第十讲、集成学习

韩丽颖、周瀚旭、甘璧丞

2020 年 6 月 24 日

Summary

在第十讲中，我们主要介绍了集成学习，介绍了其思想以及一些相关的算法。在第一节中，我们主要介绍了集成学习的动机。第二节中，我们主要讲述了 AdaBoost 算法，介绍了指数损失函数，以及 α_t 和 D_t 的推导。第三节中，我们讲述了以 bagging 和随机深林算法为例的并行方法。第四节中，我们讲了平均法、投票法以及学习法的结合策略。第五节中，我们介绍了误差与多样性，介绍了他们之间的关系以及具体的推导过程。第六节中，我们从四种扰动的角度出发，介绍了如何增加多样性。第七节中，我们对课堂上的常见问题进行了整理归纳。

目录

Chapter 1

集成学习

1.1 Motivation

集成学习（ensemble learning），一般而言是指通过构建并结合多个学习器来完成学习任务的系统。集成学习的一般结构：先产生一组“个体学习器”，再用某种策略将它们结合起来。个体学习器通常由一个现成的学习算法从训练数据产生，比如我们之前学习过的神经网络，决策树等等。

集成学习通过多个学习器进行结合，常可获得比单一学习器显著卓越的泛化性能。这对“弱学习器”（weak learner，通常指泛化性能略优于随即猜测的学习器；比如在二分类问题上精度略高于 50% 的分类器）尤为明显，但在实际中，我们往往希望通过比较少的个体学习器，或是重用关于常见学习器的一些经验等，会使用比较强的学习器。

在一般经验中, 如果把好坏不等的东西渗到一起, 那么通常结果会是比最坏的要好一些, 比最好的要坏一些. 集成学习把多个学习器结合起来, 如何能获得比最好的单一学习器更好的性能呢?

下面考虑一个简单的例子: 在二分类任务中, 假定三个分类器在三个测试样本上的表现如下图所示, 其中 $\surd$ 表示分类正确, $\times$ 表示分类错误, 集成学习的结果通过投票法 (voting) 产生, 即“少数服从多数”。在图 1.1(a) 中, 每个分类器都只有 66.6% 的精度, 但集成学习却达到了 100%; 在图 1.1(b) 中, 三个分类器没有差别, 集成之后性能没有提高; 在图 1.1(c) 中, 每个分类器的精度都只有 33.3%, 集成学习的结果变得更糟。这个简单的例子显示出: **要获得好的集成，个体学习器应“好而不同”，即个体学习器要有一定的“准确性”，即学习器不能太坏，并且要有“多样性”(diversity)，即学习器间具有差异**。

接下来，我们对集成学习分类器的总错误率做一些简单的分析。考虑二分类问题 $y \in$

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
集成	√	√	√

(a) 集成提升性能

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	√	√	×
h_3	√	√	×
集成	√	√	×

(b) 集成不起作用

	测试例1	测试例2	测试例3
h_1	√	×	×
h_2	×	√	×
h_3	×	×	√
集成	×	×	×

(c) 集成起负作用

图 1.1: 集成个体应“好而不同”（h_i 表示第 i 个分类器）

$\{-1, +1\}$ 和真实函数 f, 假定基分类器的错误率为 ϵ, 即对每个基分类器，h_i 有

$$P\left(h_i(\boldsymbol{x}) \neq f(\boldsymbol{x})\right) = \epsilon \tag{1.1}$$

现在假设集成通过简单投票法结合了 T 个相同的分类器，若有超过半数的分类器是正确的，那么集成分类就是正确的，我们定义总学习器的学习能力为:

$$H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{T} h_i(\boldsymbol{x})\right) \tag{1.2}$$

假设每个学习器的错误率相互独立，则有 Hoeffding 不等式可知，集成的错误率为:

$$\begin{aligned} P(H(\boldsymbol{x}) \neq f(\boldsymbol{x})) &= \sum_{k=0}^{\lfloor T/2 \rfloor} \left(\begin{array}{c} T \\ k \end{array} \right) (1-\epsilon)^k \epsilon^{T-k} \\ &\leqslant \exp\left(-\frac{1}{2} T(1-2\epsilon)^2\right) \end{aligned} \tag{1.3}$$

根据上限，我们可以得出一个结论，只要 $\epsilon \neq \frac{1}{2}$，无论错误率高还是低，随着个体分类器 T 的增大，集成学习的错误率将指数级下降，最终趋于 0。结论看似实用性高，但这有一个重要的前提假设，也就是学习器之间要相互独立，在实际情况中，个体学习器之间的准确性和多样性本身存在一些冲突，而研究如何平衡两者，是后面学习的内容。

根据个体学习器的生成方式，可以将集成学习方法大致分为两大类，一种是**并行方法**：个体学习器之间不存在强依赖关系，可同时生成的并行化方法，典型例子是 **Bagging** 和**随机森林（Random Forest）**；还有一种是**序列化方法**：个体学习器之间存在强依赖关系，必须串行生成的序列化方法，典型例子是 **Boosting**。

1.2 AdaBoost 算法

首先介绍一下 Boosting 算法的原理。简单来说，Boosting 是一族可将弱学习器提升为强学习器的算法。他的工作机制如下：先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器；如此反复进行，直至基学习器数目实现达到指定的值 T，最终将这 T 个基学习器进行加权结合。

在 Boosting 一族算法中，最著名的代表是 AdaBoost 算法，我们通过 “加法模型”，也就是集成学习总学习器的学习能力是基学习器的线性组合。

如下图 (2。1) 所示，我们假设总共有 m 个数据，组成数据集 $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$，其中 $y_i \in \{-1, +1\}$，f 是真实函数从 x_i 映射到 y_i。我们假设有 T 个学习器，也就是训练 T 轮。注意在第一轮的时候，我们不具备任何关于样本的信息，对第一个学习器，我们采用的样本权重也就是 $\frac{1}{m}$。在此后的每一轮迭代，我们首先需要基于上一轮得到的分布 D_t 从数据集 D 中训练出分类器 h_t。然后估计 h_t 的误差。如果我们此时发现训练的学习器错误率竟然高于 50%，这说明该分类器有可能出现模型误差，或者数据误差过大等因素，在排查过后再重新训练该学习器。最后我们通过误差更新样本分布 D_{t+1}，以便下次迭代。最终输出个体学习器的线性组合。

在上述算法中，我们首先要注意到采用何种损失函数直接决定了学习器的错误率，同时也注意其他两点：一是每个个体学习器的最终权重 α_t 是如何计算的?；二是每一轮更新迭代的 $D_{t+1}(x)$ 是如何更新的? 下面我们将对三个部分进行逐一解析。

1.2.1 指数损失函数

在 AdaBoost 算法中，我们采用的是指数损失函数 (exponential loss function)，他的具体形式为：

$$\ell_{\exp}(H \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H(\boldsymbol{x})}\right] \tag{1.4}$$

其中 $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}$ 表示这是 x 遵循 D 分布的一个概率密度函数。若集成学习的总学习能力函数 $H(x)$ 能够使该损失函数最小化，我们只需要考虑 (2.1) 式对 $H(x)$ 的偏导等于零：

$$\frac{\partial \ell_{\exp}(H \mid \mathcal{D})}{\partial H(\boldsymbol{x})} = -e^{-H(\boldsymbol{x})}P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) + e^{H(\boldsymbol{x})}P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) = 0 \tag{1.5}$$

我们不难发现，最终 $H(x)$ 应该满足：

$$H(\boldsymbol{x}) = \frac{1}{2}\ln\frac{P(f(x) = 1 \mid \boldsymbol{x})}{P(f(x) = -1 \mid \boldsymbol{x})} \tag{1.6}$$

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad h_t = \mathfrak{L}(D, \mathcal{D}_t)$;
4: $\quad \epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;
5: $\quad$ **if** $\epsilon_t > 0.5$ **then break**
6: $\quad \alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$;
7: $\quad \mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
$= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$
8: **end for**

输出： $H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$

图 1.2: AdaBoost 算法

这等价于：

$$\begin{aligned} \operatorname{sign}(H(\boldsymbol{x})) &= \operatorname{sign}\left(\frac{1}{2} \ln \frac{P(f(x) = 1 \mid \boldsymbol{x})}{P(f(x) = -1 \mid \boldsymbol{x})} \right) \\ &= \begin{cases} 1, & P(f(x) = 1 \mid \boldsymbol{x}) > P(f(x) = -1 \mid \boldsymbol{x}) \\ -1, & P(f(x) = 1 \mid \boldsymbol{x}) < P(f(x) = -1 \mid \boldsymbol{x}) \end{cases} \\ &= \underset{y \in \{-1, 1\}}{\arg\max} P(f(x) = y \mid \boldsymbol{x}) \end{aligned} \tag{1.7}$$

换而言之，当我们发现使得 $f(x) = 1$ 的样本数据 x_i 较多时，我们应该合理推测最终学习到的函数 $H(x)$ 大于 0，相似的，当我们发现使得 $f(x) = -1$ 的样本数数据 x_j 较多时，我们应该推测 $H(x)$ 小于 0。这意味着 $sign(H(x))$ 达到了贝叶斯最优错误率；若指数损失函数最小化，则分类错误率也将最小化。由于指数损失函数相比于一般的 0/1 损失函数具有更好的数学性质，比如连续可微，因此我们采用 exponential loss function 计算损失。

1.2.2 推导 α_t

在 AdaBoost 算法中，除了第一个分类器 h_1 是通过直接按照平均分布的样本进行学习产生的；后续在 h_t 基于 D_t 产生之后，该基分类器的权重 α_t 应使得 $\alpha_t h_t$ 最小化指数损失函数：

$$\begin{aligned}\ell_{\exp}\left(\alpha_t h_t \mid \mathcal{D}_t\right) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-\alpha_t}\mathbb{I}\left(f(\boldsymbol{x})=h_t(\boldsymbol{x})\right)+e^{\alpha_t}\left[f(\boldsymbol{x})\neq h_t(\boldsymbol{x})\right)\right] \\ &= e^{-\alpha_t}P_{x\sim D_t}\left(f(\boldsymbol{x})=h_t(\boldsymbol{x})\right)+e^{\alpha_t}P_{x\sim D_t}\left(f(\boldsymbol{x})\neq h_t(\boldsymbol{x})\right) \\ &= e^{-\alpha_t}\left(1-\epsilon_t\right)+e^{\alpha_t}\epsilon_t\end{aligned} \tag{1.8}$$

其中，$\epsilon_t = P_{\boldsymbol{x}\sim\mathcal{D}_t}\left(h_t(\boldsymbol{x})\neq f(\boldsymbol{x})\right)$，也即是第 t 轮迭代时，真实函数与该学习器学习到的数据不同的概率。为了使得损失函数最小化，同样的，我们对其求偏导：

$$\frac{\partial\ell_{\exp}\left(\alpha_t h_t \mid \mathcal{D}_t\right)}{\partial\alpha_t} = -e^{-\alpha_t}\left(1-\epsilon_t\right)+e^{\alpha_t}\epsilon_t = 0 \tag{1.9}$$

由此，我们不难得到：

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{1.10}$$

这既是图 2.1 中的分类器的权重更新公式。

1.2.3 推导 D_{t+1} 与 D_t 的关系

在前面的讨论中，我们知道每一轮的样本分布 D_t，是基于当前的学习器学习能力进行调整的，所以我们要首先观察每一轮的学习器 h_t 是如何学习调整的。再去考虑每轮样本权重的迭代关系。

在上述算法中，我们知道每一个 h_t 应该偏向于纠正前一轮迭代得到的学习能力函数 H_{t-1}，换而言之，偏向于更正与真实数据不符合的值。最理想的情况下，h_t 应该可以纠正 H_{t-1} 的全部错误，即最小化损失函数的 h_t：

$$\begin{aligned}min\ell_{\exp}\left(H_{t-1}+h_t \mid \mathcal{D}\right) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})\left(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x})\right)}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}\right]\end{aligned} \tag{1.11}$$

在式（2.8）中，注意到 $f^2(\boldsymbol{x}) = h_t^2(\boldsymbol{x}) = 1$，我们可以利用一阶泰勒展开近似 $e^{-f(x)h_t(x)}$，得到：

$$\begin{aligned}\ell_{\exp}\left(H_{t-1}+h_t \mid \mathcal{D}\right) &\simeq \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{f^2(\boldsymbol{x})h_t^2(\boldsymbol{x})}{2}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{1}{2}\right)\right]\end{aligned} \tag{1.12}$$

因此，理想的学习器 h_t 应该使得：

$$
\begin{aligned}
h_t(\boldsymbol{x}) &= \arg\min_h \ell_{\exp}\left(H_{t-1}+h \mid \mathcal{D}\right) \\
&= \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h(\boldsymbol{x})+\frac{1}{2}\right)\right] \\
&= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}f(\boldsymbol{x})h(\boldsymbol{x})\right] \\
&= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}f(\boldsymbol{x})h(\boldsymbol{x})\right].
\end{aligned}
$$

注意到 $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]$ 是一个概率的固定常数，我们可以替换上式：

$$
\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]} \tag{1.13}
$$

根据数学期望的定义以及 D_{t+1} 与 D_t 的关系，我们可以得到：

$$
\begin{aligned}
\mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\
&= \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\
&= \mathcal{D}_t(\boldsymbol{x})\cdot e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\frac{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]}
\end{aligned} \tag{1.14}
$$

这既是图（2.1）中样本分布的更新公式。

1.2.4 小结

我们从基于线性模型迭代式指数损失函数的角度，推导出了图（2.1）的 AdaBoost 算法。

前面的推导过程中，我们可以发现函数的最终输出是离散的，如果我们改变输出使其变为概率，则产生了 Real AdaBoost，它的推导和前者类似。如果我们替换指数损失函数呢？Gradient Boosting! AdaBoost 采用的是增加上一轮学习错误样本的权重的策略，而在 Gradient Boosting 中则将负梯度作为上一轮基学习器犯错的衡量指标，在下一轮学习中通过拟合负梯度来纠正上一轮犯的错误。

1.3 并行方法

1.3.1 bagging

bootstrapping samling（自主采样法）：

我们现在有 m 个样本，每次随机抽取一个样本，记录下来，再把样本放回去；接着再随机抽取一个样本，记录下来，把样本放回去... 一共进行 m 此采样。

以三个样本为例，如下：

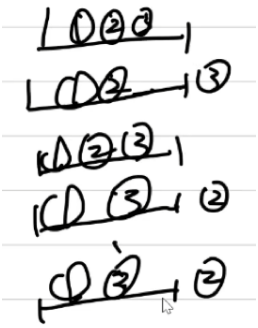


图 1.3: 三个样本为例

那么，每次每个样本都有 $\frac{1}{m}$ 的概率被采到。我们一共采了 m 次样本，则一个样本不被采到的概率:

$$P_m = (1 - \frac{1}{m})^m$$

$$\lim_{m \to \infty} P_m = \frac{1}{e} \approx 0.368$$

因此，通过自助采样，初始数据集 D 中约有 0.368 的样本未出现在采样数据集 D′ 中。

我们可将 D′ 用作训练集，D\D′ 用作验证集。

这样我们有数据集总量约 $\frac{1}{3}$ 的，没在训练集中出现的样本用于验证。这样的验证结果，亦称“包外估计”（out-of-bag estimate），H^{oob} : 0.368。

训练集被两个部分，分别用于训练和验证，测试集则用来测试。在训练的时候进行验证，能有效的防止过拟合的现象。

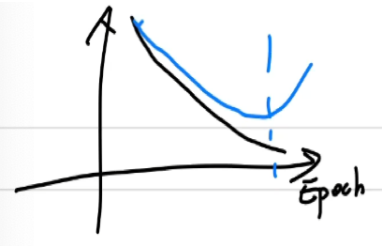


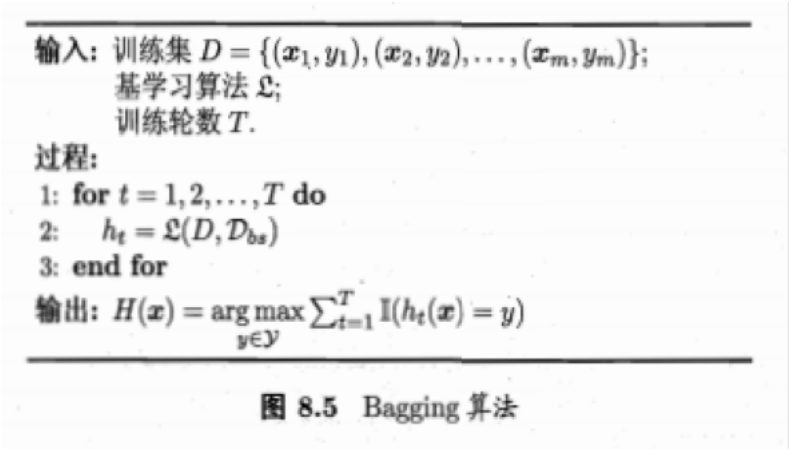
图 1.4: 验证集的作用

如上图，我们通过在验证集上 loss 的最低值时停止训练，能有效地防止过拟合现象的发生。

Bagging 算法：

基本流程：基于自助采样法，我们可以采样出 T 个含 m 个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习器进行结合。

输入：$D = \{(x_i, y_i)\}_{i=1}^{m}$。



输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $\quad h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: **end for**

输出： $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$

图 8.5 Bagging 算法

图 1.5: Bagging 算法

D_{bs} 是自助采样产生的样本分布

复杂度：假设基学习器的计算复杂度为 O(m)，则 Bagging 的复杂度大致为 T(O(m)+O(s))，考虑到采样与投票/平均过程的复杂度 O(s) 很小。因此，其复杂度约为 T · O(m)，与直接使

用基学习算法训练一个学习器的复杂度同阶。

由于每个基学习器只使用了初始训练集中约 0.632 的样本，剩下约 0.368 的样本可用作验证集来对泛化性能进行“包外估计”(out-of-bag estimate)。

令 D_t 表示 h_t 实际使用的训练样本集，令 $H^{oob}(x)$ 表示对样本 x 的包外预测，即仅考虑那些未使用 x 训练的基学习器上的预测，有：

$$H^{oob}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} I(h_t(x) = y) I(x \notin D_t)$$

Bagging 泛化误差的包外估计为：

$$\varepsilon^{oob} = \frac{1}{|D|} \sum_{(x,y) \in D} I(H^{oob}(x) \neq y)$$

1.3.2 Random Forest 随机深林

传统决策树在选择划分属性时是在当前结点的属性集合（假定有 d 个属性）中选择一个最优属性；在 RF 中，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优属性用于划分。

一般情况下，推荐值 $K = log_2 d$。

随机森林的收敛性与 Bagging 相似。随机森林的起始性能往往相对较差，特别是在集成中只包含一个基学习器时。因为通过引入属性扰动，随机森林中个体学习器的性能往往有所降低。然而，随着个体学习器数目的增加，随机森林通常会收敛到更低的泛化误差。随机森林的训练效率常优于 Bagging，因为在个体决策树的构建过程中，Bagging 使用的是“确定型”决策树，在选择划分属性时要对结点的所有属性进行考察，而随机森林使用的“随机型”决策树则只需考察一个属性子集。

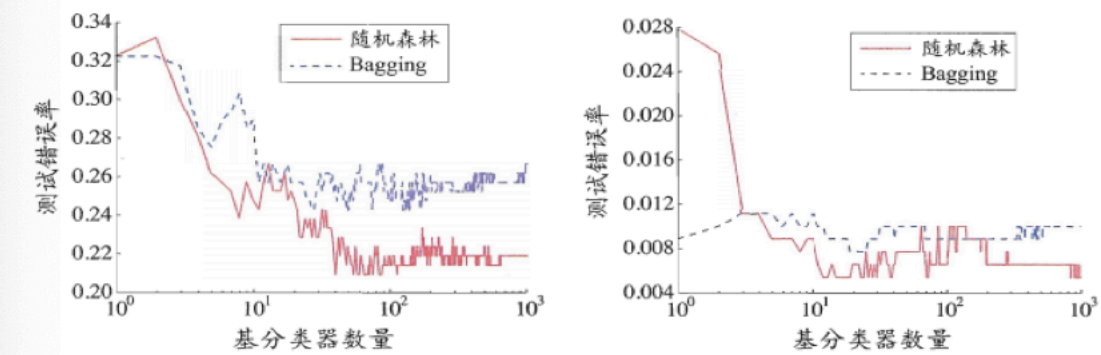


图 8.7 在两个 UCI 数据上，集成规模对随机森林与 Bagging 的影响

图 1.6: 集成规模对随机深林与 Bagging 的影响

1.4 结合策略

类别标记集合 $\{c_1, c_2, ..., c_N\}$，基学习器 $\{h_1, h_2, ..., h_T\}$

h_i 在样本 x 上的预测输出表示为一个 N 维向量 $(h_i^1(x), h_i^2(x), ..., h_i^N(x))$，其中 $h_i^j(x)$ 是 h_i 在类别标记 c_j 上的输出。

1.4.1 平均法

1. 简单平均法:

$$H(x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

2. 加权平均法:

$$H(x) = \sum_{t=1}^{T} w_t h_t(x), w_t \geq 0, \sum_{t=1}^{T} w_t = 1$$

1.4.2 投票法

1. 绝对多数投票法：

$$H(x) = \begin{cases} c_j, & if \ \sum_{i=1}^{T} h_i^j(x) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(x), \\ reject, & otherwise. \end{cases}$$

2. 相对多数投票法：

$$H(x) = c_{\arg\max_j \sum_{i=1}^{T} h_i^j(x)}$$

3. 加权投票法

$$H(x) = c_{\arg\max_j \sum_{i=1}^{T} w_i h_i^j(x)}, w_i \geq 0, \sum_{i=1}^{T} w_i = 1$$

在现实任务中，不同类型个体学习器可能产生不同类型的 $h_i^j(x)$ 值，常见有：

–类标记：$h_i^j \in \{0, 1\}$，若 h_i 将样本 x 预测为类别 c_j 则取值为 1，否则为 0。使用类标记的投票亦称 “硬投票”。

–类概率：$h_i^j \in [0, 1]$，相当于对后验概率 $P(c_j|x)$ 的一个估计，使用类概率的投票亦称 “软投票”。

不同类型的 h_i^j 值不能混用。

1.4.3 学习法

Stacking 是学习法的典型代表，这里我们把个体学习器称为初级学习器，用于结合的学习器称为次级学习器或元学习器。

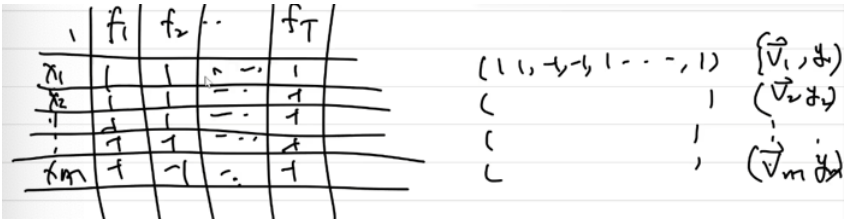
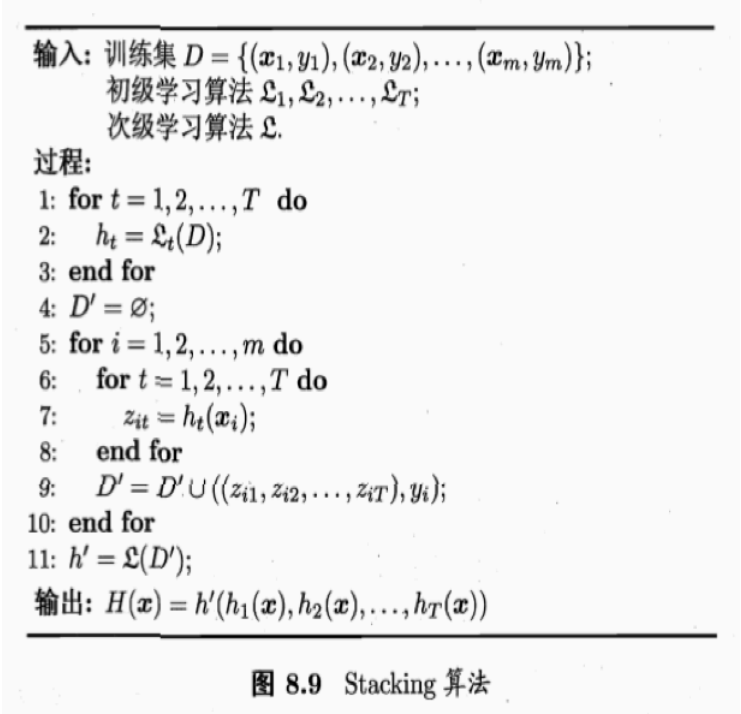


图 1.7: 初级学习器与次级学习器

在这里我们以 EOCO 作为初级学习器，得到一组向量 $\{v_1, v_2, ..., v_m\}$，再用次级学习器，学习数据集 $\{(v_i, y_i)\}_{i=1}^{m}$。

Stacking 的算法如下：



输入：训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
初级学习算法 $\mathfrak{L}_1,\mathfrak{L}_2,\ldots,\mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.

过程：

1: **for** $t=1,2,\ldots,T$ **do**
2: $h_t=\mathfrak{L}_t(D)$;
3: **end for**
4: $D'=\varnothing$;
5: **for** $i=1,2,\ldots,m$ **do**
6: **for** $t=1,2,\ldots,T$ **do**
7: $z_{it}=h_t(\boldsymbol{x}_i)$;
8: **end for**
9: $D'=D'\cup((z_{i1},z_{i2},\ldots,z_{iT}),y_i)$;
10: **end for**
11: $h'=\mathfrak{L}(D')$;

输出：$H(\boldsymbol{x})=h'(h_1(\boldsymbol{x}),h_2(\boldsymbol{x}),\ldots,h_T(\boldsymbol{x}))$

图 8.9 Stacking 算法

图 1.8: Stacking 算法

1.5 误差与多样性

假定我们用个体学习器 $h_1,h_2,...,h_T$ 通过加权平均法结合产生的集成来完成回归学习任务 $f:R^d\longrightarrow R$。对事例 x，定义学习器 h_i 的“分歧”为（多样性）：

$$A(h_i|x)=(h_i(x)-H(x))^2$$

则集成的“分歧”是：

$$\begin{aligned}\bar{A}(h|x)&=\sum_{i=1}^{T}w_iA(h_i|x)\\&=\sum_{i=1}^{T}w_i(h_i(x)-H(x))^2.\end{aligned}$$

这里的“分歧”项表征了个体学习器在样本 x 上的不一致性，即在一个程度上反映了个体学习器的多样性。

个体学习器 h_i 和集成 H 的平方误差分别为：

$$E(h_i|x) = (f(x) - h_i(x))^2$$

和

$$E(H|x) = (f(x) - H(x))^2$$

令 $\bar{E}(h|x) = \sum_{i=1}^{T} w_i E(h_i|x)$ 表示个体学习器误差的加权均值，则有：

$$\begin{aligned}
\bar{A}(h|x) &= \sum_{i=1}^{T} w_i A(h_i|x) \\
&= \sum_{i=1}^{T} w_i (h_i(x) - f(x) + f(x) - H(x))^2 \\
&= \sum_{i=1}^{T} [w_i (h_i(x) - f(x))^2 + w_i (f(x) - H(x))^2 + 2w_i (h_i(x) - f(x))(f(x) - H(x))].
\end{aligned}$$

其中 $\sum w_i = 1, H(x) = \sum w_i h_i(x)$，因此有：

$$\begin{aligned}
\bar{A}(h|x) &= [\sum_{i=1}^{T} w_i (h_i(x) - f(x))^2 + w_i (f(x) - H(x))^2] + 2(H(x) - f(x))(f(x) - H(x)) \\
&= \sum_{i=1}^{T} w_i (h_i(x) - f(x))^2 - (f(x) - H(x))^2.
\end{aligned}$$

我们有：

$$\bar{A}(h|x) = \bar{E}(h|x) - E$$

亦即，泛化误差 = 个体误差-多样性：

$$E = \bar{E} - \bar{A}$$

因此，个体学习器准确性越高，多样性越大，则集成越好。

1.6 增加多样性

1.6.1 数据样本扰动

给定初始数据集，可从中产生出不同的数据子集，再利用不同的数据子集训练出不同的个体学习器。数据样本扰动通常是基于采样法，例如在 Bagging 中使用自助采样，在 AdamBoost 中使用序列采样。

数据样本扰动法对例如：决策树、神经网络等“不稳定基学习器”很有效（训练样本稍加变化就会导致学习器有显著变动）；但对例如线性学习器、支持向量机等稳定基学习器就不是很有效（他们对数据样本的扰动不敏感）。

1.6.2 输入属性扰动

即随机选取原属性空间的一个子空间来训练不同的基学习器。之前讲过的 Random Forest 就是通过输入属性扰动来获得比 Bagging 更加好的泛化性能的。但是，如果训练集中的属性维度少，用这种方法会使得单个基学习器的性能大大下降，最终集成学习器的泛化性能不一定有提升。

1.6.3 输出表示扰动

此类做法的基本思路是对输出表示进行操作以增强多样性。可对训练样本的类标记稍作变动，如“翻转法”随机改变一些训练样本的标记；也可对输出表示进行转化，如“输出调制法”将分类输出转化为回归输出后构建个体学习器；还可将原任务拆解为多个可同时求解的子任务，如 ECOC 法利用纠错输出码将多分类任务拆解为一系列二分类任务来训练基学习器。

1.6.4 算法参数扰动

基学习算法一般都有参数需要进行设置，例如神经网络的隐藏神经元数、初始连接权值等，通过随机设置不同的参数，往往可产生差别较大的个体学习器。例如：在卷积神经网络里面加“Dropout”，在训练过程中不改变某些参数的值。

1.7 课堂常见问题

1. Q：对准确性和多样性不是很理解

A：在现实任务中，个体学习器是为解决同一个问题训练出来的，他们显然不可能相互独立。事实上，个体学习器的准确性和多样性本身就存在冲突。一般的，准确性很高之后，要增加多样性就要牺牲准确性。如何产生“好而不同”的个体学习器，是集成学习研究的核心。

2. Q：D_{t+1} 与 D_t 的推导中，推导动机不理解，泰勒展开不理解

A：首先，我们要明确，对于 h_t，它的目标就是最小化指数损失函数，我们为了简化问题，在第 t 次迭代目标是建立在 t-1 次迭代优化完成的基础上的。那么对于单个数据点来说，我们需要做的就是一旦被正确分类，他的权值下降，一旦错误分类，权值上升。即，$D_t(X_i) = exp(-Y_i H_t(X_i))$.

在推导过程中，我们利用了数据的离散化的性质，也就是 $f^2(\boldsymbol{x}) = h_t^2(\boldsymbol{x}) = 1$，将 $e^{-f(x)h_t(x)}$ 进行近似处理，这是为了简化处理 $h_t(x)$ 与 $f(x)$ 相比较得到的错误率。

3. Q：Bootstrap 抽样和 Monte Carlo 抽样思想上的区别?

A: Bootstrap 抽样的基本思想是在全部样本未知的情况下，借助部分样本的有放回多次抽样，构建某个估计的置信区间。而，蒙特卡罗是一类随机算法的统称，其主要思想是采样越多，得到的结果越近似于最优解。更多的是从总体中抽一个样本，计算估计量（均值等），作为整体估计。

Monte Carlo 和 Bootstrap 是两种思想，都是基于 random sampling 去近似某一目标。Monte Carlo 的目标一般是一个难以计算的积分，bootstrap 的目标一般是统计推断。bootstrap 是从部分样本有放回的重采样 i 次（全部样本是未知的），将多次抽样的估计量（均值等）的分布作为整体的分布结果。而蒙特卡罗是在已知总体样本的情况下，不想计算全部值，就从中抽取一个样本（或多个），用这个抽取样本的估计量当做整体估计。