等任务中，召回率较低可能意味着严重的后果，因此通常更关注召回率。

(5) F1-score：F1-score 是精度和召回率的调和平均数，用于在两者之间取得平衡。

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.26}$$

F1-score 的取值范围在 0 到 1，值越大，表示模型的综合性能越好。对于类别不均衡的任务，F1-score 比单独的准确率更能反映模型的真实性能。

(6) 支持度：支持度表示每个类别中实际存在的样本数，即混淆矩阵中每行的总和。它反映了数据分布的情况，有助于理解模型在不同类别上的表现。

3.4.3　Adaboost 算法实践

本节介绍利用 Scikit-learn 库中的 Adaboost 类，为勒索软件数据集构建一个用于二分类问题的 Adaboost 模型。数据集中包含 138047 条样本，每条样本具有 56 个特征，并通过一个名为 legitimate 的标签字段标注样本是否为合法文件(1 表示合法文件，0 表示非法文件)。

Scikit-learn(简称为 sklearn)是一个用于数据挖掘和数据分析的 Python 库，构建在 NumPy、Scipy 和 Matplotlib 基础之上。它提供了一套高效且全面的工具集，涵盖广泛的机器学习任务，包括分类、回归、聚类、降维、模型选择和预处理。Scikit-learn 的功能强大且易于使用，通过简洁的一致性 API，用户可以轻松实现复杂的机器学习模型构建和评估，其高效的算法实现和丰富的文档支持，使其成为数据科学和机器学习领域的标准库，广泛应用于学术研究和工业项目中。无论是初学者还是专业研究人员，Scikit-learn 都能提供强大的功能和灵活的解决方案，以应对各种数据分析和建模需求。

利用 Scikit-learn 封装的 AdaBoostClassifier 构建 Adaboost 模型，该分类器通过组合多个弱分类器(默认是决策树)来提升整体预测性能。常用的参数主要如下。

(1) n_estimators：弱分类器的数量。增加弱分类器的数量通常可以提升模型的表现，但也可能导致训练时间增加。

(2) learning_rate：每个弱分类器的权重缩减系数，用于平衡各个弱分类器在最终模型中的影响。较低的学习率通常需要更多的迭代次数。

(3) base_estimator：弱分类器的基模型，默认是决策树桩，也可以根据需求进行调整。

```
#构建 Adaboost 分类器
ada_clf =AdaBoostClassifier(
    n_estimators=50,          #弱分类器数量
    learning_rate=1.0,        #学习率
    algorithm='SAMME.R',      #选择 SAMME.R 算法，默认情况下
    random_state=42
)
#训练模型
ada_clf.fit(X_train, y_train)
#在测试集上进行预测
y_pred =ada_clf.predict(X_test)
```

在训练完成后，使用混淆矩阵、分类报告以及准确率评估 Adaboost 模型在勒索软件数据集上的表现。

```
#计算并打印准确率
accuracy =accuracy_score(y_test, y_pred)
print("测试集准确率：", accuracy)

#打印分类报告
report =classification_report(y_test, y_pred)
```

```
print("分类报告：\n", report)
#绘制混淆矩阵
cm =confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] =False
plt.xlabel("预测标签")
plt.ylabel("真实标签")
plt.title("混淆矩阵")
plt.show()
```

运行以上代码后，可以直观地了解 Adaboost 模型在区分良性和恶性样本方面的效果（图 3.10 和图 3.11）。

测试集准确率： 0.9879512254014247
分类报告：

	precision	recall	f1-score	support
0	0.99	0.99	0.99	28934
1	0.98	0.98	0.98	12481
accuracy			0.99	41415
macro avg	0.99	0.99	0.99	41415
weighted avg	0.99	0.99	0.99	41415

图 3.10　Adaboost 模型的测试准确率与分类报告

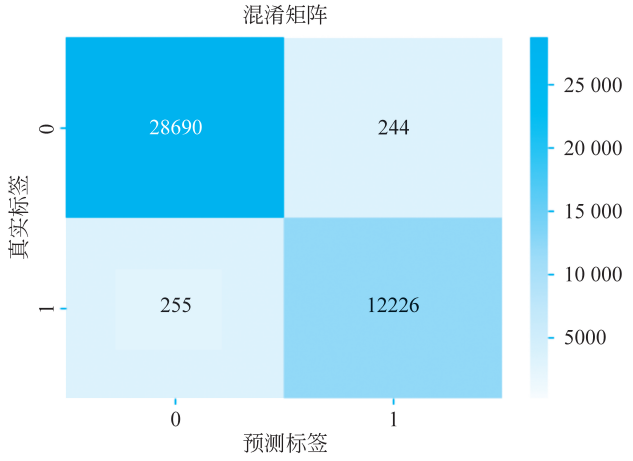


图 3.11　Adaboost 模型的混淆矩阵

3.4.4　Adaboost 算法的优缺点

Adaboost 具有自适应性和高效性，通过迭代训练弱分类器并动态调整样本权重，将多个弱学习器整合为强学习器，提升模型准确性和泛化能力。其算法简单易实现，能够重点关注难分类的样本，逐步降低训练误差，尤其适用于特征明显的分类任务。

然而，Adaboost 对噪声和异常值敏感，容易过拟合，同时训练过程需要依次训练多个弱分类器，计算开销较大，处理大规模数据时效率受限。此外，模型效果依赖于弱分类器的性能，若弱分类器较弱或多样性不足，则整体模型效果可能受限。

3.4.5　Random Forest 原理

随机森林通过集成多棵决策树，在对样本随机抽样的同时，还对数据特征进行随机抽取和组合，降低过拟合风险，提升稳定性和泛化能力，并具备特征重要性评估功能。

随机森林算法的主要步骤如下。

（1）Bootstrap 采样：采用自助抽样法从原始数据集中随机抽取出多个样本子集，每个子集等规模且有放回抽样，导致部分样本被重复选取，一些未被选中。这样，每棵决策树在不同数据集上训练，以提高模型多样性。

（2）特征子集选择：在构建每棵决策树的过程中，为进一步降低树与树之间的相关性，随机森林在每个节点随机选取部分特征，再从中选择最佳分裂特征。这样既减少了计算开销，又增加了模型的随机性，有助于降低过拟合风险。

（3）决策树构建：对于每个样本子集，在随机选择的特征子集上构建浅层的决策树。通常使用传统的决策树算法（如 CART），而且无须对每棵树进行剪枝操作。

（4）集成预测：所有决策树构建完成后，通过集成这些弱学习器的分类结果加权投票，选择票数最多的类别作为最终预测；回归任务通常计算所有树预测值的平均值作为最终结果。

（5）模型评估与特征重要性：通过在每棵树中统计各个特征在节点分裂中的贡献，随机森林能够评估每个特征对整体模型预测的影响程度，从而为后续的特征选择和模型解释提供依据。

3.4.6　Random Forest 算法实践

本节将利用勒索软件数据集演示如何使用 Scikit-learn 中的 RandomForestClassifier 构建模型，并对模型进行训练和评估（图 3.12 和图 3.13）。

```
#构建随机森林模型
rf_clf =RandomForestClassifier(random_state=42)

#训练模型
rf_clf.fit(X_train, y_train)

#在测试集上进行预测
y_pred =rf_clf.predict(X_test)
```

利用分类报告和混淆矩阵对随机森林模型在测试集上的表现进行评估，从而验证模型的预测能力。

```
#计算准确率
accuracy =accuracy_score(y_test, y_pred)
print("测试集准确率：", accuracy)

#输出分类报告
report =classification_report(y_test, y_pred)
print("分类报告：\n", report)
#绘制混淆矩阵
cm =confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.rcParams['font.sans-serif']=['SimHei']
```